Mechatronics

Day 13 – 14

YOLO Object Detection and Coordinate Transformation

YOLO Custom Dataset Training

Pretrained YOLO models work well with common objects (COCO dataset), but for industrial applications, custom datasets are needed to detect specific objects like:

- Defective products in a manufacturing line.
- Electrical components in a circuit board.
- Machinery parts in an assembly line.

Custom Dataset Step

Step 1 : Prepare Dataset

Step 2 : Setup Environment

Step 3 : Train Model

Step 4 : Evaluate and Validate the Model

Step 5 : Export Model

Preparing a Custom Dataset

A dataset for YOLO needs:

1. **Images**: Collected from a real-world environment.
2. **Annotations**: Bounding boxes that label objects in the images.
3. **Dataset Management & Augmentation**
4. **Dataset Format**: Pascal VOC, COCO JSON, or YOLO TXT format.

Dataset Preparation Software for YOLO Custom Training

1. LabelImg
2. Computer Vision Annotation Tool (CVAT)
3. LabelMe
4. Roboflow

LabelImg

- **Free & Open-source**
- **Description:** A widely used graphical image annotation tool.
- **Features:**
  - Simple and easy to use.
  - Exports labels in YOLO format.
  - Supports Pascal VOC and COCO formats.
  - Works on Windows, macOS, and Linux.
- **Download:**
  - pip install labelImg
  - https://github.com/HumanSignal/labelImg

Computer Vision Annotation Tool (CVAT)

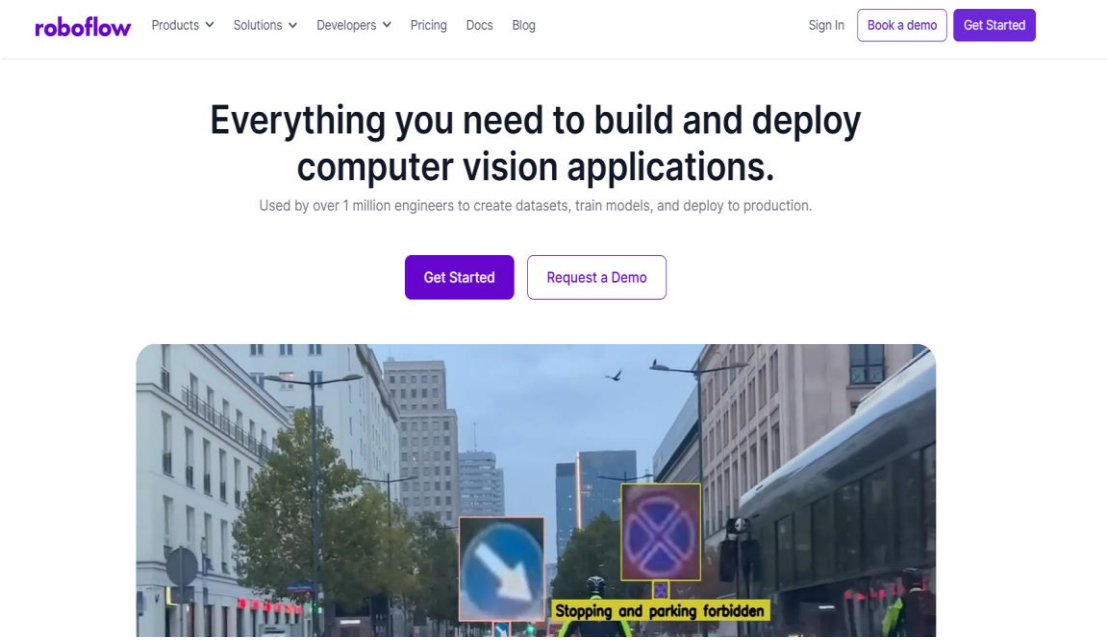
- **Description:** A web-based annotation tool developed by Intel.
- **Features:**
  - Supports bounding boxes, segmentation, and keypoints.
  - Can be deployed on a local server.
  - Collaborative labeling.
- **Website:** CVAT.org

LabelMe

- **Description:** Open-source labeling tool for object detection and segmentation.
- **Features:**
  - Can label polygons, rectangles, and lines.
  - Exports labels in JSON format (can be converted to YOLO format).
- **Download:**
  - pip install labelme
  - https://github.com/wkentaro/labelme

Roboflow

- **Free & Cloud-Based**
- **Description:** A cloud-based platform for dataset management and annotation.
- **Features:**
  - Auto-labeling support using AI.
  - Data augmentation tools.
  - Supports YOLO, COCO, Pascal VOC formats.
  - Dataset versioning and pre-processing.
- **Website:**
  - https://roboflow.com/



Dataset Augmentation Software for YOLO Custom Training

1. **FiftyOne**
2. OpenCV
3. Albumentations (Augmentation Library)
4. **Roboflow**

FiftyOne

Advanced Dataset Management

- **Description:** A powerful tool for visualizing, analyzing, and managing datasets.
- **Features:**
  - Supports YOLO, COCO, and VOC formats.
  - Data filtering and organization.
  - Python API for automation.
- **Installation** : pip install fiftyone

OpenCV

- **Description:** A computer vision library that helps in image augmentation, resizing, and manipulation.
- **Features:**
  - Image transformations: rotation, flipping, blurring.
  - Supports loading and saving YOLO datasets.
- **Installation** : pip install opencv-python
- **Website:** OpenCV.org

Albumentations

- **Description:** A fast image augmentation library for deep learning.
- **Features:**
  - Supports YOLO-compatible transformations.
  - Random flipping, rotation, and brightness adjustments.
- **Installation** : pip install albumentations
- **Website:** https://albumentations.ai/

Image Annotation and Augmentation in Roboflow for YOLO

- Roboflow simplifies **image annotation** and **augmentation** for training object detection models like **YOLO** (You Only Look Once)
- **Advantages of Using Roboflow**
  - ✓ Easy Web-Based Annotation.
  - ✓ Automatic Dataset Splitting.
  - ✓ Supports YOLO Format Export.
  - ✓ Dataset Augmentation for Better Accuracy.
  - ✓ Cloud Storage and Collaboration.

Steps to Create a Custom Dataset in Roboflow

Step 1: Collect and Upload Images

Step 2: Annotate Images

Step 3 : Annotate Verify and DataSet Splitting

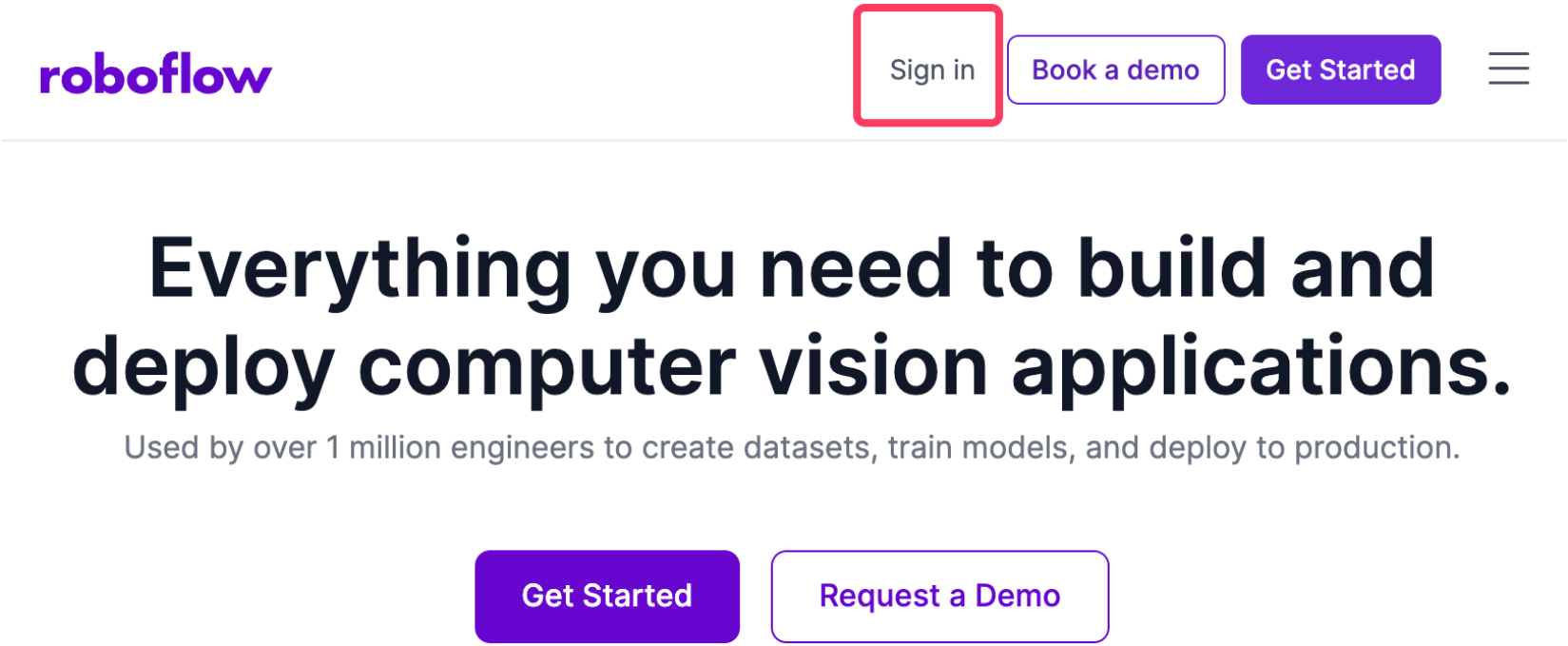
Step 4: Augment the Dataset (Optional)

Step 5: Export the Dataset for YOLOv11

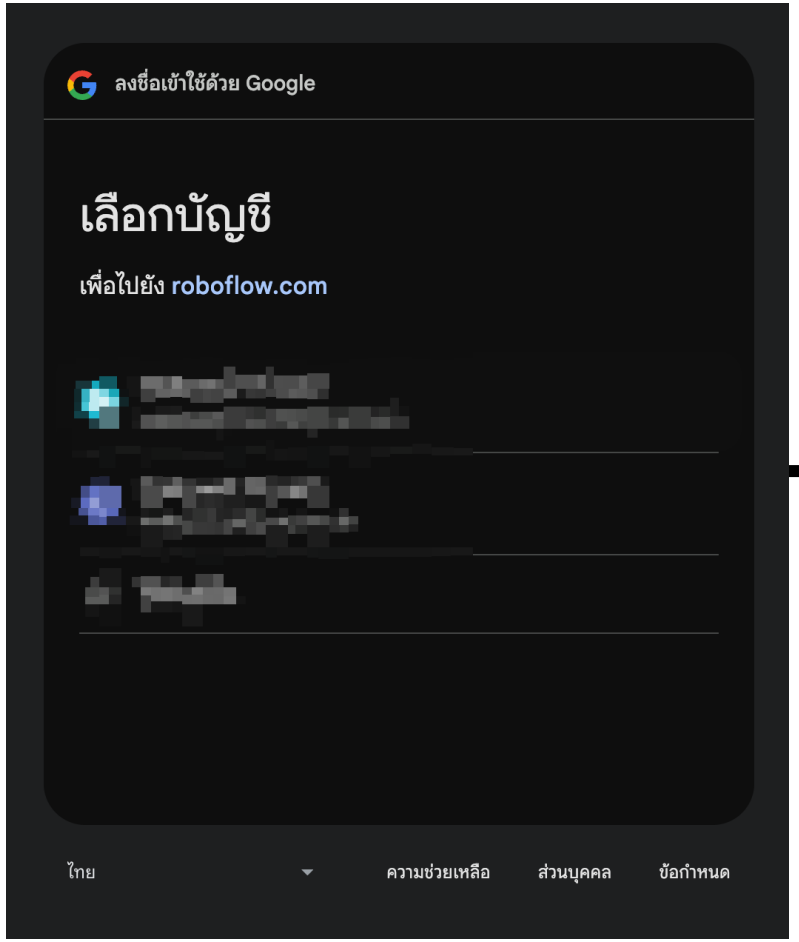
Step 1 : Collect and Upload Images

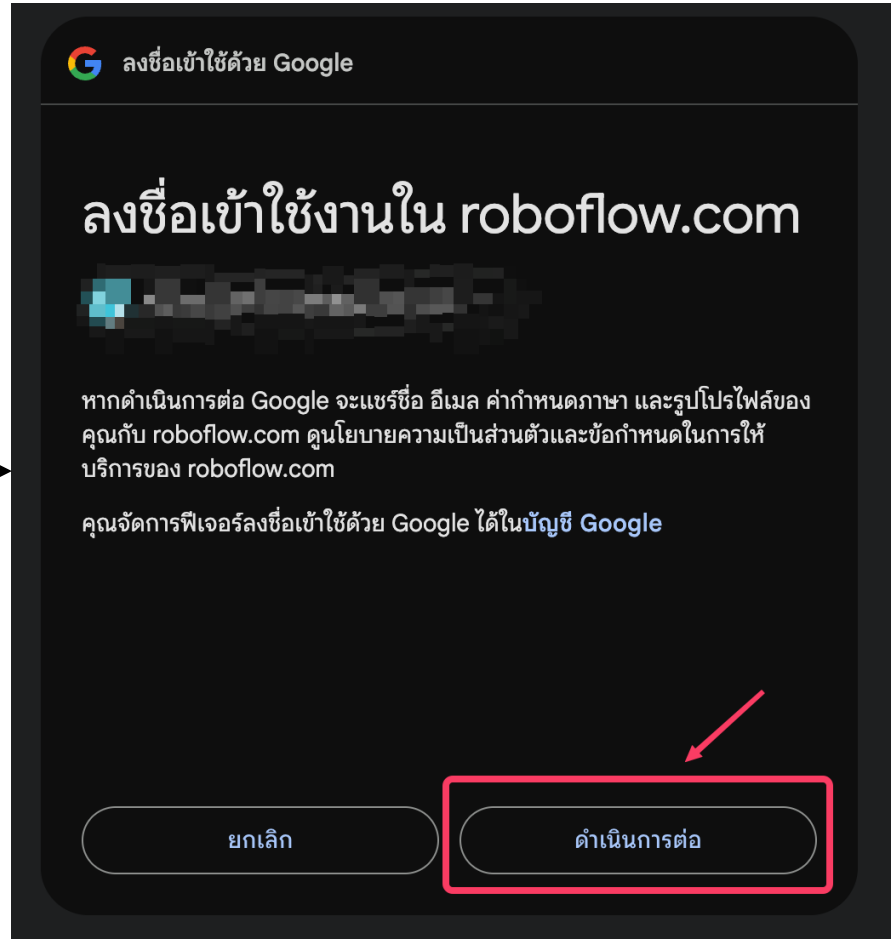
- **Go to Roboflow**: https://roboflow.com/
- **Sign in / Sign up** if you don't have an account.
- Click **"Create New Project"**, then:
  - Enter a **Project Name** (e.g., "Industrial Defect Detection").
  - Select **"Object Detection"** as the project type.
  - Click **"Create Project"**.
- Click **"Upload Images"** and select images from your local storage.

Roboflow: Sign up

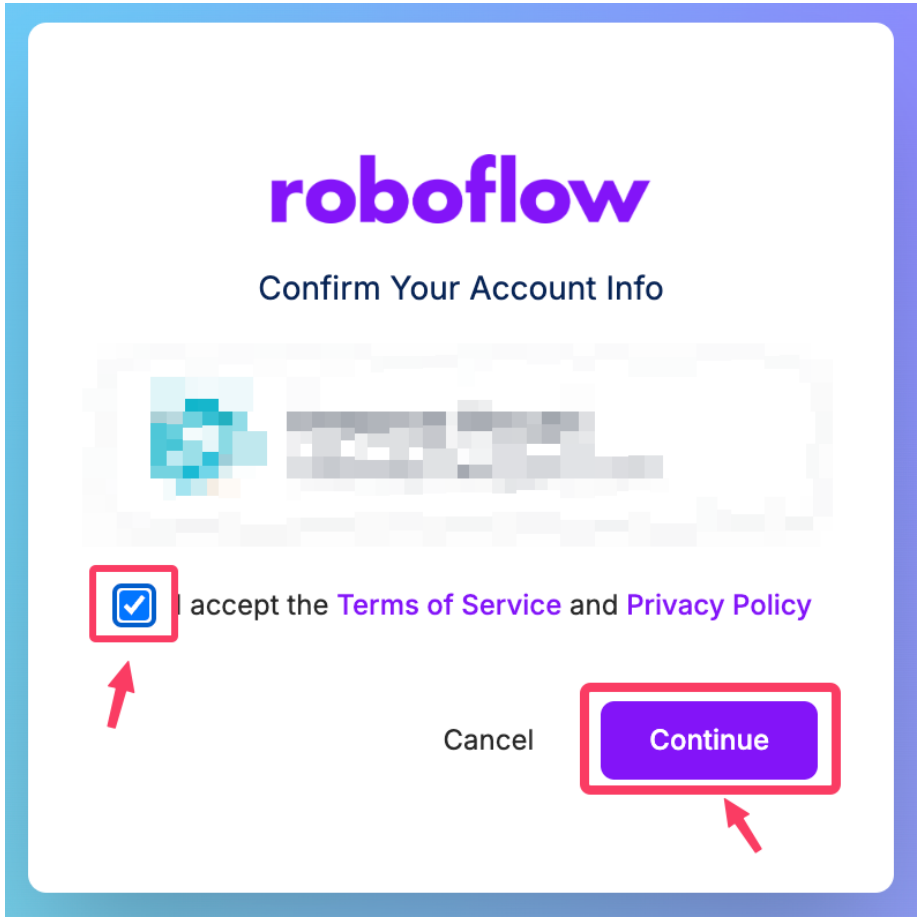


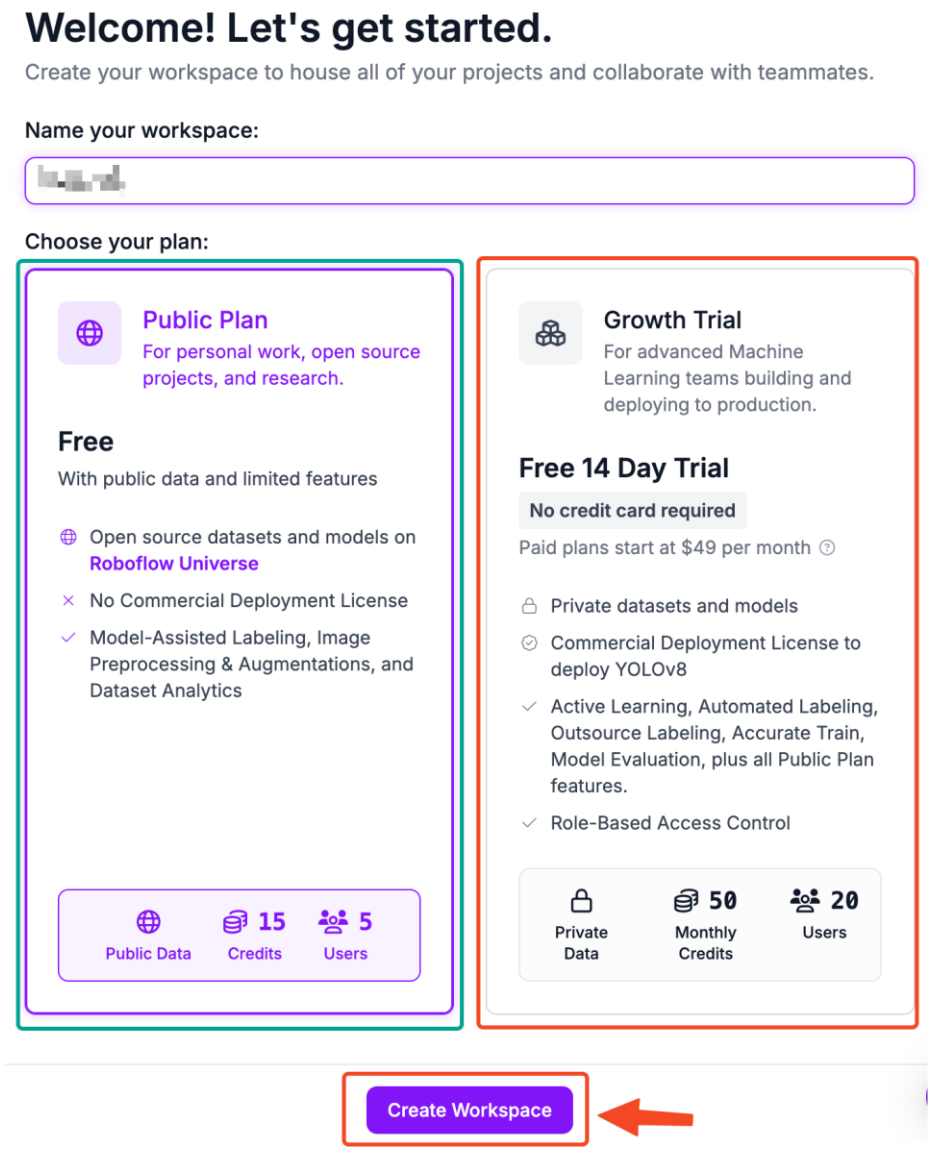
Roboflow: Sign up



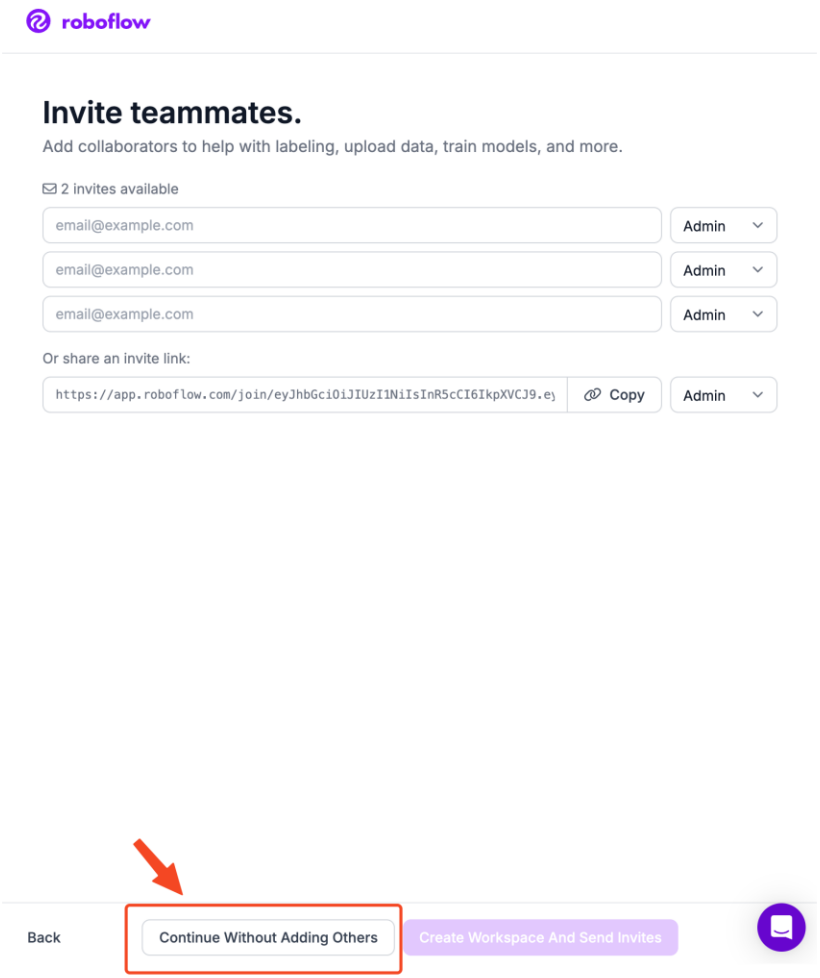


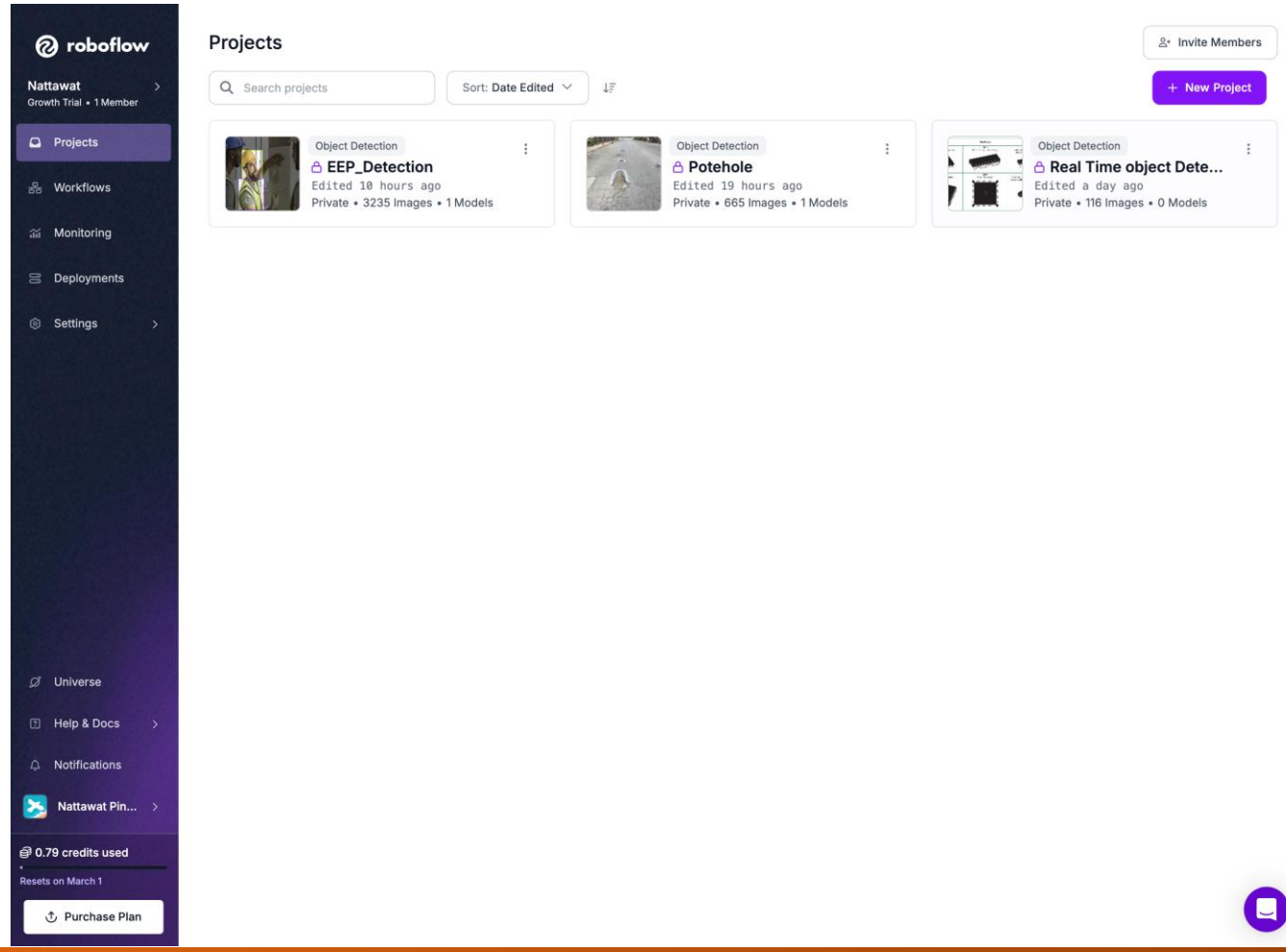
Roboflow : Sign up





Roboflow : Sign up





Step 2: Image Annotation (Bounding Box Labeling)

Uploading images :

1. Create Project and Upload Image

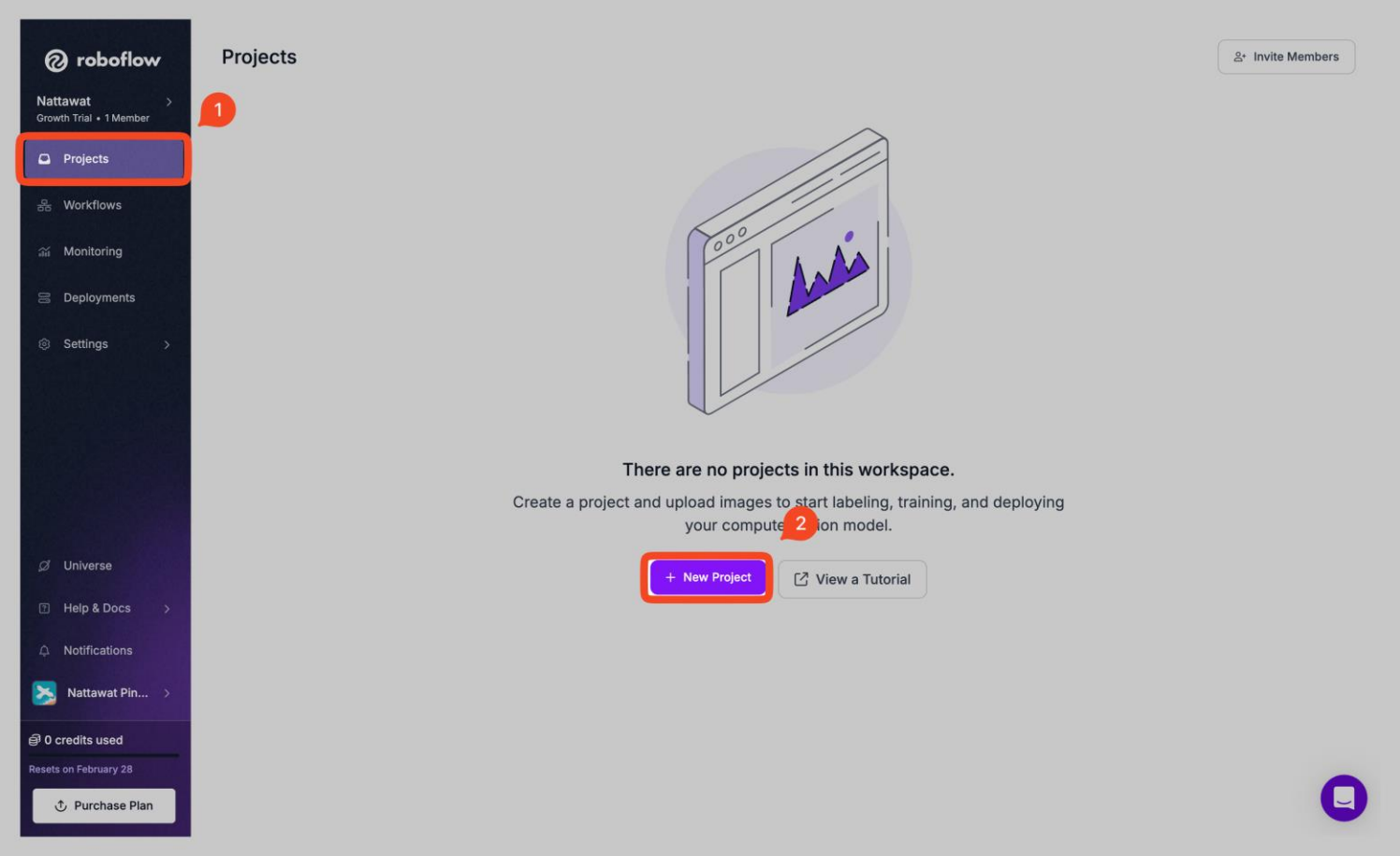
Manual Annotation

1. Click on **"Annotate"** to open the annotation tool.
2. Select an image to **start labeling**.
3. Draw bounding boxes:
4. Click **"Add Annotation"** and **drag** to draw a rectangle around an object.
5. Adjust the box to tightly fit the object.
6. Assign Class Labels:
7. After drawing a bounding box, **enter a label name** (e.g., "Screw", "Defect", "Circuit Board").
8. If you have multiple objects, repeat for each.
9. Click **"Save"** to store the annotations.

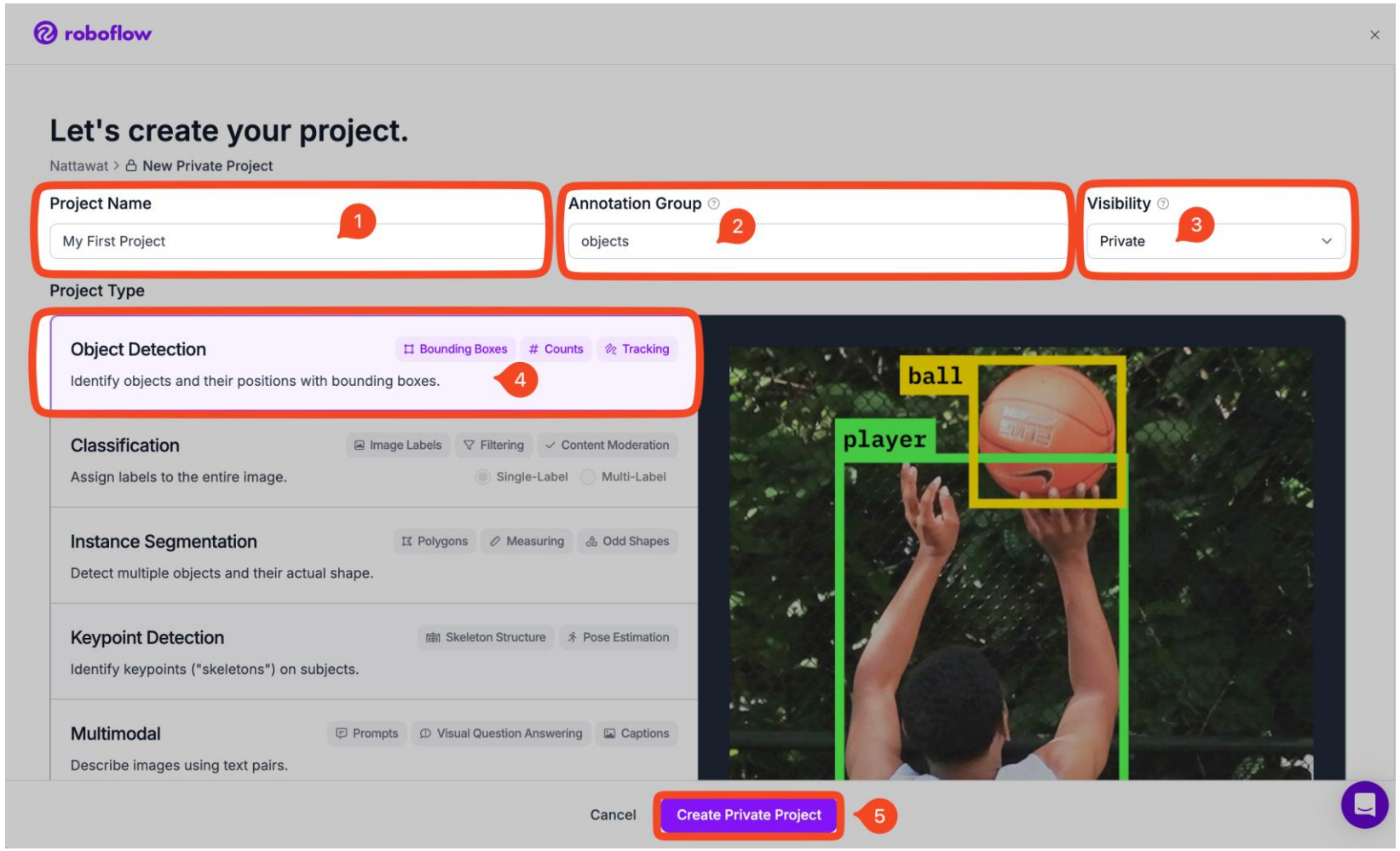
Collaborative Annotation

- You can **invite team members** to assist with annotation.
- Roboflow allows **multiple annotators** to work on the dataset **simultaneously**.

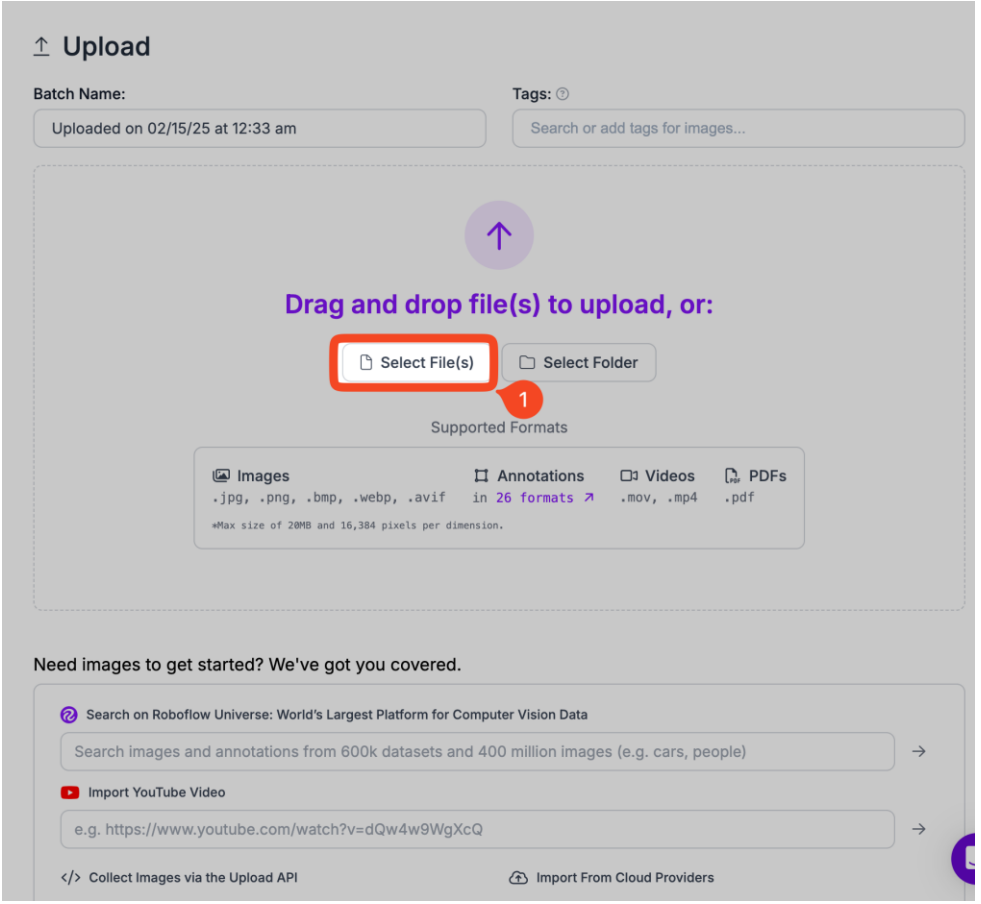
Step 2: Create project

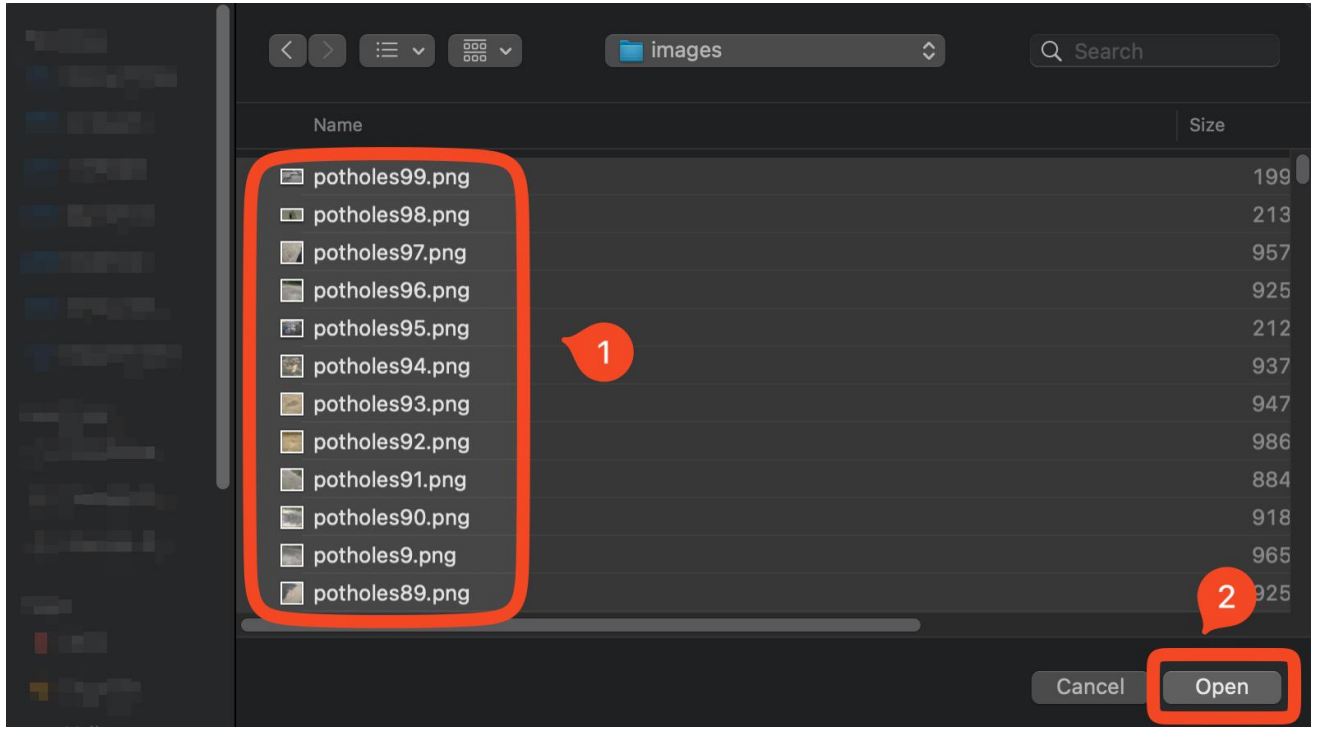


Step 2: Create project

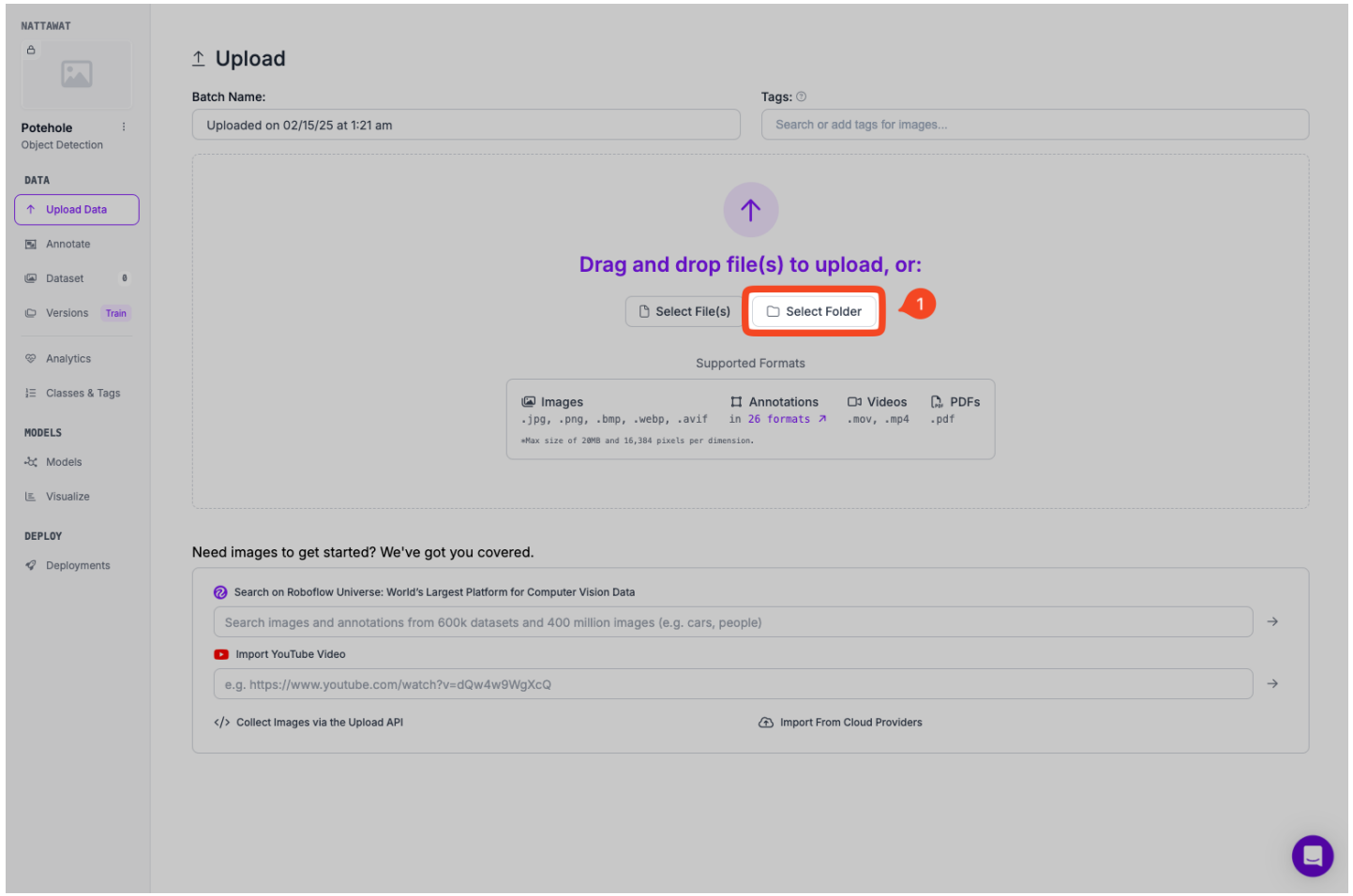


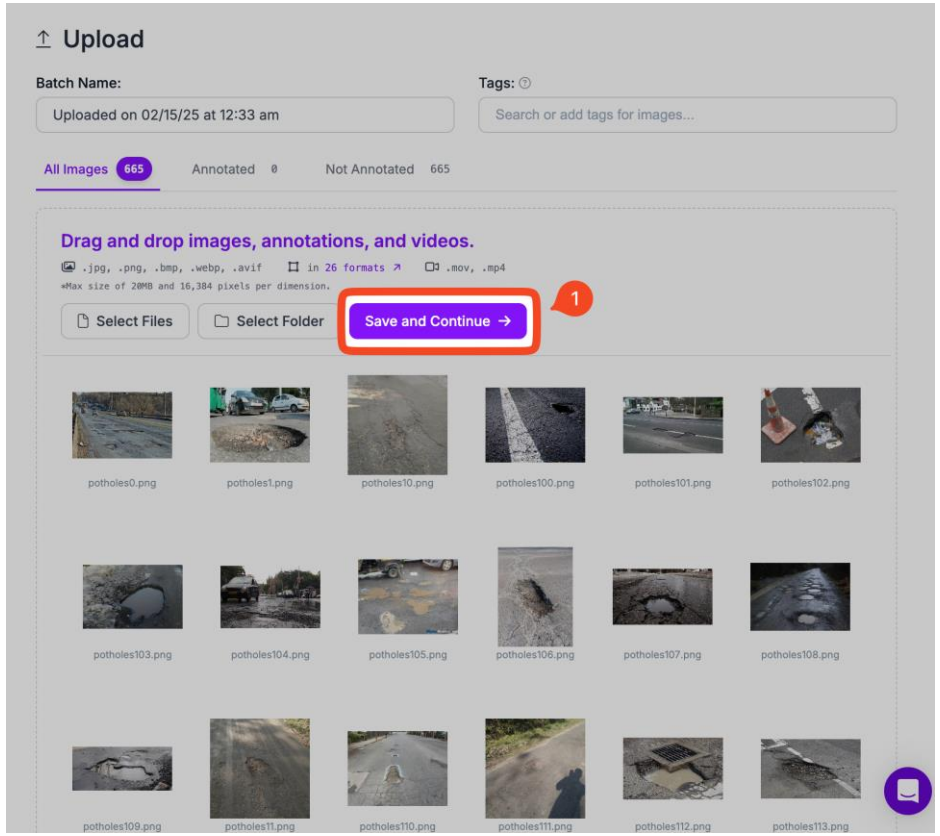
Step 2: Uploading Images



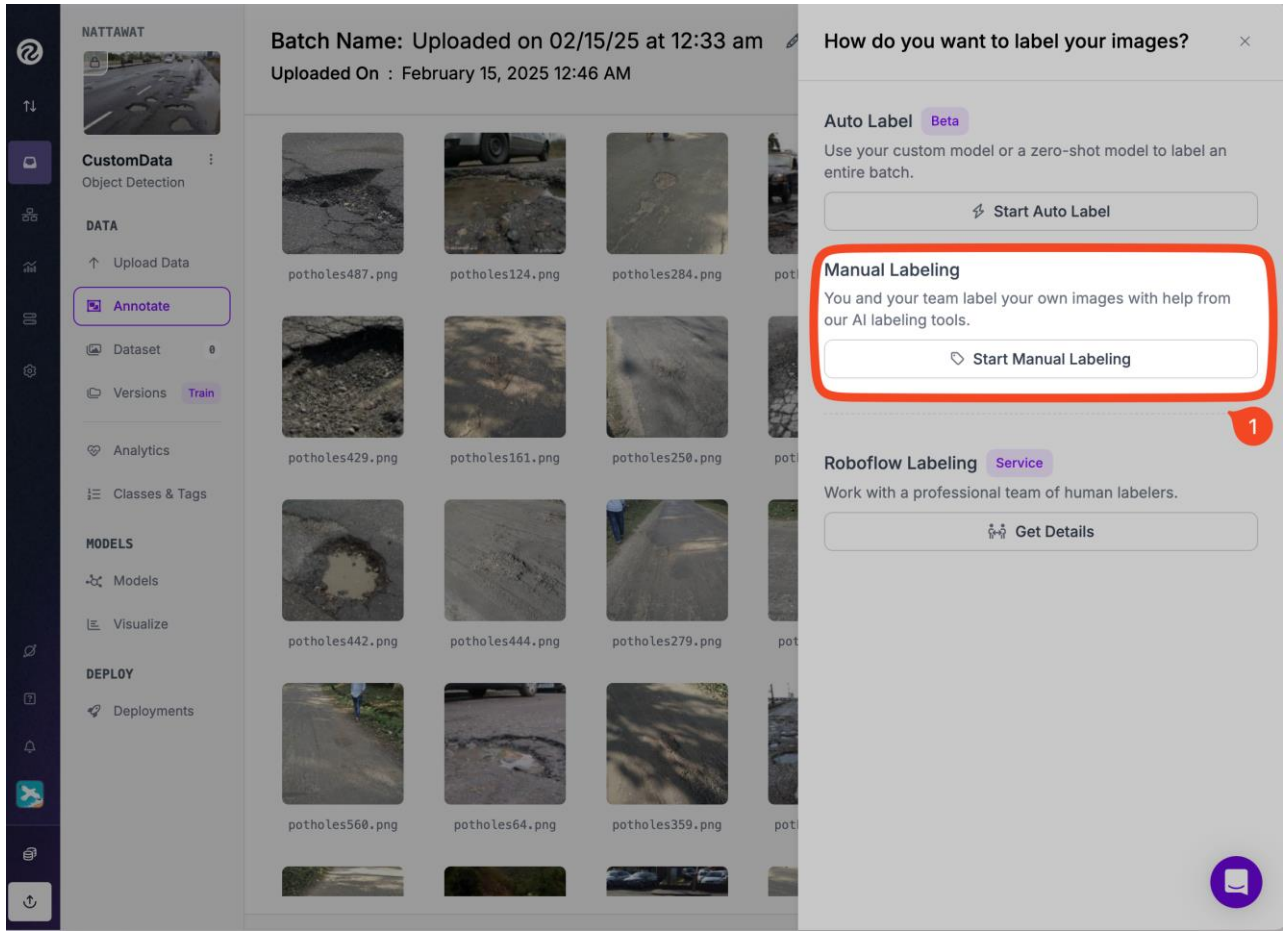


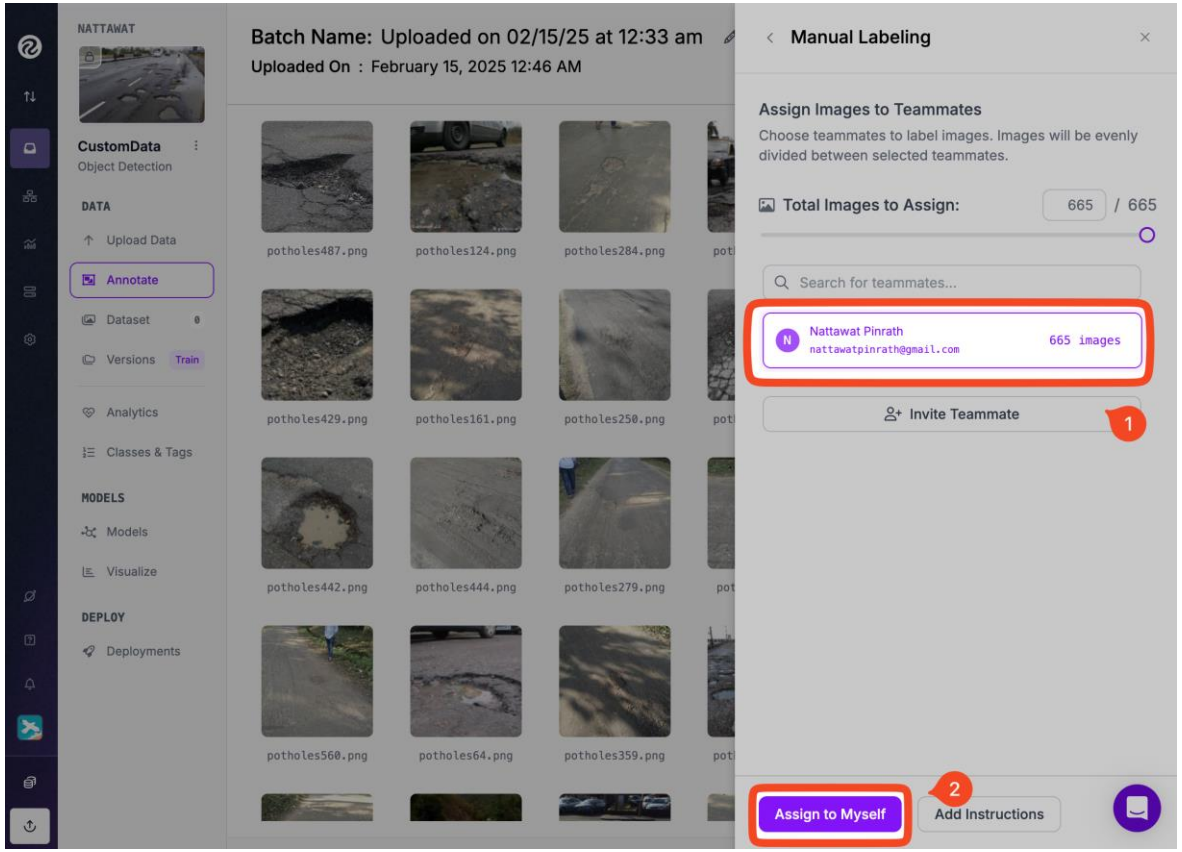
Step 2: Uploading Images



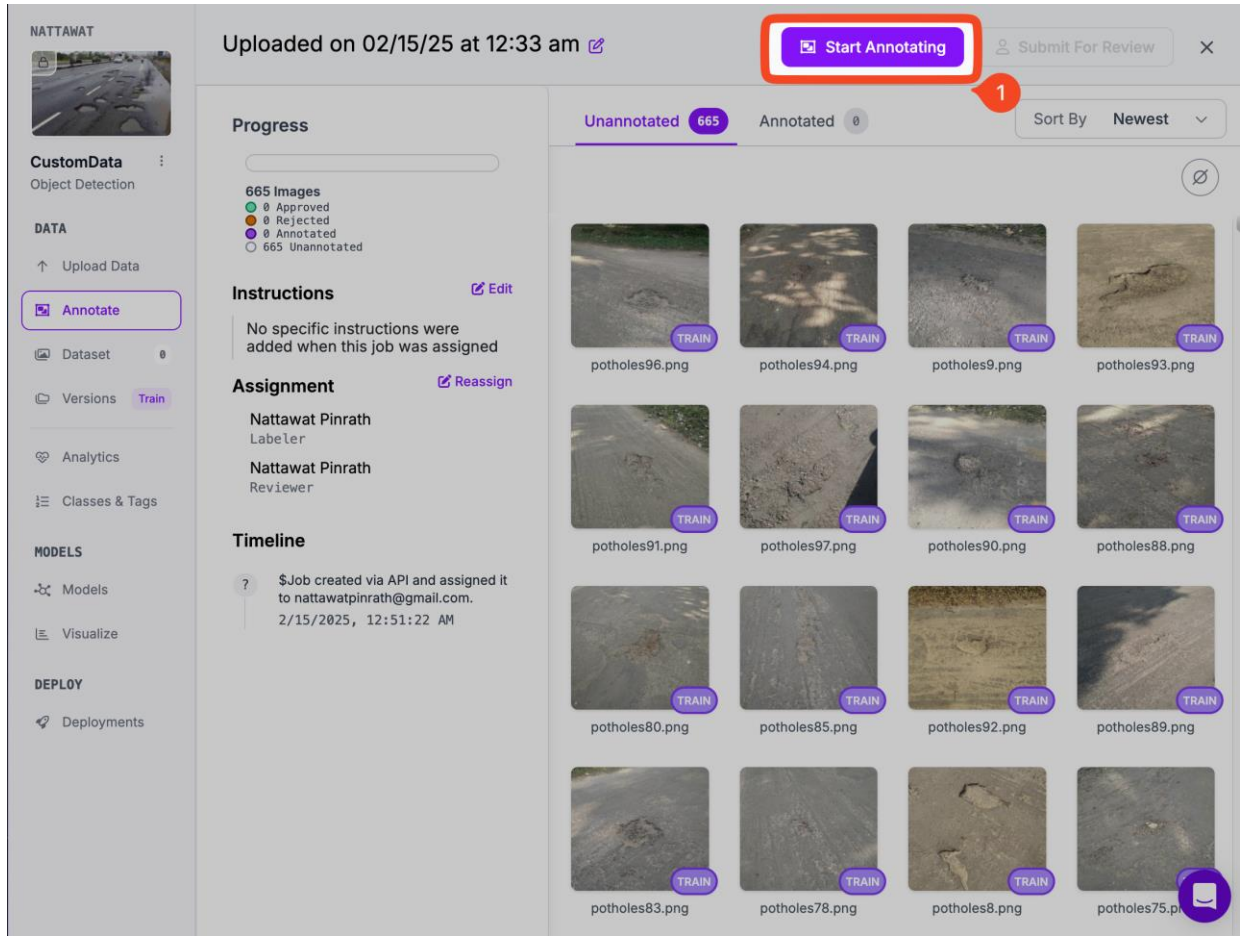


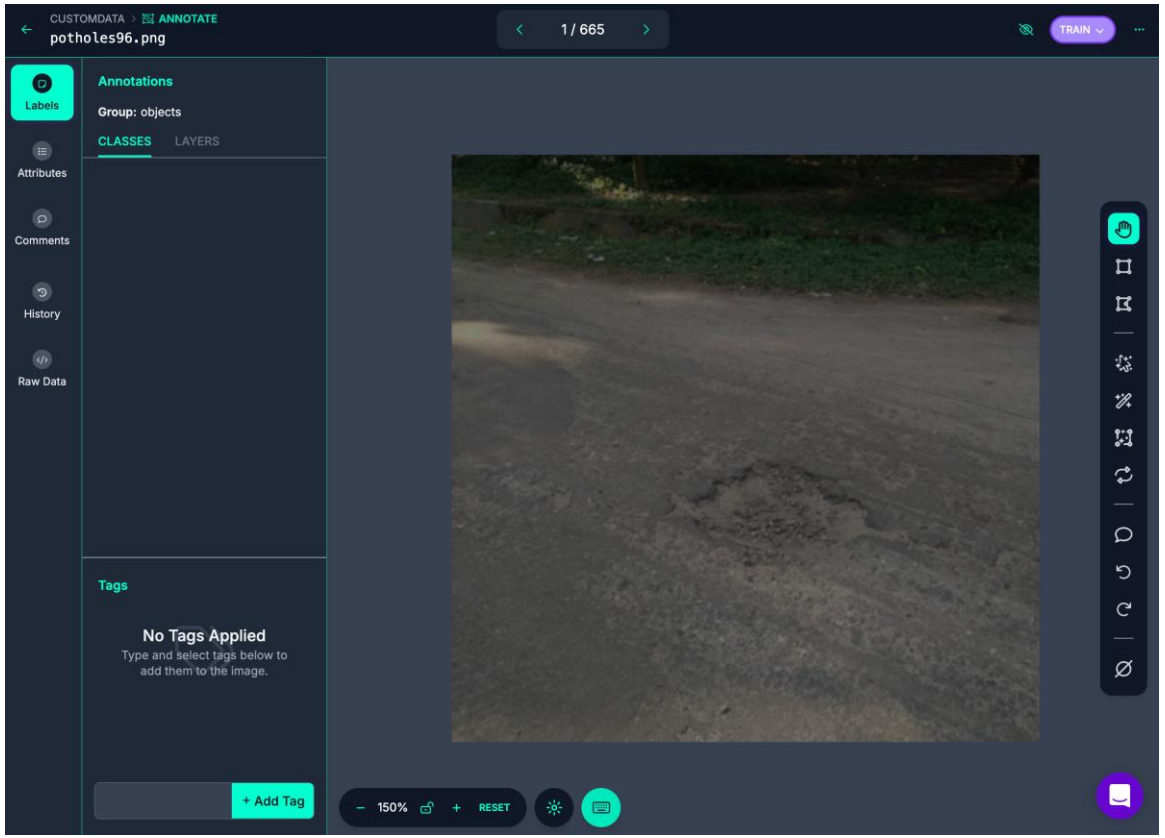
Step 2: Manual Annotation



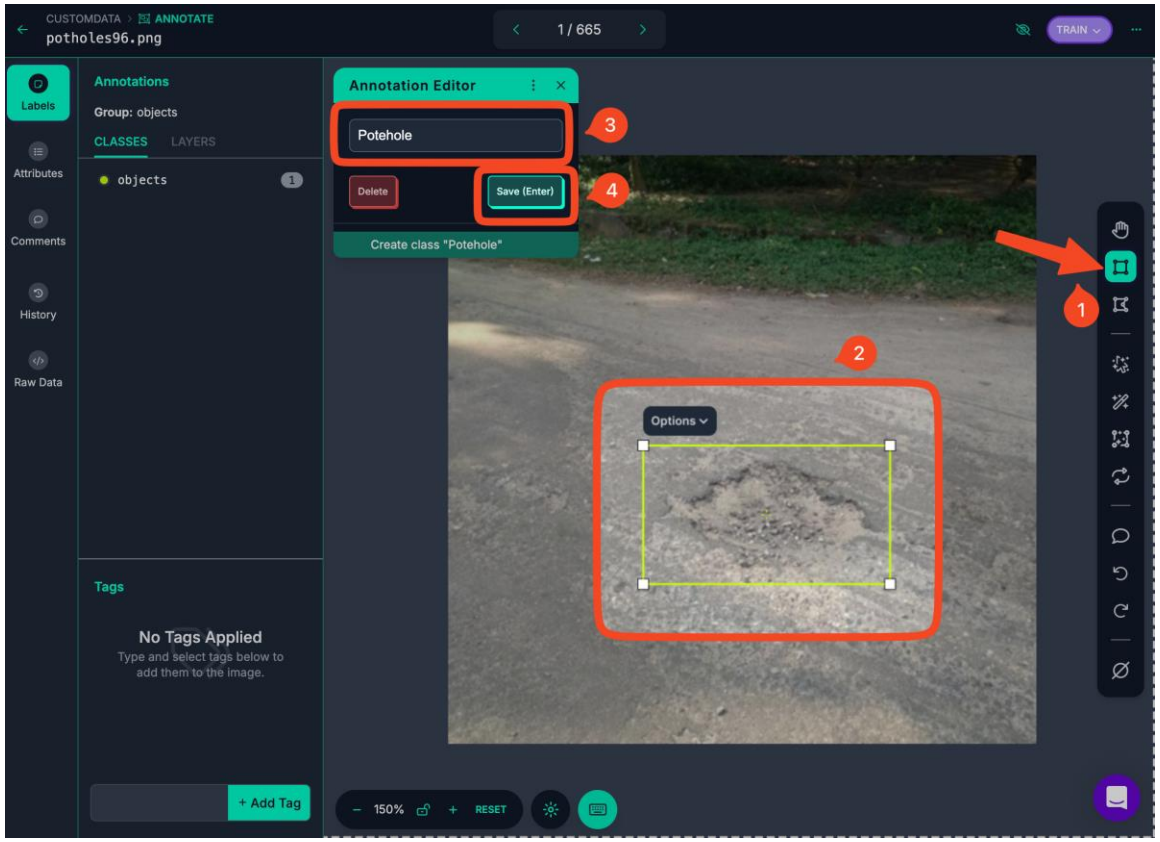


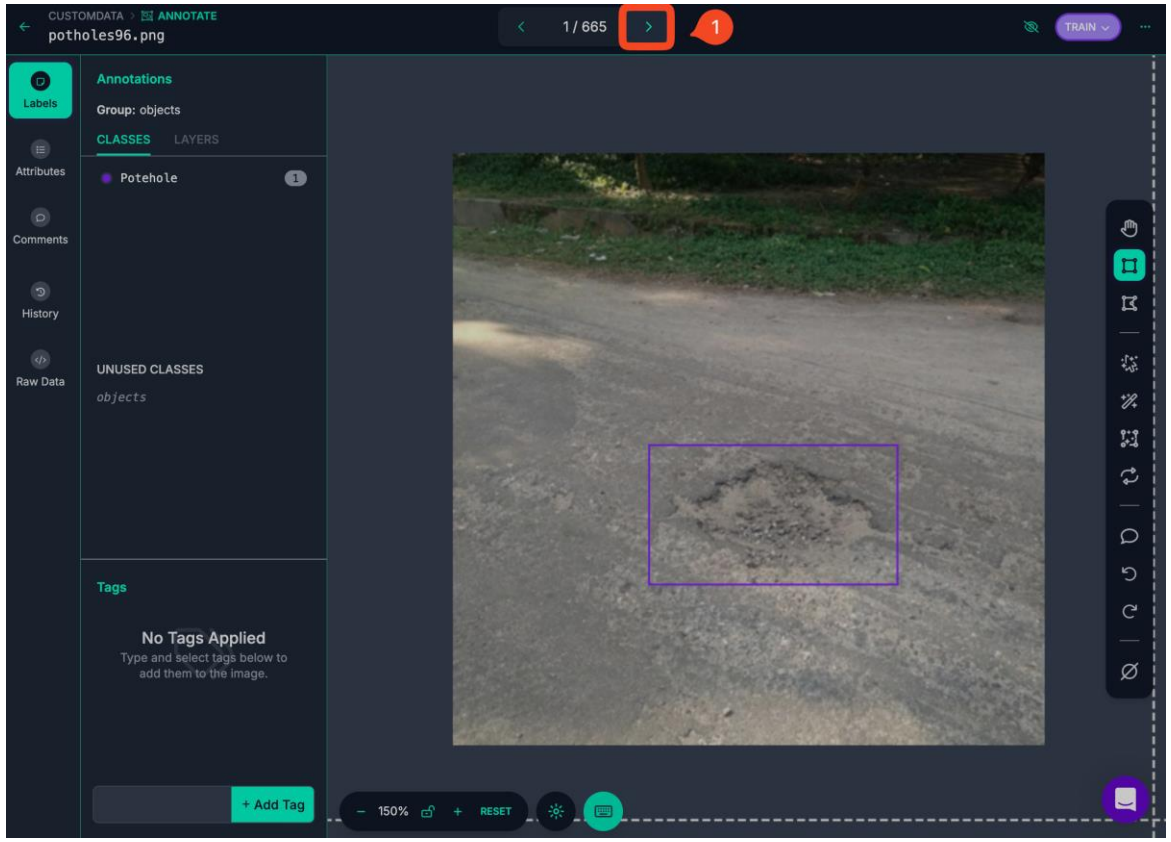
Step 2: Manual Annotation



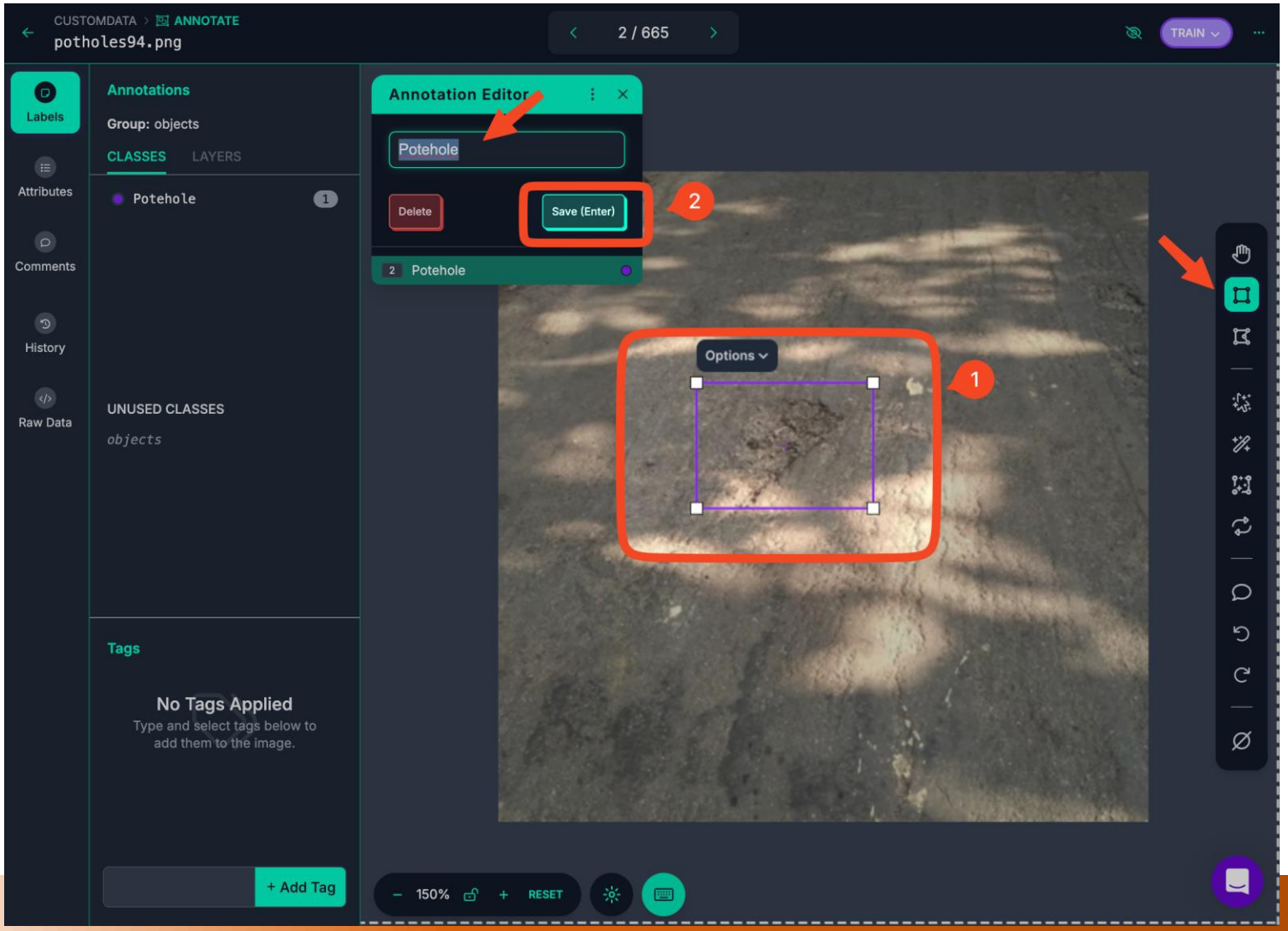


Step 2: Manual Annotation

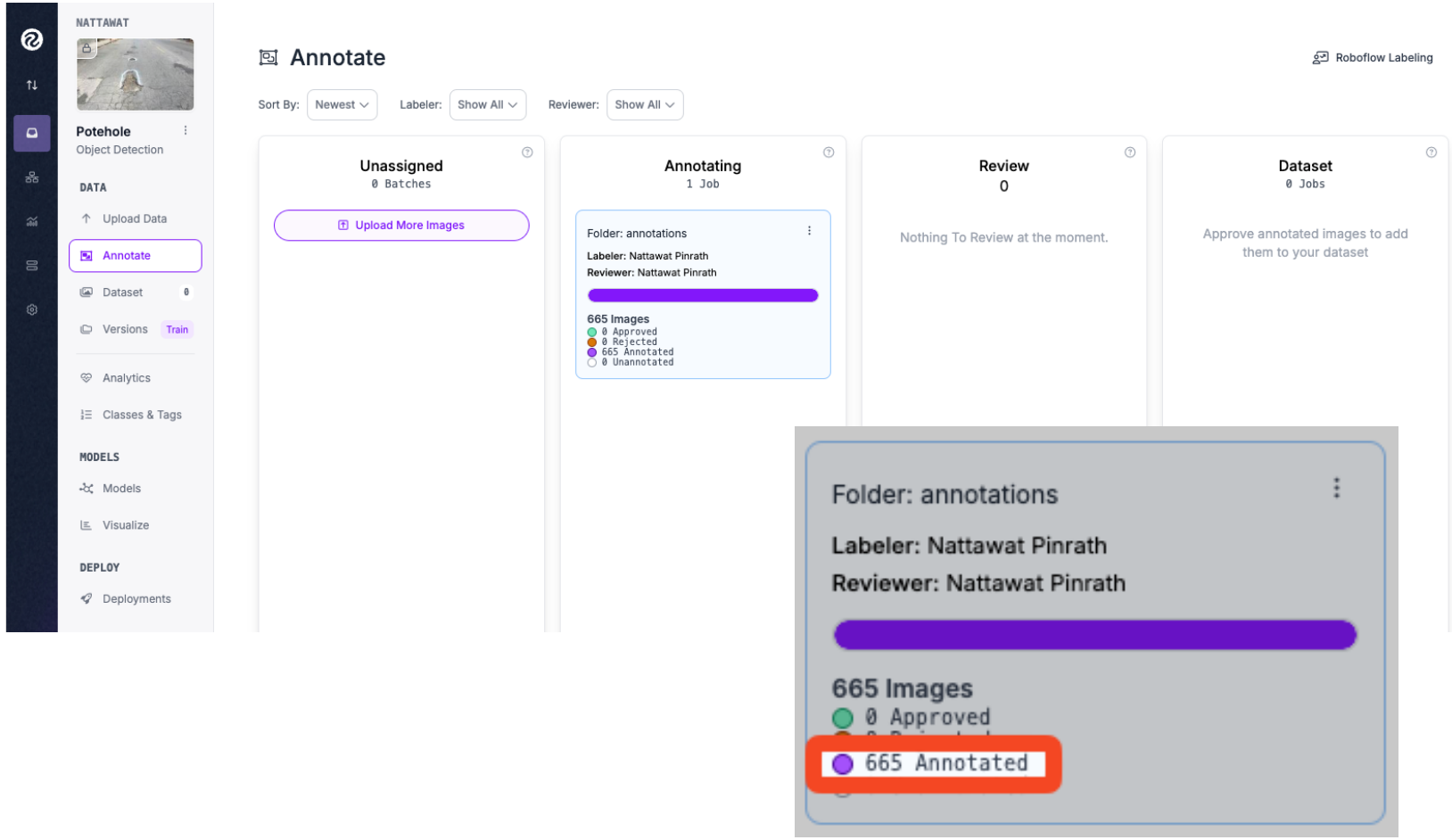




Step 2: Manual Annotation



Step 2: Manual Annotation

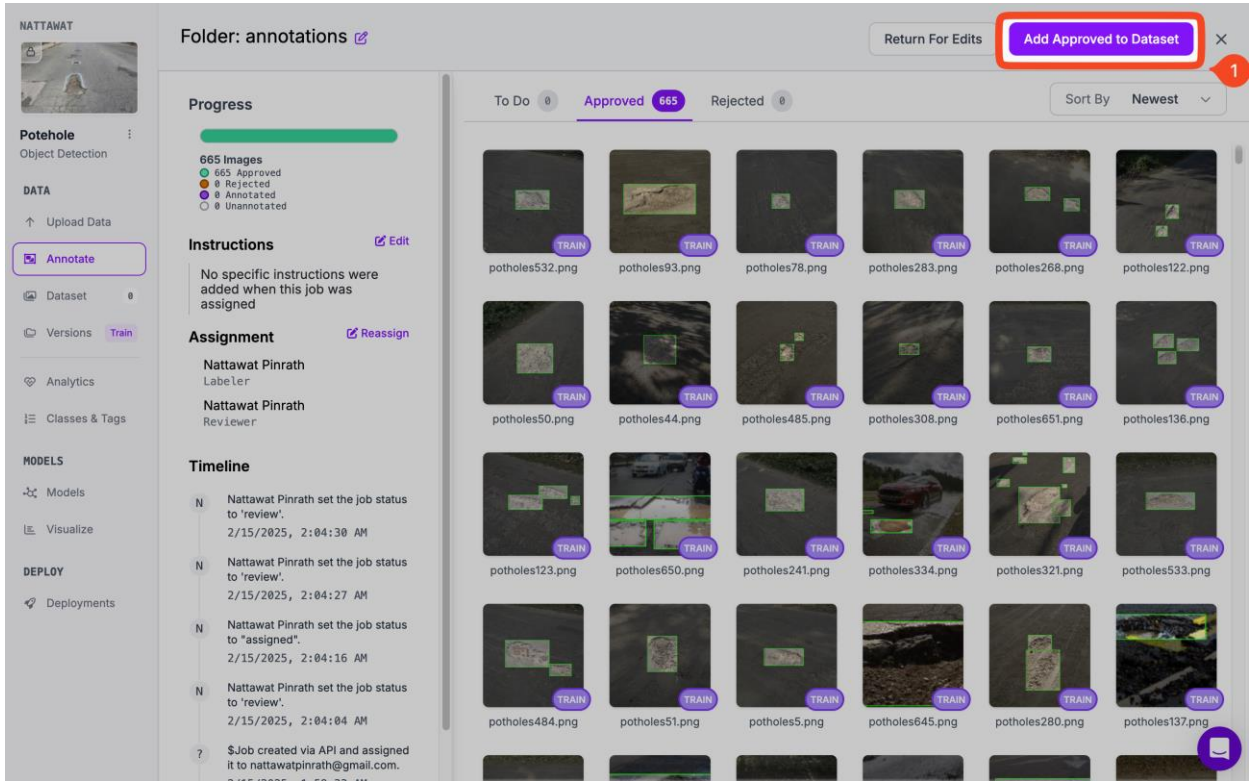


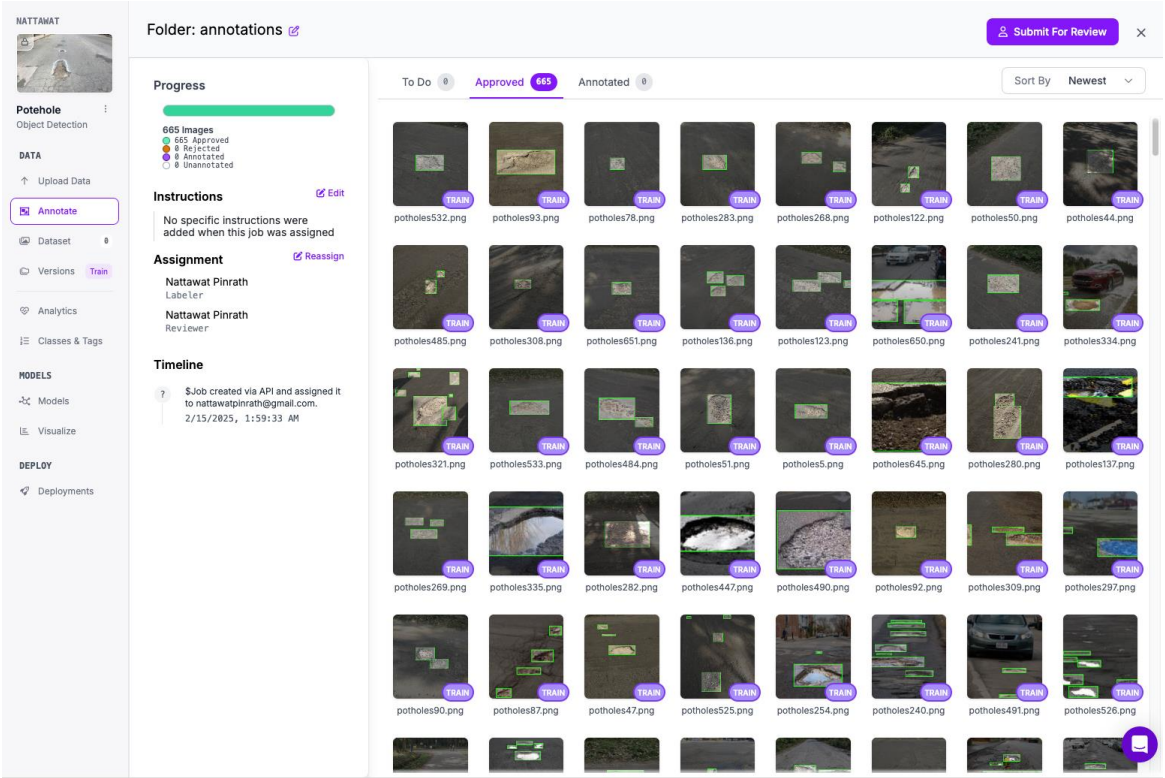
Step 3 : Annotate Verify and DataSet Splitting

A. Before exporting the dataset, ensure:

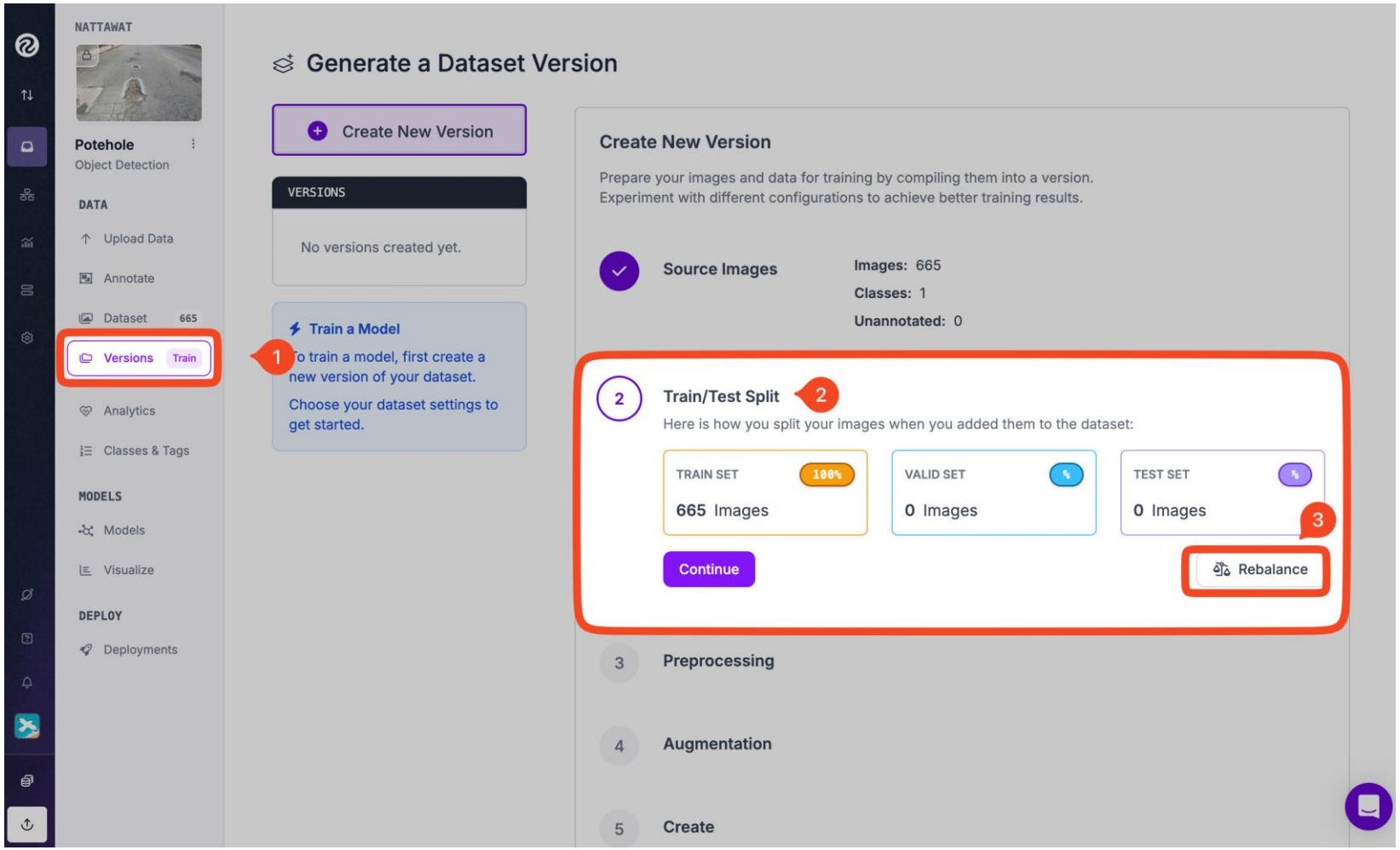
- ✓ All objects are labeled correctly (no missing objects).
- ✓ Bounding boxes are well-fitted to objects.
- ✓ Class labels are consistent across all images.

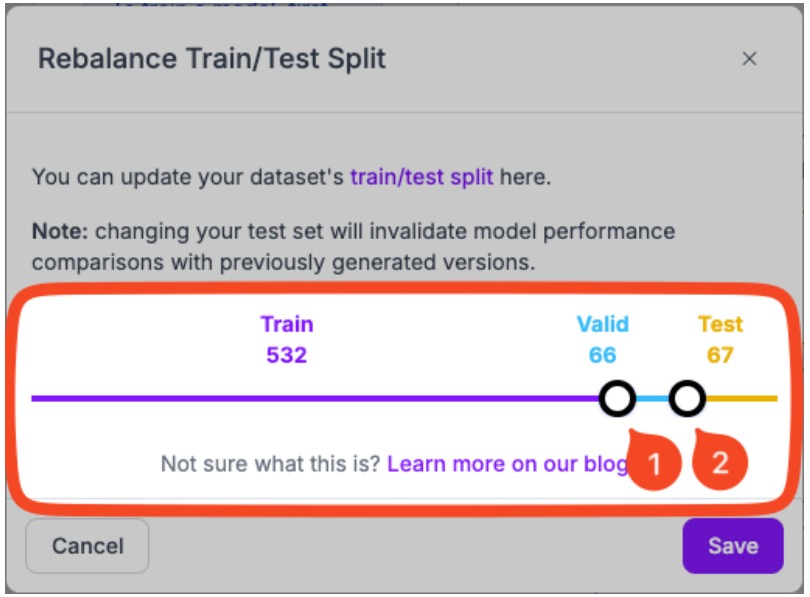
Step 3 : Annotate Verify





Step 3 : DataSet Splitting





Train-Test Splitting

The performance of YOLOv11 in object detection depends on the careful partitioning of the dataset into training, validation, and test sets. A well-balanced train-test split is essential for enhancing model generalization, minimizing overfitting, and ensuring the model performs reliably in real-world applications.

Recommended Dataset Splitting Strategy

To maximize learning efficiency and performance, consider the following dataset allocation:

- Training Set (70–80%) – This set provides the model with sufficient examples to learn feature representations and optimize weight adjustments.
- Validation Set (10–15%) – Used to fine-tune hyperparameters and monitor model performance, helping to prevent overfitting.
- Test Set (10–15%) – An independent dataset reserved for final evaluation to ensure generalization to unseen data.

Key Factors in Data Partitioning

1. **Dataset Size and Structure:**
    1. For large datasets, an **80%-10%-10%** split ensures ample training data while maintaining evaluation rigor.
    2. Smaller datasets benefit from a **70%-15%-15%** split to provide a more balanced assessment.
2. **Class Distribution and Stratification:**
    1. Each subset should reflect the same class distribution as the entire dataset to avoid biased learning.
    2. Stratified sampling techniques should be applied to maintain class balance across training, validation, and test sets.
3. **Cross-Validation for Improved Robustness:**
    1. When working with limited data, **k-fold cross-validation** is a recommended technique to enhance model robustness by iterating across multiple training and validation sets.

Key Factors in Data Partitioning

HAL open science

Performance Benchmarking of YOLOv11 Variants for Real-Time Delivery Vehicle Detection: A Study on Accuracy, Speed, and Computational Trade-offs

Rabinandan Kishor

▶ **To cite this version:**

Rabinandan Kishor. Performance Benchmarking of YOLOv11 Variants for Real-Time Delivery Vehicle Detection: A Study on Accuracy, Speed, and Computational Trade-offs. Asian Journal of Research in Computer Science, 2024, 17 (12), pp.108 - 122. 10.9734/ajrcos/2024/v17i12532 . hal-04838801

HAL Id: hal-04838801

https://hal.science/hal-04838801v1

Submitted on 16 Dec 2024

- Larger training sets generally lead to better performance for YOLOv11.
- More training data allows the model to learn complex patterns and generalize better to unseen data.
- Splits with more training data (e.g., 80–20 or 90–10) tend to yield higher accuracy on the test set. However, extremely imbalanced splits (e.g., 99–1) may overestimate real-world performance.
- Including a validation set is crucial for YOLOv11 to prevent overfitting

Overfitting

Training Set Size

- Larger training sets generally reduce the risk of overfitting in YOLOv11.
- More diverse training data helps the model learn generalizable features rather than memorizing specific examples.

Test Set Reliability

- A smaller test set may not accurately represent the true distribution of data, potentially masking overfitting.
- An overly small test set can lead to high variance in performance metrics, making it difficult to detect overfitting reliably.

Step 4: Augment the Dataset (Optional)

Augmentation is crucial for **increasing dataset diversity** and improving YOLO model performance. Roboflow provides multiple augmentation techniques.

Step 4: Augment the Dataset (Optional)

Step 1: Applying Augmentation

1. Go to **"Generate Dataset"**.
2. Click **"Augment"** and select augmentation methods.

Step 2: Available Augmentation Methods

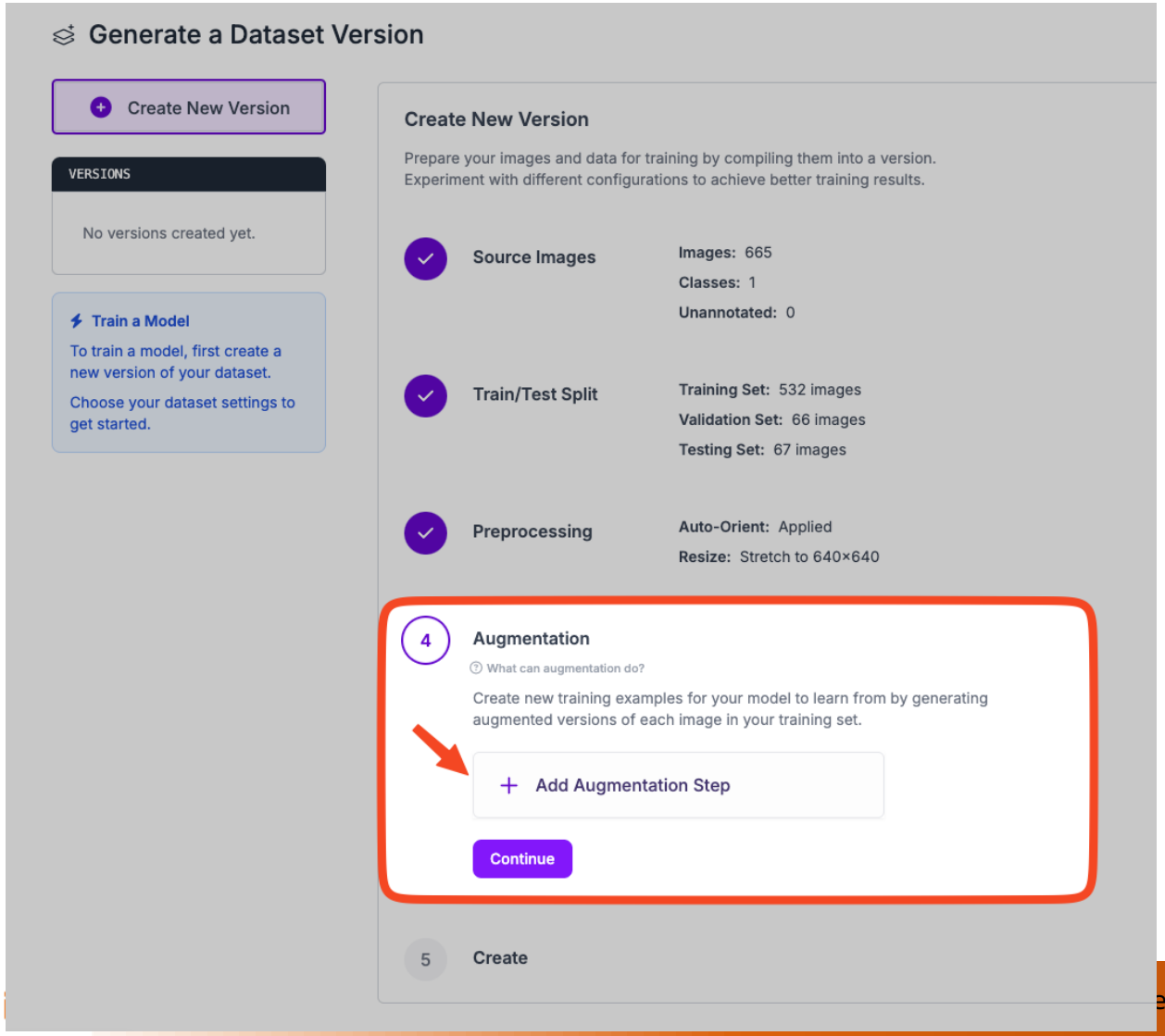
- **Rotation** (e.g., ±10°)
- **Flipping** (horizontal, vertical)
- **Brightness & Contrast Adjustment**
- **Noise Addition** (Gaussian, Salt & Pepper)
- **Blur (Gaussian Blur, Motion Blur)**
- **Cutout (Simulating Occlusions)**
- **Color Jitter (Hue, Saturation, Exposure)**

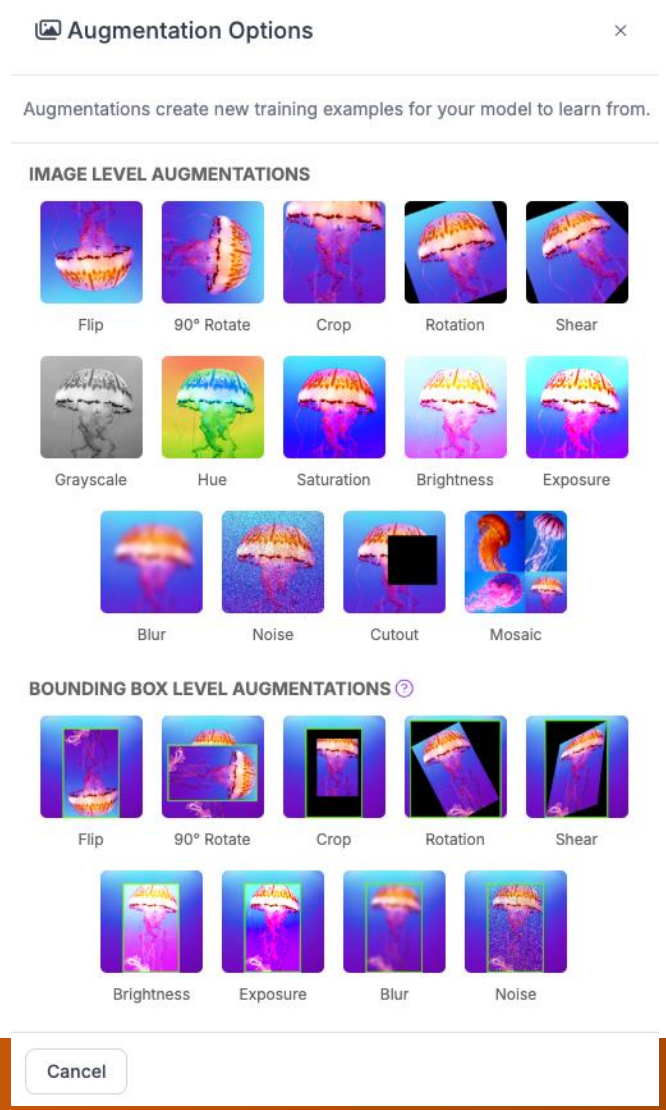
• Step 3: Configuring Augmentation

- Choose how many **augmented copies** you want per image. **Example:** If you have 500 original images and create **2 augmentations per image**, the dataset will have **1,500 images**.
- Click **"Apply Augmentation"** to process.

• Step 4: Dataset Versioning

Step 4 : Augment the Dataset





Key Augmentation Techniques

- Geometric Transformations:
  - Applying random crops, rotations, and scaling alters the spatial properties of images, helping the model recognize objects in varied orientations and sizes.
- Photometric Adjustments:
  - Modifying image brightness, contrast, and adding noise or blur enhances the model's resilience to different lighting conditions and image qualities. Studies have shown that incorporating blur and noise can improve detection metrics, making the model more robust in unclear conditions
- Advanced Augmentation Strategies:
  - Techniques like Mosaic augmentation, which combines multiple images into one, expose the model to complex backgrounds and object overlaps, thereby improving its contextual understanding.

Function of Augmentation

Image Transformations

- Flipping: Both horizontal and vertical flipping are applied to create mirror images, helping the model detect objects regardless of their orientation.
- Rotation: Images are rotated at various angles, enabling the model to recognize objects from different perspectives.
- Scaling: Input images are resized to different dimensions, allowing the model to adapt to objects of various sizes.
- Translation: Images are shifted horizontally or vertically, simulating different object placements within the frame.
- Shearing: This technique distorts images along an axis, further diversifying the training data.

Function of Augmentation

Color Adjustments

- HSV Manipulation: Adjusting hue, saturation, and brightness helps the model perform well under different lighting conditions.
- Grayscale Conversion: Some images are converted to grayscale, improving the model's ability to detect objects based on shape and texture rather than color.

Advanced Techniques

- Mosaic Augmentation: This method combines multiple images into a single training sample, enhancing the model's ability to detect objects in complex scenes with overlapping elements.
- Random Cropping: Images are randomly cropped, with dimensions ranging from 0% to 49% of the original size, helping the model learn to detect partially obscured objects.
- Cutout: Random sections of the image are masked out, forcing the model to rely on partial information for detection.

Impact of Augmentation on Model Performance in yolov11

Data augmentation techniques can significantly enhance the performance of object detection models, particularly for certain types of datasets.

While YOLOv11 can benefit from data augmentation across various domains, some datasets tend to show more pronounced improvements:

Impact of Augmentation on Model Performance in yolov11

Small and Imbalanced Datasets

- Datasets with limited samples or class imbalances often benefit the most from data augmentation techniques when used with YOLOv11:
- Few-shot Learning (FSL) Scenarios: In cases where only a small number of samples are available for certain classes, data augmentation can effectively expand the training set1. Low-shot Learning (LSL) Contexts: Similar to FSL, LSL scenarios with very few examples per class can see substantial improvements through augmentation1.

Specialized Domain Datasets

- Certain domain-specific datasets can greatly benefit from tailored augmentation strategies:
- Example
  - Agricultural Datasets: crop datasets with underrepresented classes show significant improvements. YOLO11 maintained stronger performance across minority classes without compromising detection accuracy for other species. Underwater or Low-light Datasets: These challenging environments often require specific augmentation techniques like color adjustments to enhance model performance.

Implementation Consideration

1. Use a combination of techniques such as flipping, rotation, and color adjustments to create a diverse training set.
2. Implement automated data augmentation selection methods like AutoAugment or RandAugment to optimize the augmentation strategy.
3. Tailor the augmentation techniques to the specific characteristics of your dataset and the challenges it presents.

Impact of Augmentation on Model Performance in yolov11

Accuracy Improvements

- Data augmentation techniques have shown to significantly boost YOLOv11's detection accuracy:
  - Physics-informed augmentations resulted in a 7.3% increase in mean average precision (mAP) and a 12% improvement in accuracy under challenging conditions, such as extreme tilt angles or severe shading.

Practices for Augmentation in YOLOv11

To maximize the benefits of data augmentation in YOLOv11:

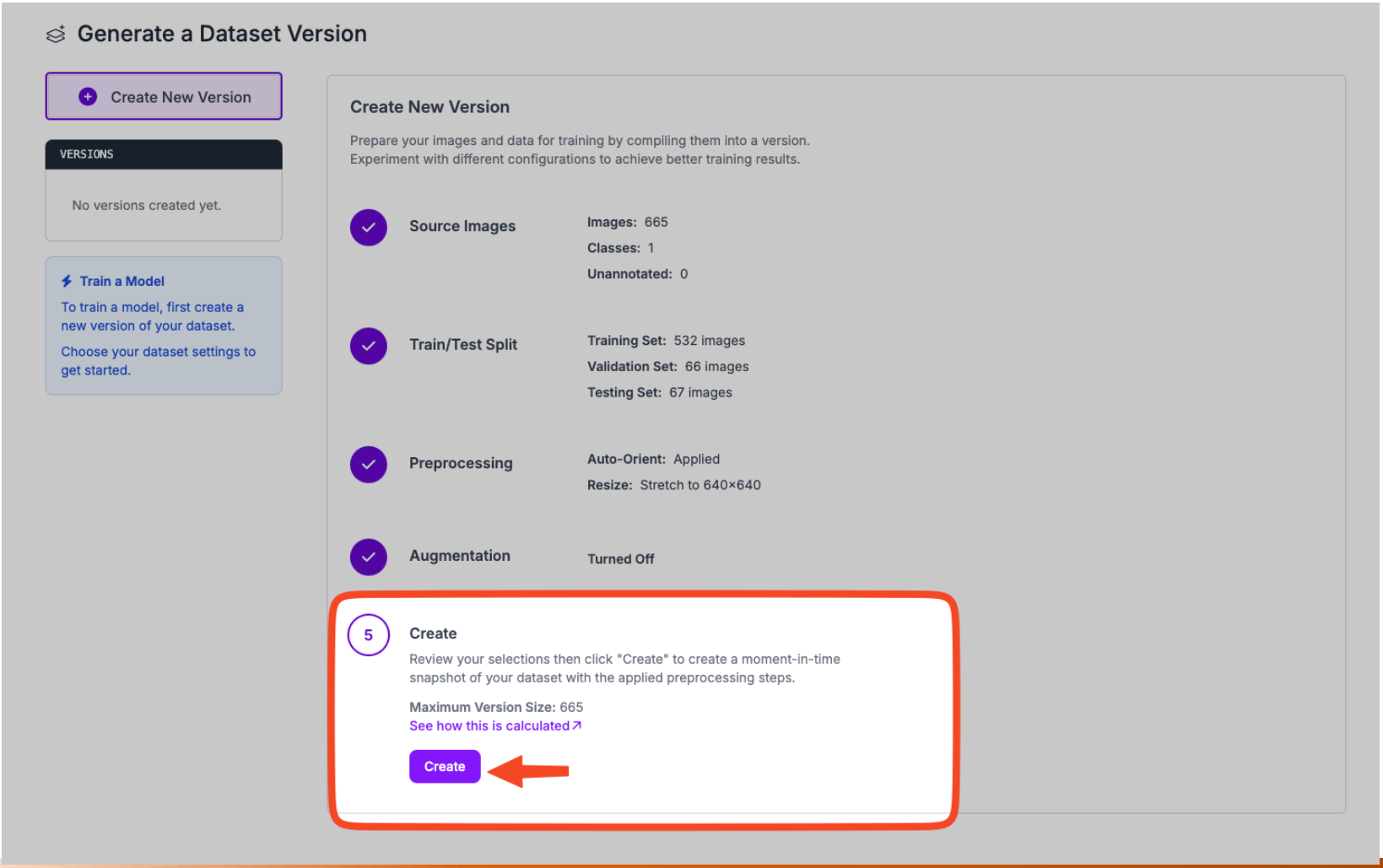
- **Balanced Application:**
  - While augmentation enriches the dataset, excessive transformations can lead to unnatural images that may confuse the model. It's crucial to find a balance that enhances diversity without compromising data quality.
- **Consistency in Augmentation:**
  - Ensure that the applied augmentations are consistent with real-world variations the model is expected to encounter. This alignment enhances the model's practical applicability.
- **Utilize Robust Libraries:**
  - Employing well-tested augmentation libraries, such as Albumentations, can streamline the augmentation process and reduce the risk of introducing errors that could degrade model performance

Step 5: Export the Dataset for YOLOv11

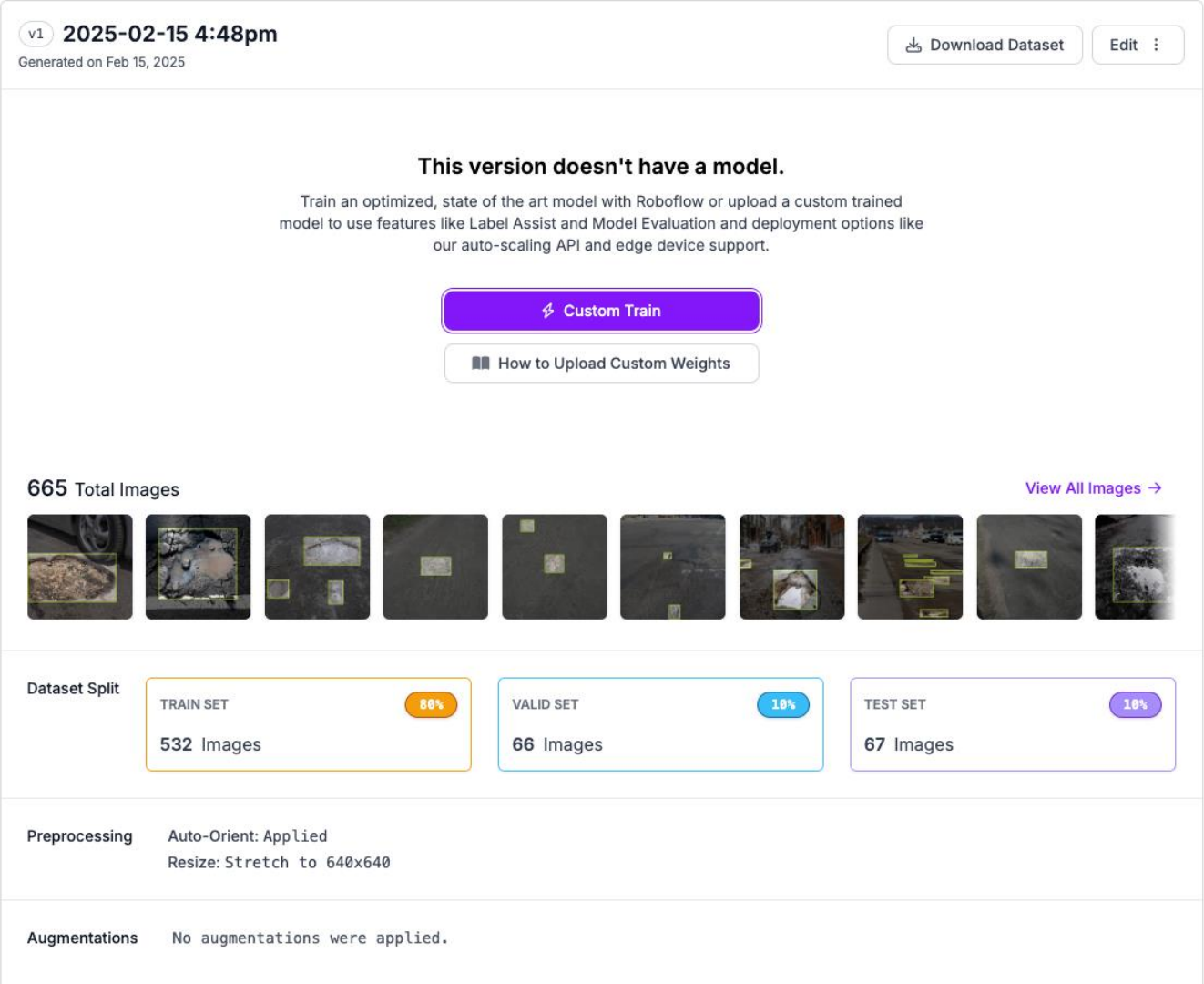
Once annotation and augmentation are complete:

1. **Click "Export Dataset"**.
2. **Choose "YOLO" Format** (Select YOLO version).
3. Download the dataset as a **ZIP file or Code**.

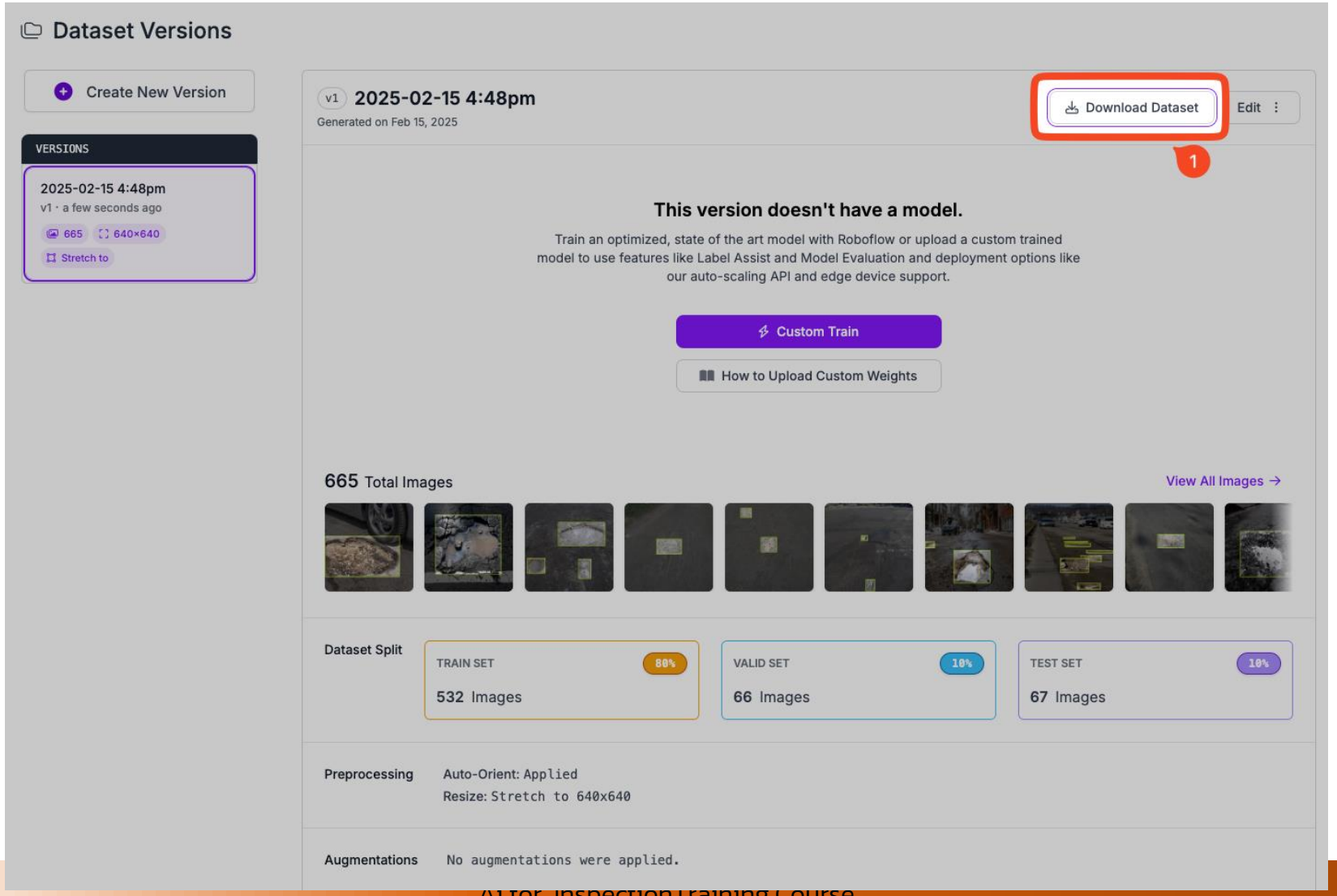
Step 5: Export the Dataset for YOLOv11



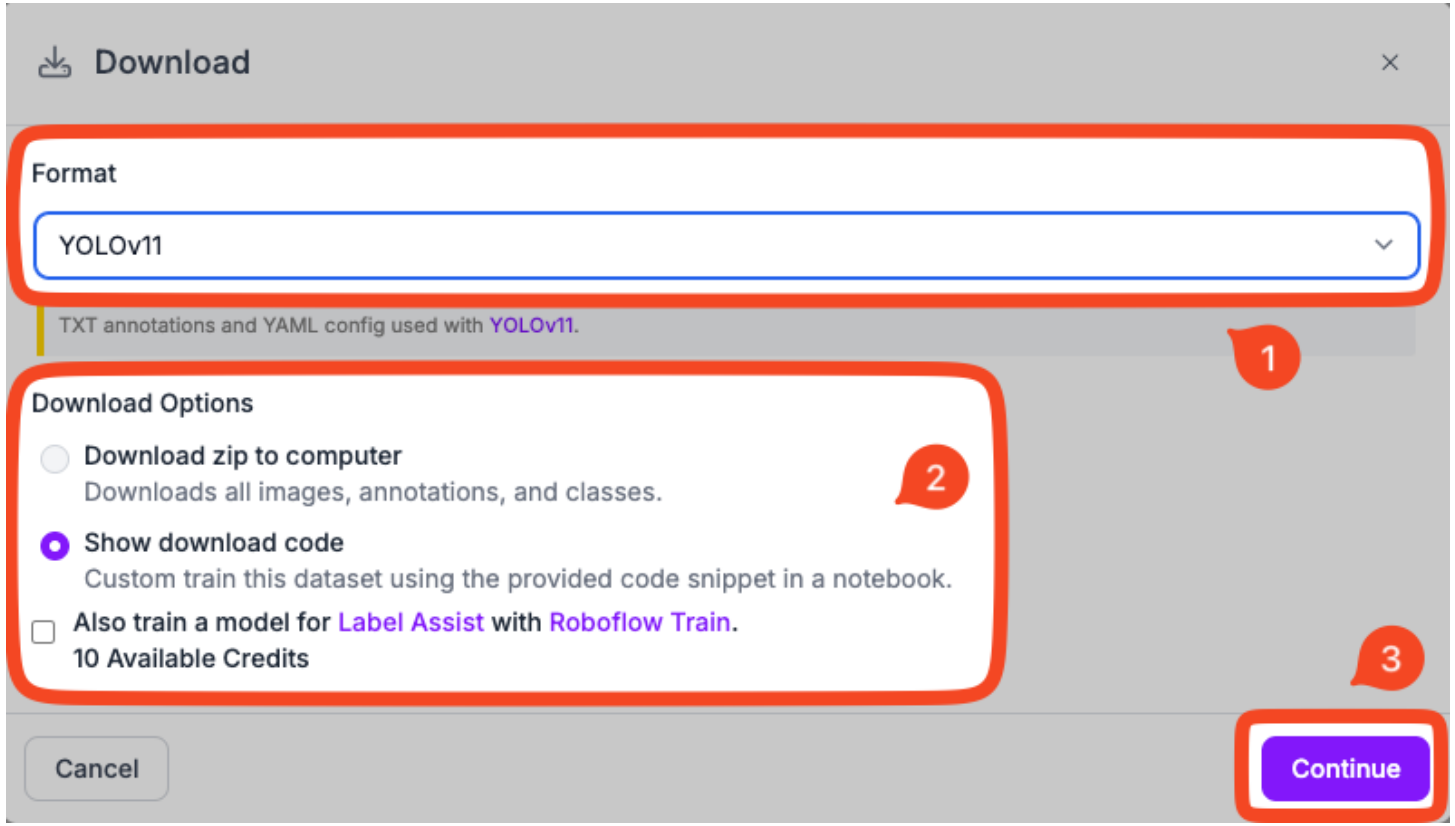
Step 5: Export the Dataset for YOLOv11



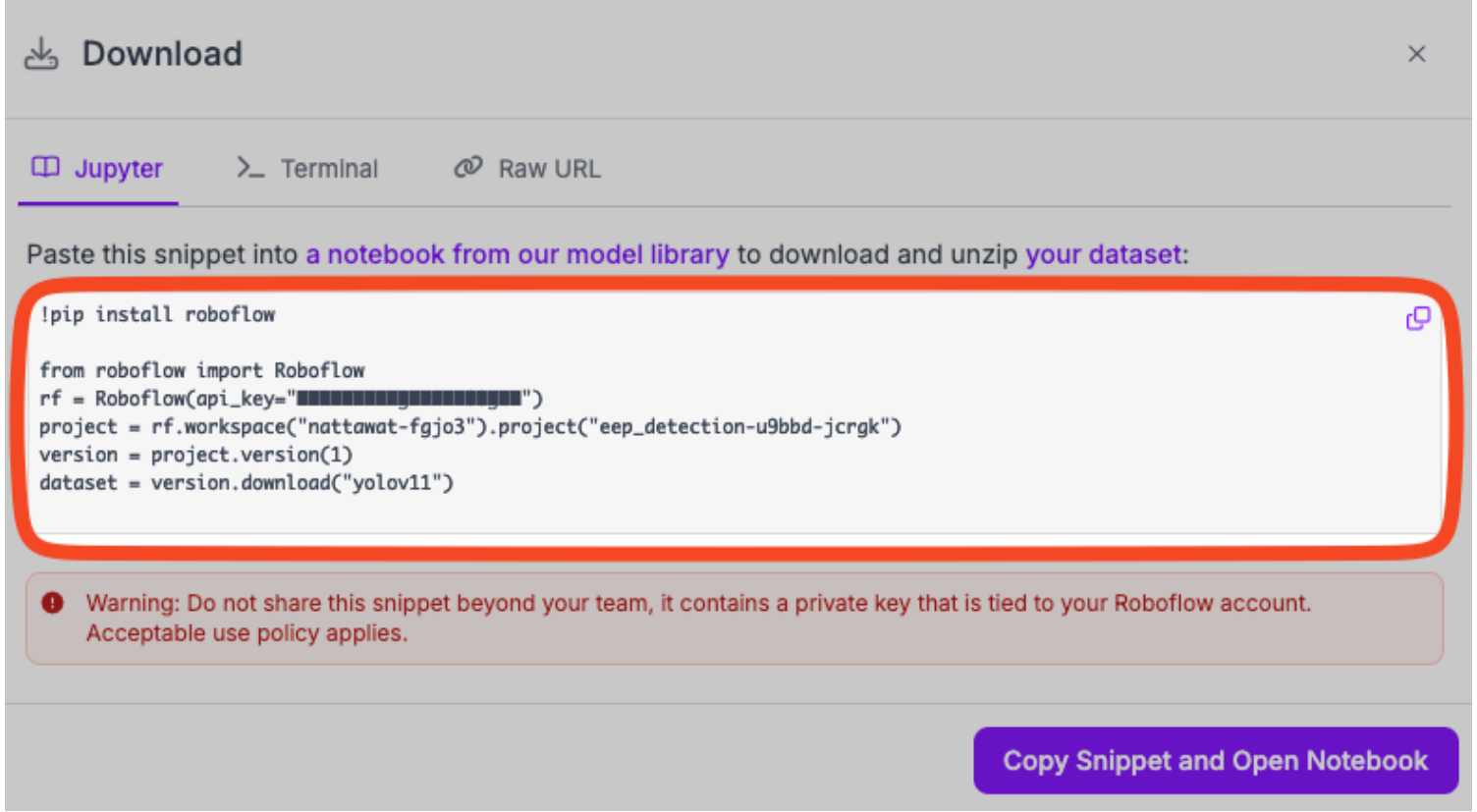
Step 5: Export the Dataset for YOLOv11



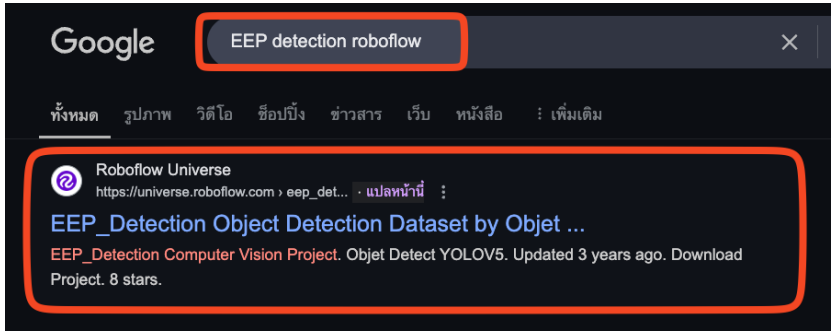
Step 5: Export the Dataset for YOLOv11

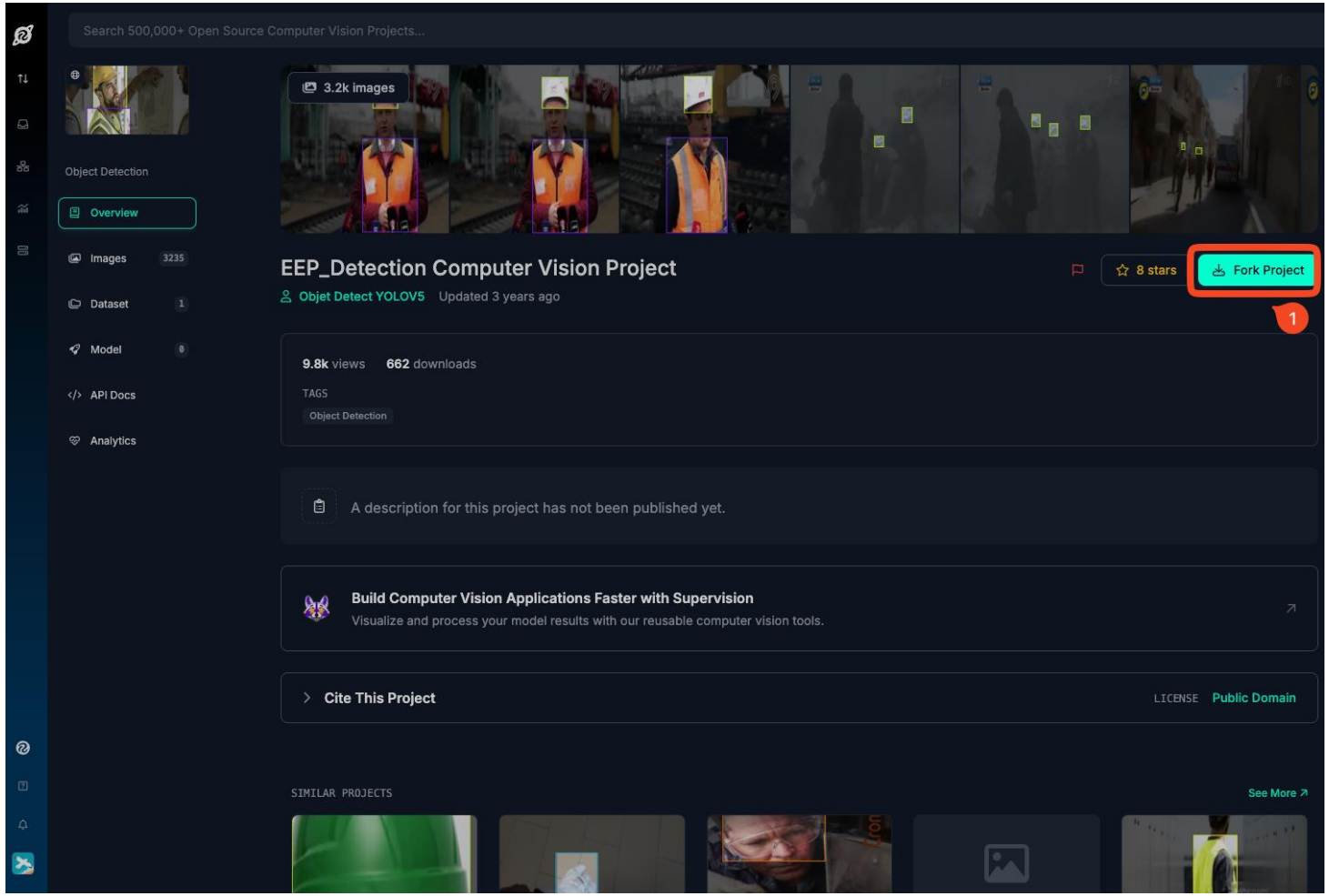


Step 5: Export the Dataset for YOLOv11

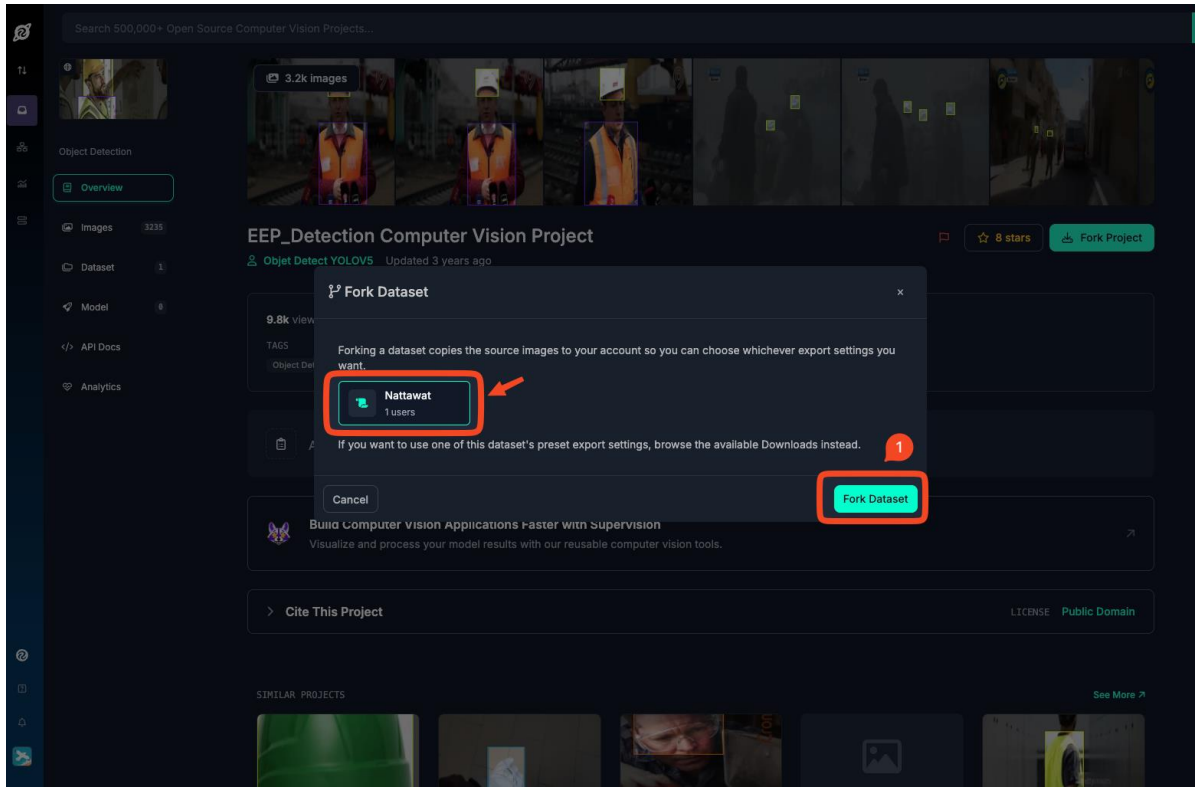


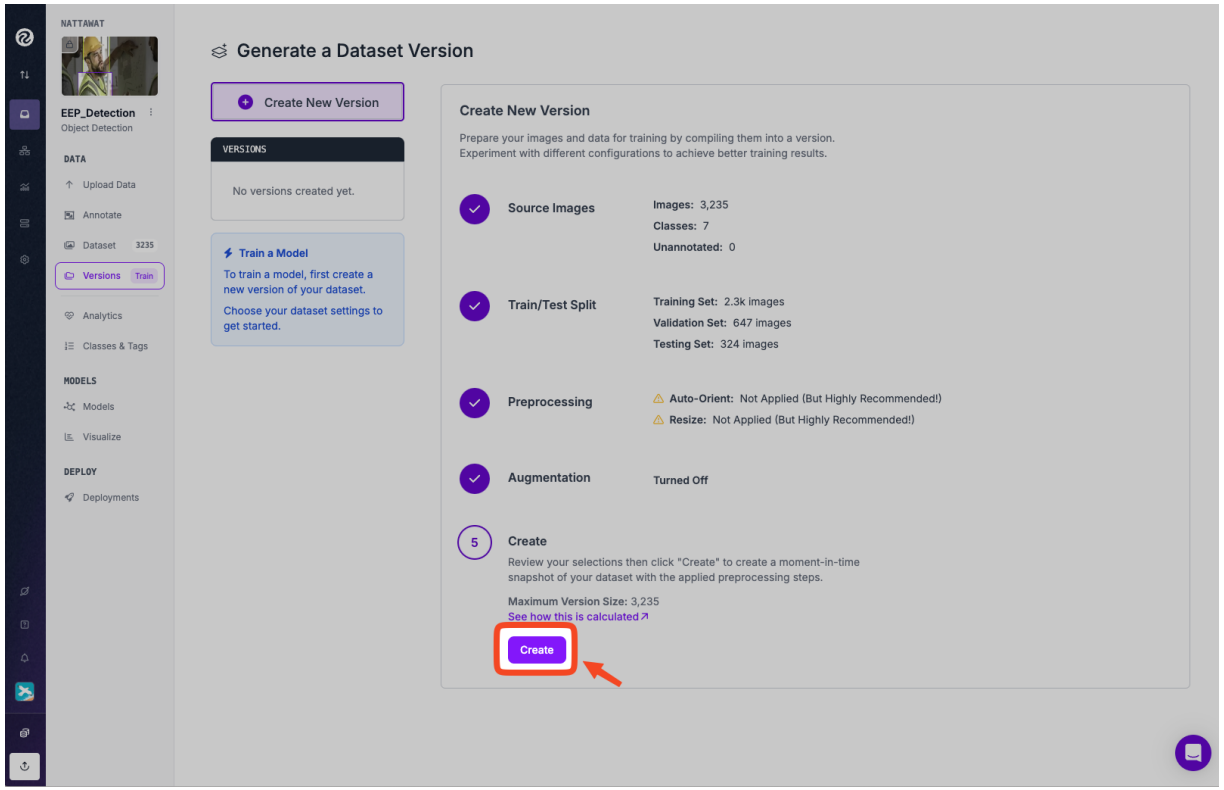
Download Existing DataSet in Roboflow



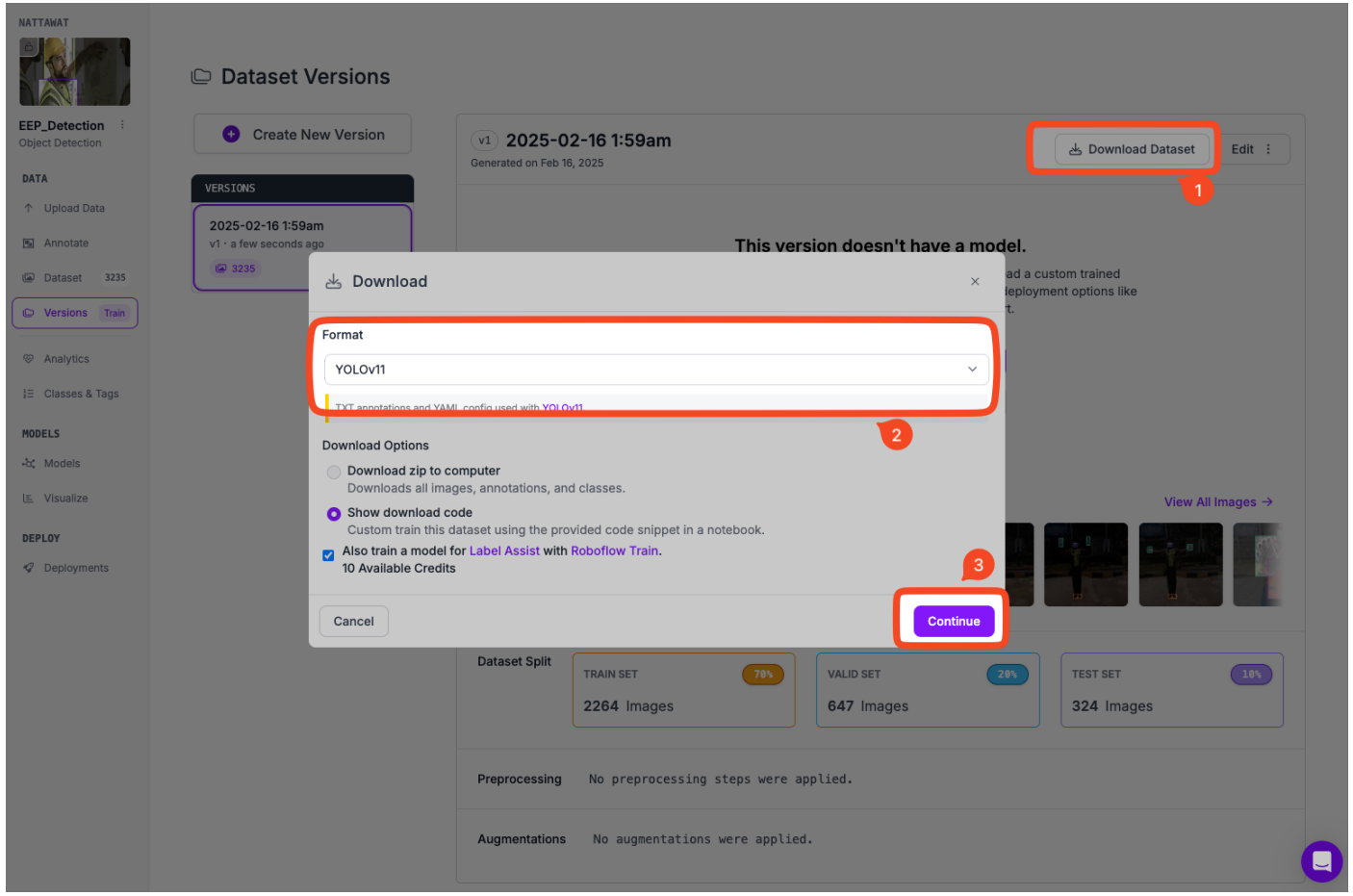


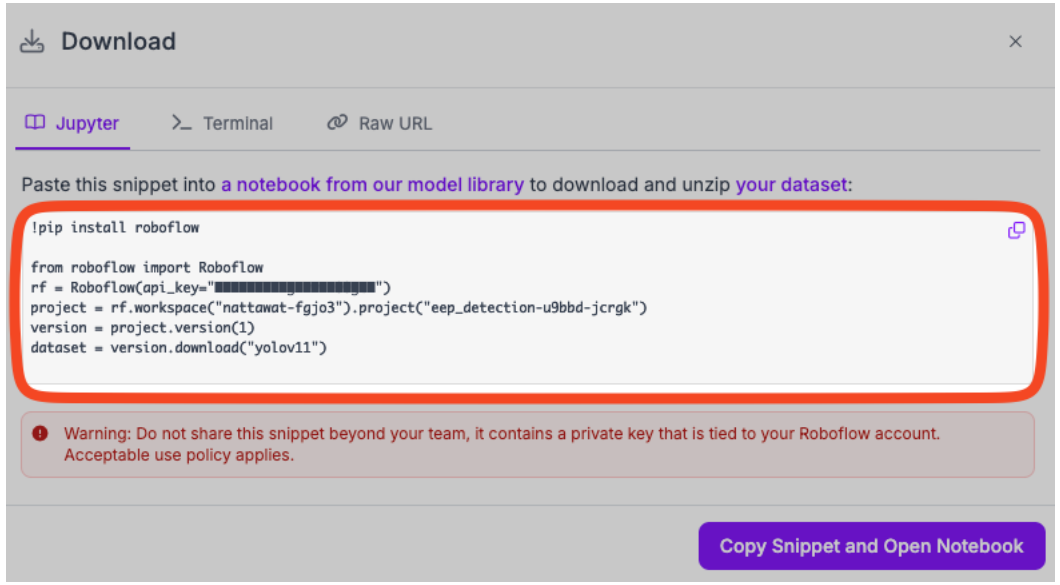
Download Existing DataSet in Roboflow





Download Existing DataSet in Roboflow





Steps to Trian YOLO in Google Colab

Step 1 : Prepare the Google Colab Environment

Step 2 : Set Up Your Dataset

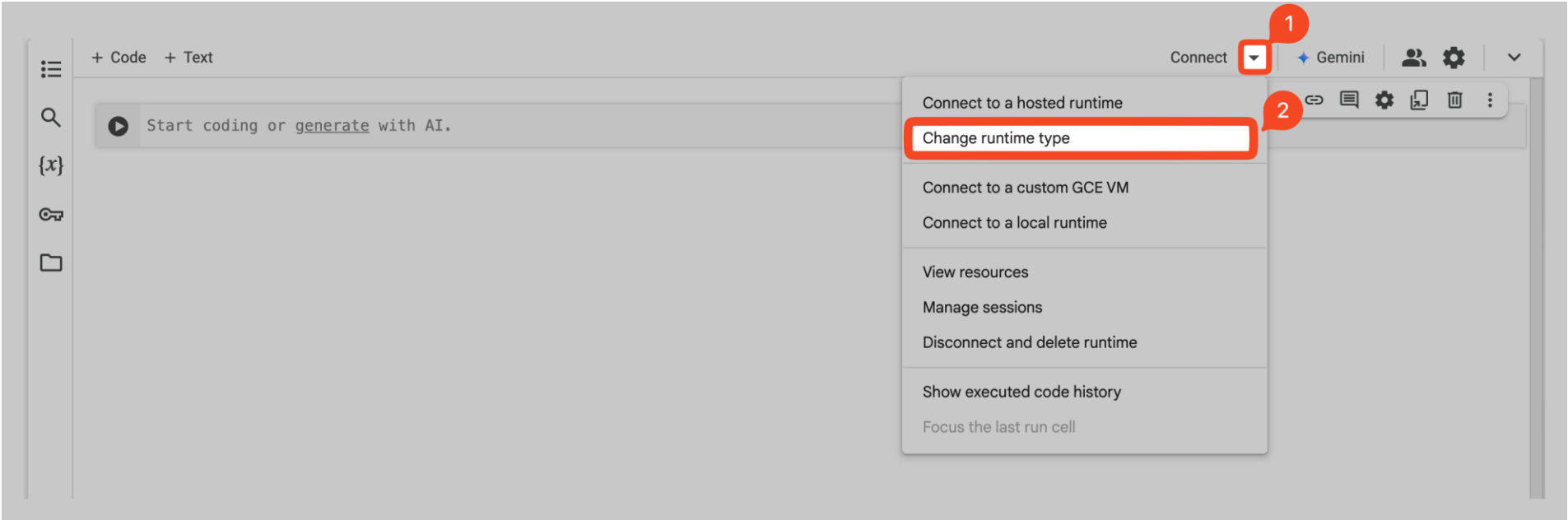
Step 3 : Train the YOLOv11 Model

Step 4 : Evaluate and Visualize the Model

Step 5 : Evaluate and Visualize the Model

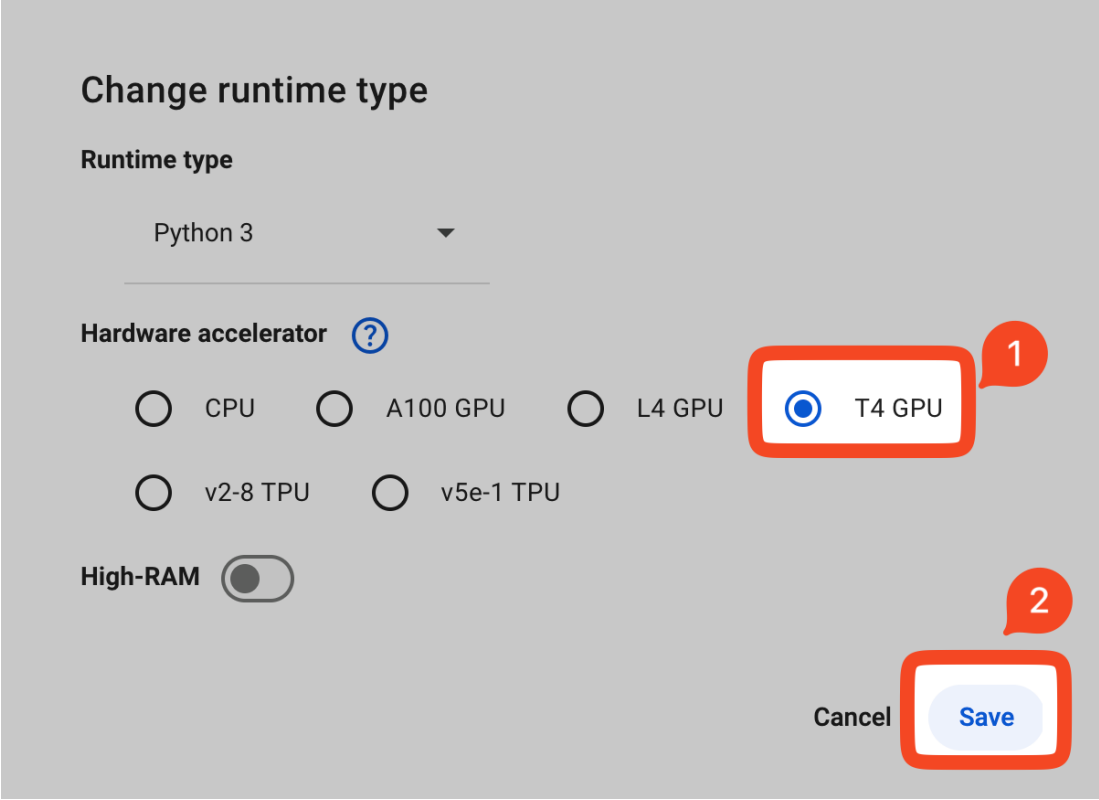
Step 1 : Prepare the Google Colab Environment

Setup Google Colab –> GPU Memory Usage



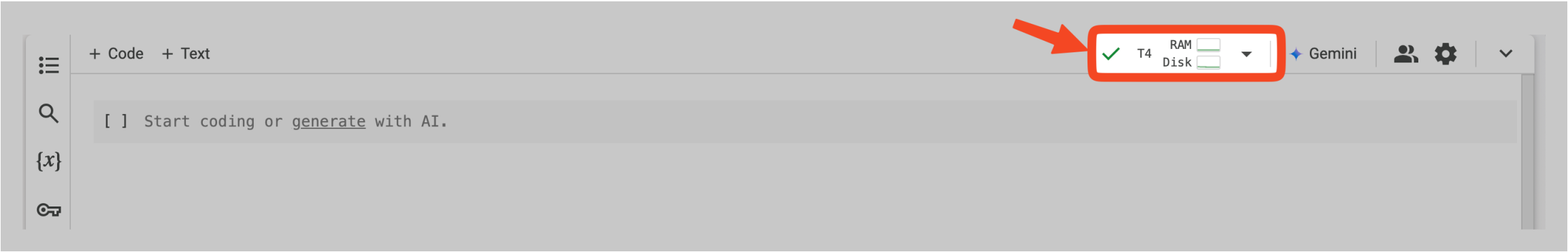
Step 1 : Prepare the Google Colab Environment

Setup Google Colab –> GPU Memory Usage



Step 1 : Prepare the Google Colab Environment

Setup Google Colab –> GPU Memory Usage



Step 1 : Prepare the Google Colab Environment

Check GPU Memory Usage

!nvidia-smi

```
!nvidia-smi

Sun Feb 16 09:24:47 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   46C    P8             10W /   70W |       0MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                                         |
|        ID   ID                                                                          |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

1

GPU Memory Usage

Step 1 : Prepare the Google Colab Environment

Install the ultralytics Package

```
!pip install ultralytics
```

Show hidden output

Import All the required Libraries

```
import ultralytics
ultralytics.checks()
```

```
Ultralytics 8.3.75 🚀 Python-3.11.11 torch-2.5.1+cu124 CUDA:0 (Tesla T4, 15095MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 33.1/235.7 GB disk)
```

```
from ultralytics import YOLO
from IPython.display import Image
```

Step 2 : Set Up Your Dataset

Download DataSet From Roboflow

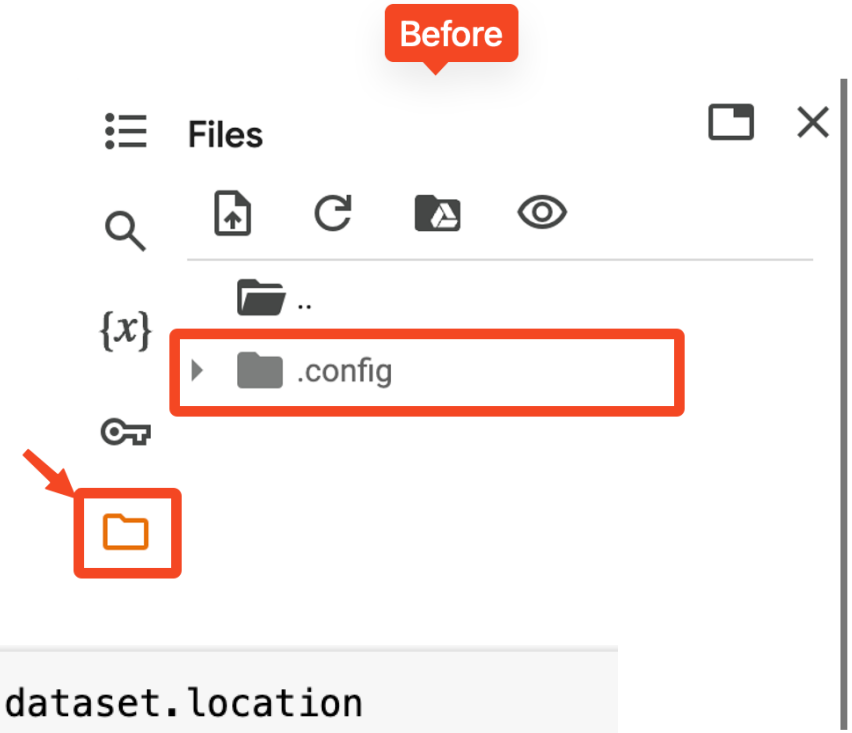
Download Code From Roboflow

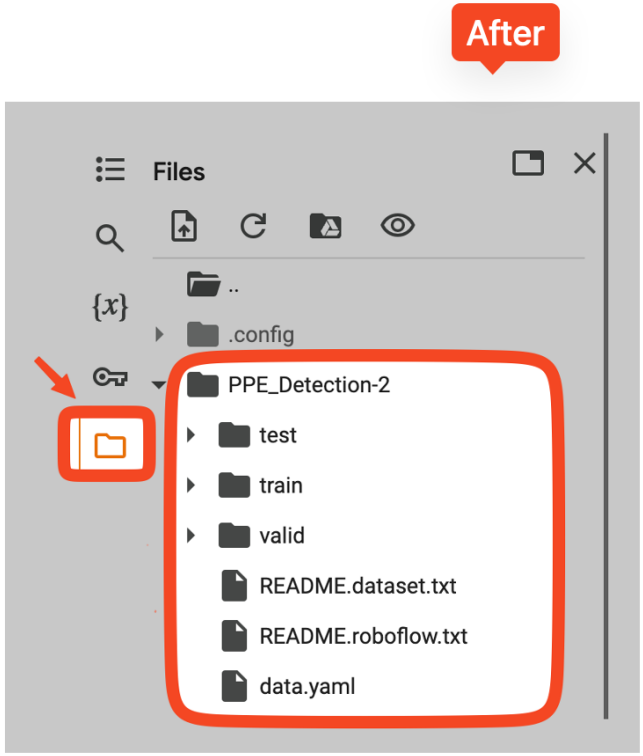
```
!pip install roboflow
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="sCoQ1oYiNNK5RxMwVs72")
project = rf.workspace("project-uyrxf").project("ppe_detection-v1x3l")
version = project.version(2)
dataset = version.download("yolov11")
```

Step 2 : Set Up Your Dataset

Download DataSet From Roboflow






```
dataset.location
```

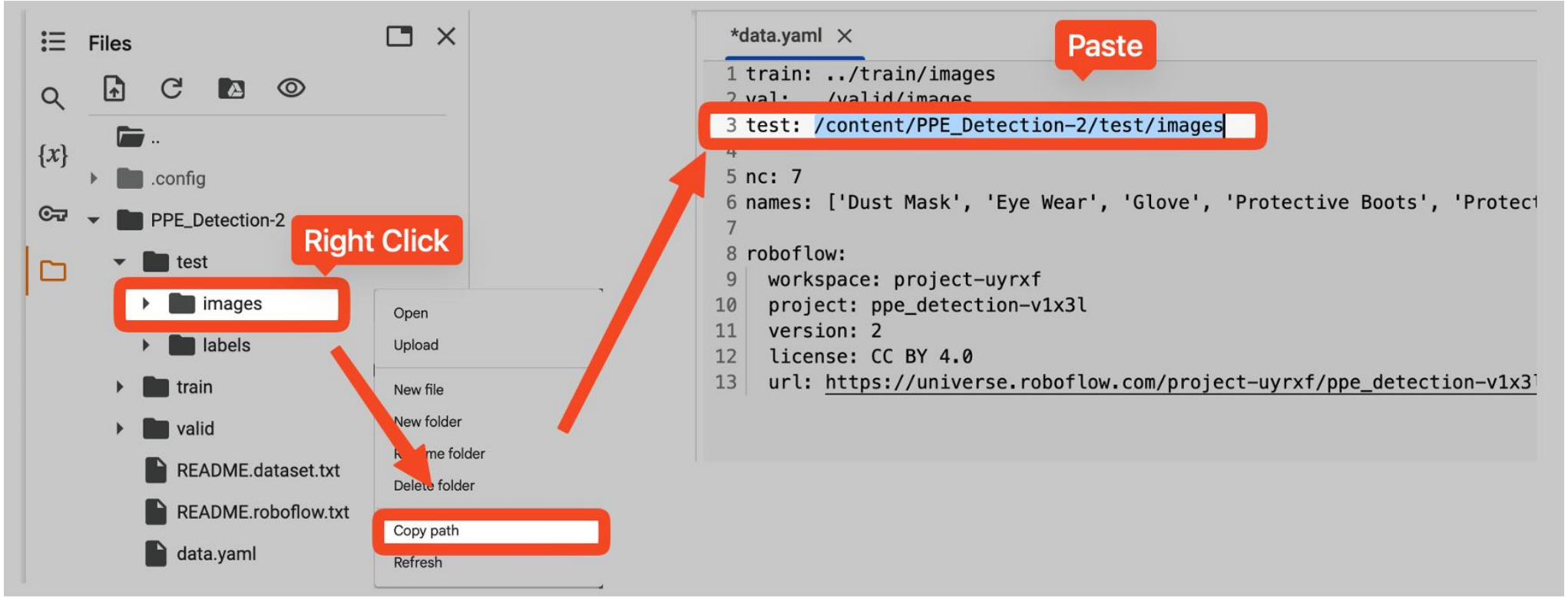
```
'/content/PPE_Detection-2'
```

Step 2 : Set Up Your Dataset

Prepare YOLOv11 Setting (.yaml)

Step 2 : Set Up Your Dataset

Prepare YOLOv11 Setting (.yaml)



Step 2 : Set Up Your Dataset

Prepare YOLOv11 Setting (.yaml)

data.yaml

```yaml
train: /content/PPE_Detection-2/train
val: /content/PPE_Detection-2/valid
test: /content/PPE_Detection-2/test/images

nc: 7
names: ['Dust Mask', 'Eye Wear', 'Glove', 'Protective Boots', 'Protective Helmet', 'Safety

roboflow:
  workspace: project-uyrxf
  project: ppe_detection-v1x3l
  version: 2
  license: CC BY 4.0
  url: https://universe.roboflow.com/project-uyrxf/ppe_detection-v1x3l/dataset/2
```

Copy images Path
1. Train
2. Val
3. Test

Step 3 : Train the YOLOv11 Model_Default

```
!yolo task=detect mode=train data={dataset.location}/data.yaml model="yolo11n.pt" epochs=40 imgsz=640
```

Explanation of Parameters:

- `!yolo` → Calls the YOLO CLI from the `ultralytics` package.
- `task=detect` → Specifies **Object Detection** as the task.
- `mode=train` → Sets the mode to **training**.
- `data={dataset.location}/data.yaml` → Points to the dataset configuration file (`data.yaml`), which contains paths to train/val/test sets and class labels.
- `model="yolo11n.pt"` → Uses the **YOLOv11 Nano** model (`n` = nano, optimized for speed and low computational cost).
- `epochs=40` → Trains the model for **40 epochs** (iterations over the dataset).
- `imgsz=640` → Resizes input images to **640x640 pixels**.

What This Command Does:

1. Loads the **YOLOv11 Nano** model (`yolo11n.pt`).
2. Reads the dataset configuration from `{dataset.location}/data.yaml`.
3. Trains the model for **40 epochs** on the specified dataset.
4. Uses images of size **640x640 pixels**.
5. Saves the trained model weights (`best.pt`) in `runs/detect/train/weights/` for later use.

Step 3 : Train the YOLOv11 Model_Hyper Parameter

```
!yolo task=detect mode=train data={dataset.location}/data.yaml model="yolo11n.pt" epochs=40 imgsz=640 batch=16
workers=4 optimizer=SGD lr0=0.01 lrf=0.1 momentum=0.937 weight_decay=0.0005
warmup_epochs=3 warmup_momentum=0.8 warmup_bias_lr=0.1 box=0.05 cls=0.5 obj=1.0
hsv_h=0.015 hsv_s=0.7 hsv_v=0.4 degrees=0.0 translate=0.1 scale=0.5 shear=0.0 flipud=0.0 fliplr=0.5 mosaic=1.0 mixup=0.0
```

Explanation of Hyperparameters:

- `batch=16` → Number of images per batch (adjust based on GPU memory).
- `workers=4` → Number of workers for data loading.
- `optimizer=SGD` → Optimizer type (`SGD` or `Adam`).
- `lr0=0.01` → Initial learning rate.
- `lrf=0.1` → Final learning rate as a fraction of `lr0`.
- `momentum=0.937` → SGD momentum.
- `weight_decay=0.0005` → L2 regularization (prevents overfitting).
- `warmup_epochs=3` → Number of warmup epochs.
- `warmup_momentum=0.8` → Warmup momentum.
- `warmup_bias_lr=0.1` → Warmup bias learning rate.
- `box=0.05` → Box loss gain.
- `cls=0.5` → Class loss gain.
- `obj=1.0` → Object loss gain.

- **Augmentation:**
  - `hsv_h=0.015`, `hsv_s=0.7`, `hsv_v=0.4` → HSV color augmentation.
  - `degrees=0.0` → Rotation augmentation.
  - `translate=0.1` → Translation augmentation.
  - `scale=0.5` → Scale augmentation.
  - `shear=0.0` → Shear augmentation.
  - `flipud=0.0` → Vertical flip probability.
  - `fliplr=0.5` → Horizontal flip probability.
  - `mosaic=1.0` → Mosaic augmentation probability.
  - `mixup=0.0` → Mixup augmentation probability.

Step 3 : Train the YOLOv11 Model_Hyper Parameter

```
!yolo task=detect mode=train data={dataset.location}/data.yaml model="yolo11n.pt" epochs=40 imgsz=640 batch=16
workers=4 optimizer=SGD lr0=0.01 lrf=0.1 momentum=0.937 weight_decay=0.0005
warmup_epochs=3 warmup_momentum=0.8 warmup_bias_lr=0.1 box=0.05 cls=0.5 obj=1.0
hsv_h=0.015 hsv_s=0.7 hsv_v=0.4 degrees=0.0 translate=0.1 scale=0.5 shear=0.0 flipud=0.0 fliplr=0.5 mosaic=1.0 mixup=0.0
```

General Training Parameters

These parameters control the basic settings of the training process.

- `task=detect` → Specifies that the task is object detection.
- `mode=train` → Specifies that we are in training mode.
- `data={dataset.location}/data.yaml` → Path to the dataset configuration file.
- `model="yolo11n.pt"` → The pre-trained YOLO model to start training from.
- `epochs=40` → The number of training epochs (full passes over the dataset).
- `imgsz=640` → The input image size (640x640 pixels).

Hardware & Resource Management

These parameters optimize training based on available hardware.

- `batch=16` → Number of images processed per batch. Higher values require more GPU memory.
- `workers=4` → Number of CPU workers for data loading.

Optimizer & Learning Rate

These parameters control how the model updates its weights.

- `optimizer=SGD` → Specifies the optimization algorithm:
  - **SGD (Stochastic Gradient Descent)**: Works well for large datasets.
  - **Adam (Adaptive Moment Estimation)**: Can adapt learning rates dynamically but requires more memory.
- `lr0=0.01` → Initial learning rate (how much the model updates weights per step).
- `lrf=0.1` → Final learning rate (as a fraction of `lr0`, helps with fine-tuning).
- `momentum=0.937` → Momentum factor for SGD (helps accelerate training in the right direction).
- `weight_decay=0.0005` → L2 regularization strength (prevents overfitting).

Step 3 : Train the YOLOv11 Model_Hyper Parameter

```
!yolo task=detect mode=train data={dataset.location}/data.yaml model="yolo11n.pt" epochs=40 imgsz=640 batch=16
workers=4 optimizer=SGD lr0=0.01 lrf=0.1 momentum=0.937 weight_decay=0.0005
warmup_epochs=3 warmup_momentum=0.8 warmup_bias_lr=0.1 box=0.05 cls=0.5 obj=1.0
hsv_h=0.015 hsv_s=0.7 hsv_v=0.4 degrees=0.0 translate=0.1 scale=0.5 shear=0.0 flipud=0.0 fliplr=0.5 mosaic=1.0 mixup=0.0
```

Warmup Parameters

Warmup helps the model gradually adjust learning rates at the beginning of training.

- `warmup_epochs=3` → Number of warmup epochs (gradually increasing learning rate).
- `warmup_momentum=0.8` → Momentum during warmup.
- `warmup_bias_lr=0.1` → Learning rate for bias layers during warmup.

Loss Function Parameters

These control how errors are calculated during training.

- `box=0.05` → Weight assigned to bounding box regression loss.
- `cls=0.5` → Weight assigned to classification loss.
- `obj=1.0` → Weight assigned to objectness loss.

Step 3 : Train the YOLOv11 Model_Hyper Parameter

```
!yolo task=detect mode=train data={dataset.location}/data.yaml model="yolo11n.pt" epochs=40 imgsz=640 batch=16
workers=4 optimizer=SGD lr0=0.01 lrf=0.1 momentum=0.937 weight_decay=0.0005
warmup_epochs=3 warmup_momentum=0.8 warmup_bias_lr=0.1 box=0.05 cls=0.5 obj=1.0
hsv_h=0.015 hsv_s=0.7 hsv_v=0.4 degrees=0.0 translate=0.1 scale=0.5 shear=0.0 flipud=0.0 fliplr=0.5 mosaic=1.0 mixup=0.0
```

Data Augmentation Parameters

These settings apply transformations to the dataset to improve generalization.

- **Color Augmentation (HSV adjustments)**:
  - `hsv_h=0.015` → Hue adjustment range.
  - `hsv_s=0.7` → Saturation adjustment range.
  - `hsv_v=0.4` → Brightness adjustment range.
- **Geometric Transformations**:
  - `degrees=0.0` → Random rotation range (0 means no rotation).
  - `translate=0.1` → Random translation range (moving the object within 10% of the image size).
  - `scale=0.5` → Scaling range (up to 50% larger or smaller).
  - `shear=0.0` → Shear range (distortion applied diagonally).
- **Flipping Augmentations**:
  - `flipud=0.0` → Probability of vertical flipping (0 means no flipping).
  - `fliplr=0.5` → Probability of horizontal flipping (50% chance).
- **Mosaic & Mixup (Advanced Augmentations)**:
  - `mosaic=1.0` → Mosaic augmentation probability (merging multiple images together).
  - `mixup=0.0` → Mixup augmentation probability (blending two images together).

Step 4 : Evaluate and Visualize the Model

```
!yolo task=detect mode=val model="best.pt" data={dataset.location}/data.yaml
```

Explanation of Parameters:

- `!yolo` → Calls the YOLO CLI from the `ultralytics` package.
- `task=detect` → Specifies **Object Detection** as the task.
- `mode=val` → Runs the **validation** process to assess model performance.
- `model="best.pt"` → Uses the **best-trained model weights** from the training process (`best.pt`).
- `data={dataset.location}/data.yaml` → Points to the dataset configuration file (`data.yaml`), which includes paths to train, validation, and test datasets.

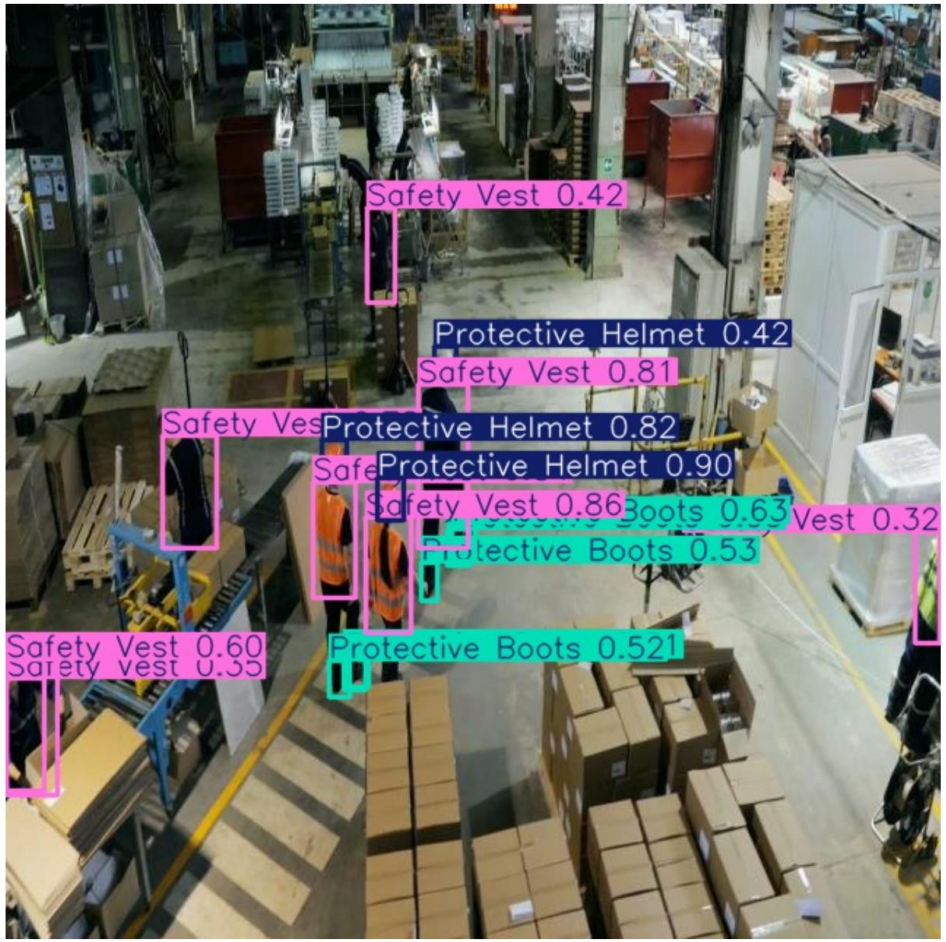
What This Command Does:

1. Loads the **trained model weights** (`best.pt`).
2. Reads the **validation dataset** specified in `{dataset.location}/data.yaml`.
3. Evaluates the model's performance using standard metrics such as:
   - **mAP (mean Average Precision)**
   - **Precision & Recall**
   - **Confusion Matrix**
4. Displays results in **runs/detect/val/**, including:
   - Performance plots
   - Validation images with predicted bounding boxes

Inference with Custom Model on Images

File : Image2.jpg

```
Image("/content/runs/detect/predict2/image2.jpg", width=600)
```



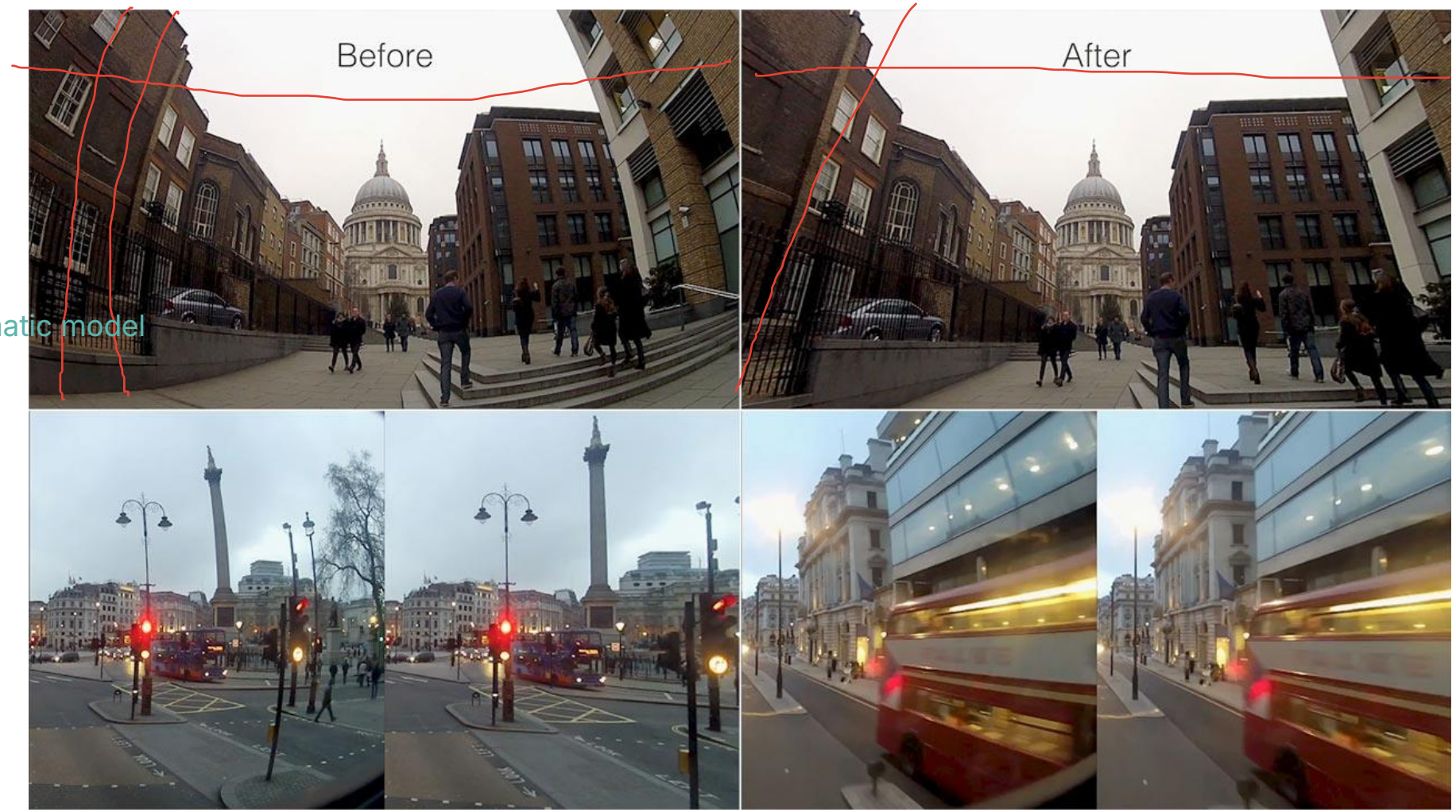
Dimension Inspection, Bin picking, Coordinate search
Pixel to distance[m/mm/um/nm]

Image Distortion

from lens

find camera's mathmatic model

and undistortion

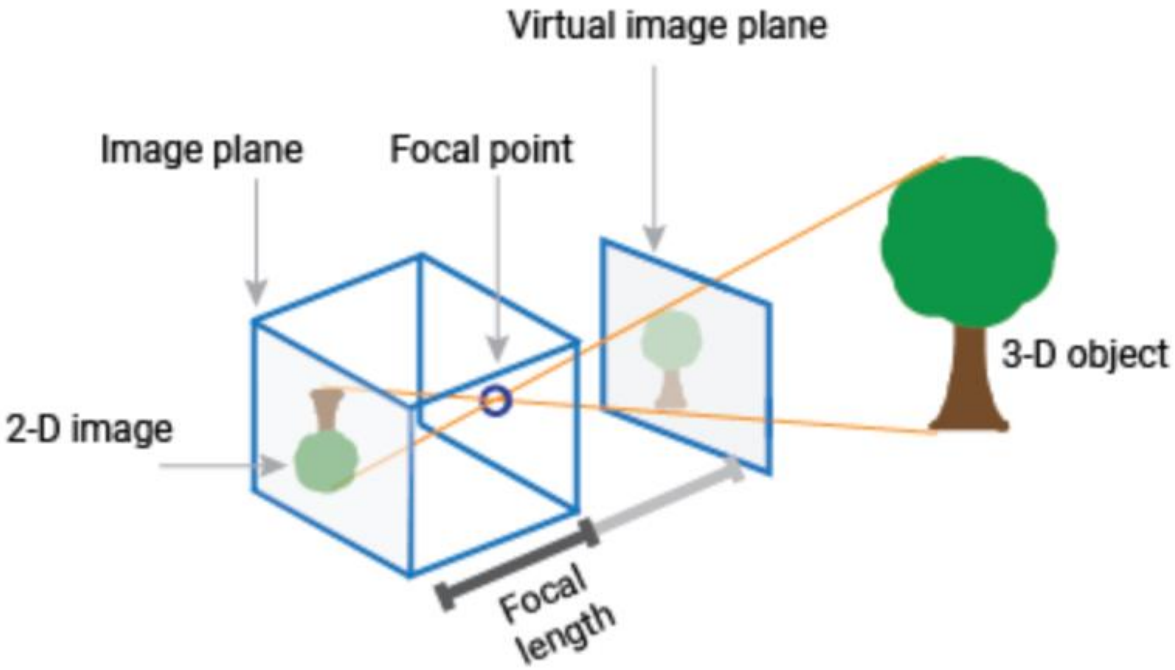


Camera Calibration

A camera introduces distortions due to lens properties, affecting measurement accuracy. Camera calibration helps correct distortions and establishes a relationship between image pixels and real-world measurements.

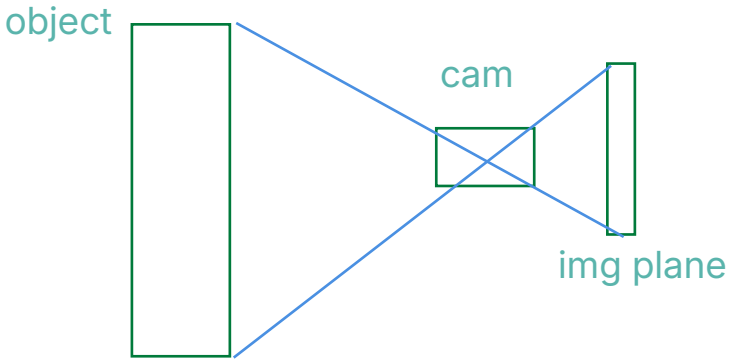
Camera Calibration

Camera Calibration is the process of determining the **intrinsic** and **extrinsic** parameters of a camera to correct distortions and establish a mapping between 3D world coordinates and 2D image coordinates.



Why is Camera Calibration Needed?

1. **Correcting Lens Distortion**: Cameras introduce **radial** and **tangential distortions** that warp the image.
2. **Determining Camera Parameters**:
    1. **Intrinsic parameters** i distortion relationship within camera -> object and image plane -> from calibation
    2. **Extrinsic parameters** e relationship of outside cam systems -> move position, angle
3. **3D Reconstruction & Pose Estimation**: Required for applications like **machine vision, robotics, and AR**.



Mathematical Model of Camera Calibration

The relationship between a **3D point in the world (X, Y, Z)** and its **2D projection (u, v) in an image** is modeled as:

depth cam got 3D x, y, depth

2D img

obj 3D coordinate

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

R11	R12	R13	dx
R21	R22	R23	dy
R31	R32	R33	dz
0	0	0	1

- (X,Y,Z) = 3D coordinates of the point in the world
- (u,v) = 2D coordinates in the image
- s = Scale factor
- K = **Intrinsic Matrix** (camera parameters)
- [R|T] = **Extrinsic Matrix** (rotation & translation)
- R = Rotation matrix (3×3) Rx, Ry Rz
- T = Translation vector (3×1) Tx Ty Tz

offset img frame & object frame

Intrinsic Camera Matrix *K*

distance from cam to img plane

focal length

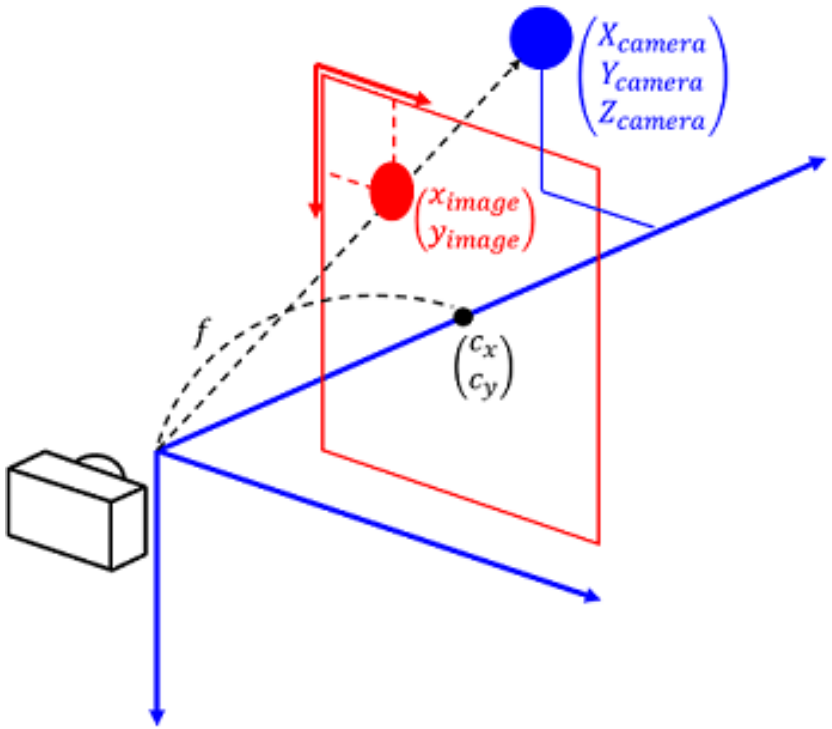
fx value close to fy

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

optical img center

Intrinsic parameters are related to the physical characteristics of the camera, such as the sensor and lens. **They do not depend on the camera's position in the real world.**

1. **Focal Length (f_x, f_y):**
   - Defines how much the camera focuses light onto the sensor, measured in pixels.
   - f_x and f_y may differ if the sensor pixels are not square.
2. **Principal Point (c_x, c_y):**
   - The optical center of the image where light converges on the sensor.
   - It may not be perfectly centered on the sensor.
3. **Distortion Coefficients:**
   - Used to correct lens distortions such as **Barrel Distortion (outward bulging)** and **Pincushion Distortion (inward**

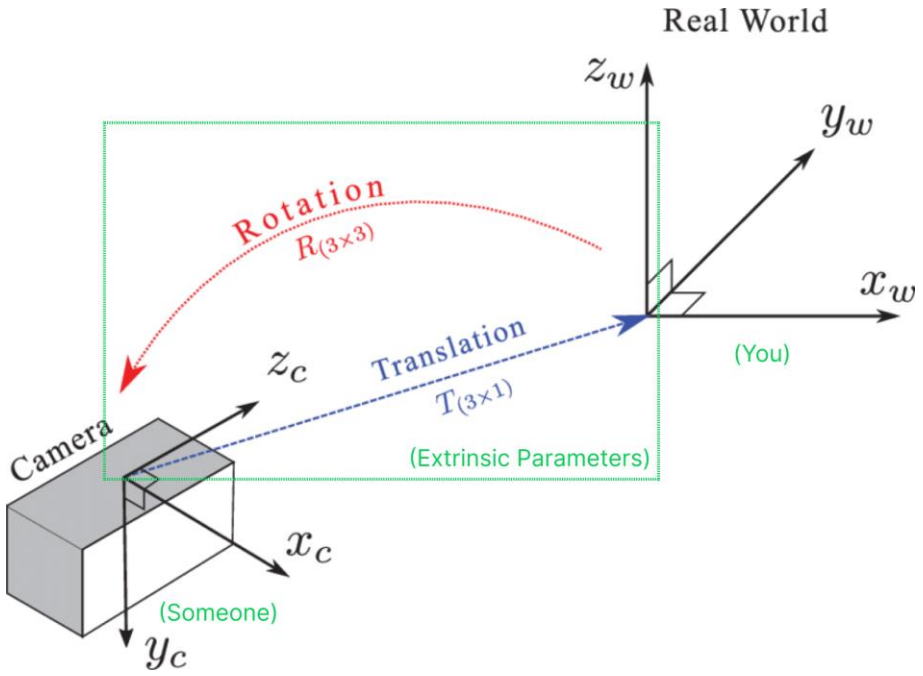


Extrinsic Parameters (Camera Position and Orientation)

Extrinsic parameters describe the position and orientation of the camera in the real world (relative to world coordinates).

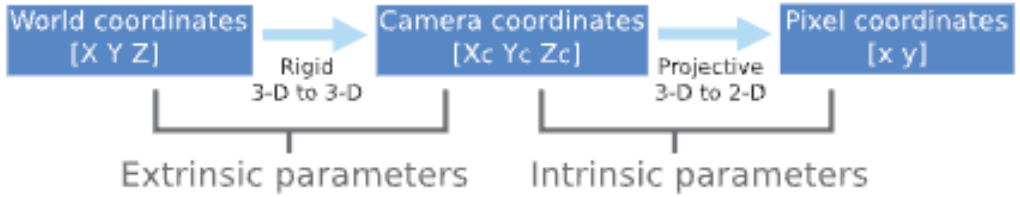
1. **Rotation Matrix (R):**
   - Defines **the direction the camera is facing** (rotation around X, Y, Z axes).
   - Used to change the camera's perspective.
2. **Translation Vector (T):**
   - Defines **the camera's position** in the world coordinate system (X, Y, Z).
   - Used to shift the camera's location.

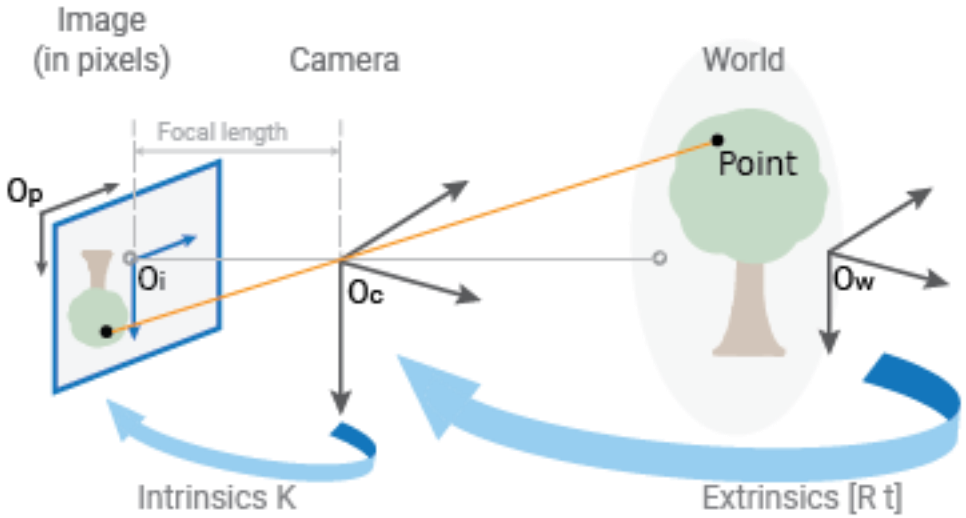
$$[R|T] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$



Extrinsic and Intrinsic Parameters

- **Intrinsic Parameters** = "camera settings," like lens type, zoom, and sensor properties.
- **Extrinsic Parameters** = "camera movement," such as changing the viewing angle or moving the camera position.





Distortion in Camera Calibration

In real-world applications, camera lenses do not produce perfectly accurate images. Distortions affect the accuracy of object positioning calculations.

Types of Distortion

- Radial Distortion
- Tangential Distortion

Radial Distortion

Insert text here

Caused by light bending when passing through the lens, affecting straight lines in the real world.

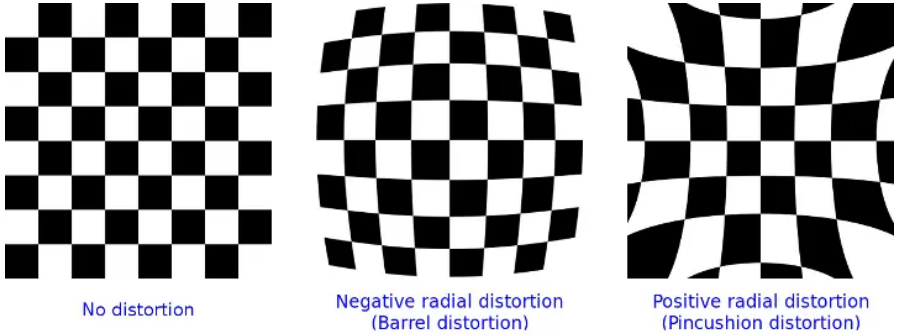
Equation for Distortion Correction :

$$x_{corrected} = x(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

$$y_{corrected} = y(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

$R^2 = x^2+y^2$ (Distance form the Center)

k1,k2,k3 = Radial Distortion Coefficients



Tangential Distortion

Caused by light bending when passing through the lens, affecting straight lines in the real world

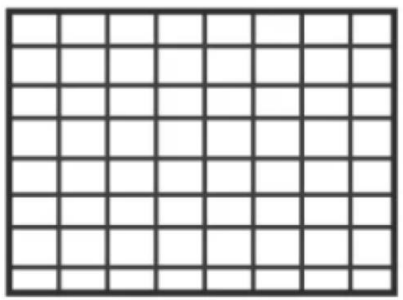
Equation for Distortion Correction :

$$x_{corrected} = x + (2p_1xy + p2(r^2 + 2x^2)$$

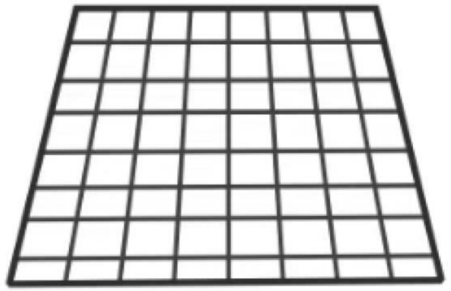
$$y_{corrected} = y + (p1(r^2 + 2y^2 + k2p_2xy)$$

$r^2 = x^2+y^2$ (Distance form the Center)

P_1,p_2= Tangential Distortion Coefficients



Original Image



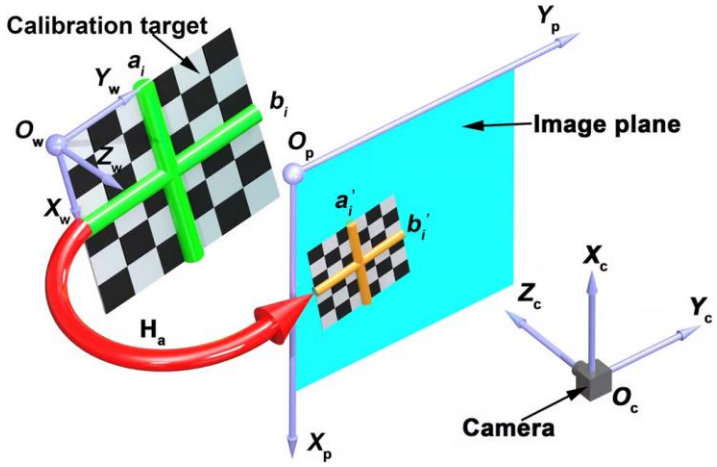
Tangential Distortion

Steps to Correct Distortion

1. Capture images using the camera
    - Take multiple images from different angles using a Checkerboard Pattern or Grid Pattern.
    - A higher number of images improves accuracy.
2. Use software to compute camera parameters
3. Calculate lens distortion values
4. Apply distortion correction (Undistortion)

Calibration Method

- Zhang's Method: Introduced in 1998, it uses multiple planar images to estimate camera parameters. Known for its robustness and accuracy.
- Tsai's Method: Introduced in 1987, it uses a single image with a coplanar calibration object. Simpler but less accurate than Zhang's method.
- DeepCalib: A deep learning-based method that estimates intrinsic parameters from a single image. Suitable for wide field-of-view cameras.
- Wand Calibration: Uses a wand with markers to estimate camera parameters. Comparable in accuracy to 2D chessboard calibration in underwater applications.
- 2D Chessboard Calibration: Uses a planar chessboard pattern to estimate camera parameters. Widely used due to its simplicity and effectiveness.



Ref : https://www.genspark.ai/

Calibration Method Accuracy and Noise

Accuracy

- Zhang's Method: Achieves high accuracy with a mean Reprojection Error of 0.0283.
- Tsai's Method: Lower accuracy with a mean reprojection error of 2.5077.
- DeepCalib: Lowest accuracy among the three methods with a mean reprojection error of 3.5272.
- 2D Chessboard Calibration: Provides highly accurate results, comparable to wand calibration, especially in controlled environments.

Noise

- Zhang's Method: Robust to noise but accuracy decreases with higher noise variance.
- Tsai's Method: Less robust to noise, with higher variance in reprojection error when noise is added.
- DeepCalib: Not evaluated for noise robustness in the study.
- 2D Chessboard Calibration: Generally robust in controlled environments but can be affected by noise in point correspondences.

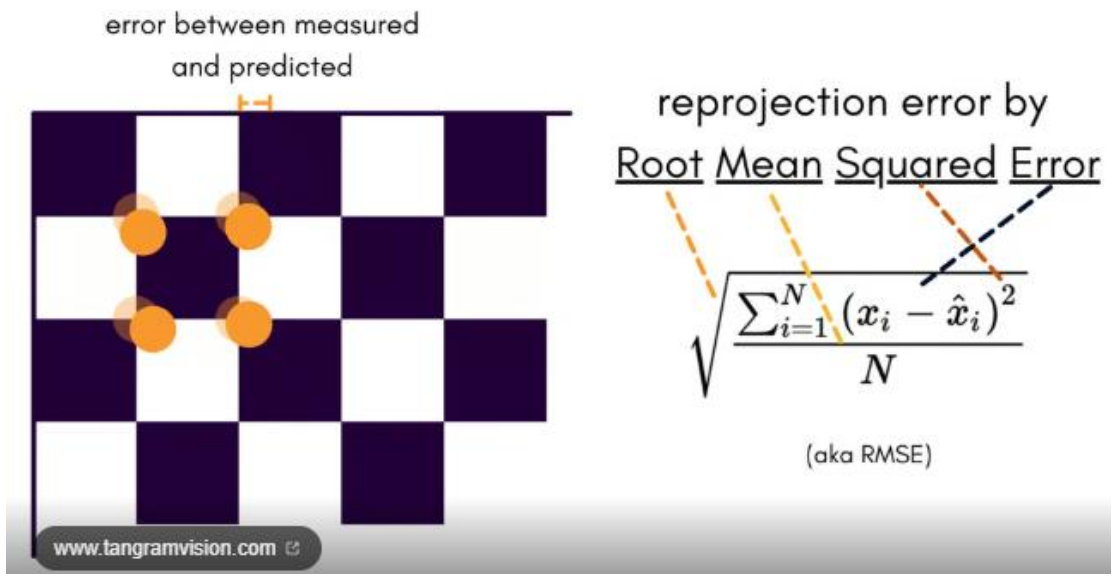
Ref : https://www.genspark.ai/

Chessboard Pattern Calibration

•2D Chessboard Calibration: This method uses a planar chessboard pattern to estimate a camera's intrinsic and extrinsic parameters. It is widely used due to its simplicity and effectiveness.

•Accuracy: Studies indicate that 2D chessboard calibration provides highly accurate results, comparable to other traditional methods like Wand Calibration, especially in controlled environments.

•Comparison with Other Methods: Traditional methods like Zhang's and Tsai's calibration methods are often compared with 2D chessboard calibration. Zhang's Method is noted for its robustness and accuracy, while Tsai's Method is simpler but less accurate.



Ref : https://www.genspark.ai/

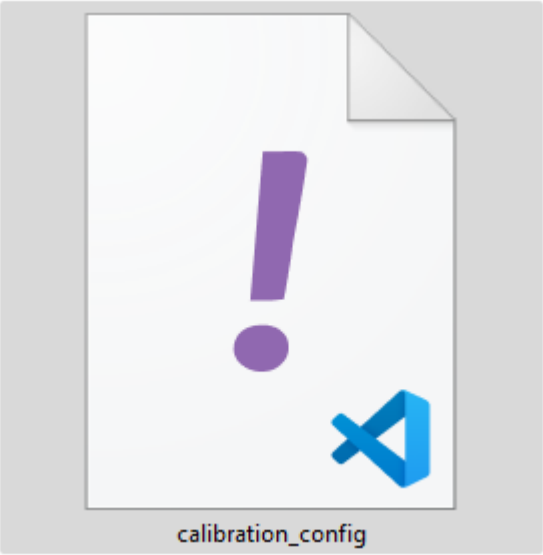
Use software to compute camera parameters

1. Download Software calibration software form
    1. https://solarframework.github.io/use/tools/#tutorial
    2. Unzip

Use software to compute camera parameters

Setting Chessboard Parameter

- File : Calibration_config



calibration_config.yml

File Edit View

```yaml
%YAML:1.0
---
# the number of inner corners on board width
chessboard_width: 9
# the number of inner corners on board height
chessboard_height: 6
# square size in some user-defined units
square_size: 0.026
#fix aspect ratio (fx/fy)
aspect_ratio: 1
# number of frames to calibrate the camera
nb_frames: 10
# OpenCV Flags for camera calibration
flags: 0
# delay between each frame in milliseconds
delay: 2000
```

Use software to compute camera parameters

Setting Camera Port

- File : Calibration_config

SolARTool_CameraCalibration_con

File Edit View

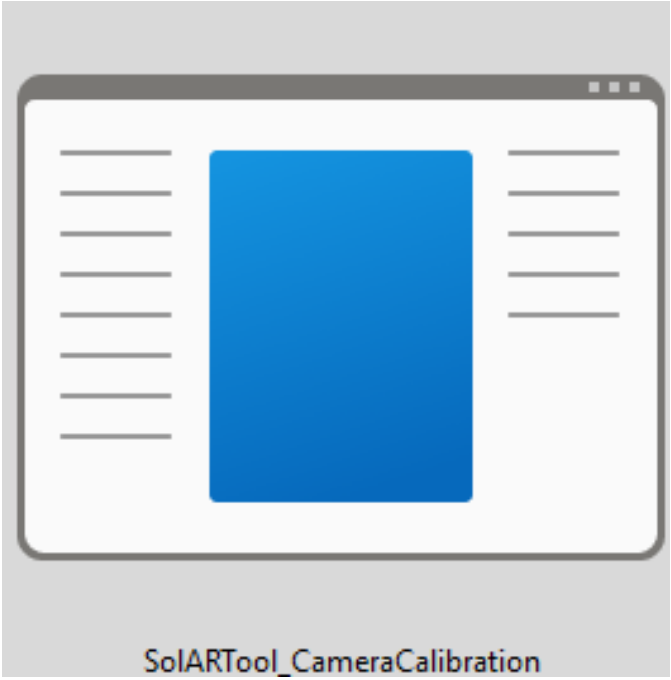
```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<xpcf-registry autoAlias="true">
    <module uuid="15e1990b-86b2-445c-8194-0cbe80ede970" name="SolARModuleOp
        <component uuid="5B7396F4-A804-4F3C-A0EB-FB1D56042BB4" name="SolARC
            <interface uuid="125f2007-1bf9-421d-9367-fbdc1210d006" name="IC
            <interface uuid="5DDC7DF0-8377-437F-9C81-3643F7676A5B" name="IC
        </component>
        <component uuid="b8a8b963-ba55-4ea4-b045-d9e7e8f6db02" name="SolARI
            <interface uuid="125f2007-1bf9-421d-9367-fbdc1210d006" name="IC
            <interface uuid="5DDC7DF0-8377-437F-9C81-3643F7676A5B" name="IC
        </component>
        <component uuid="fa4a780a-9720-11e8-9eb6-529269fb1459" name="SolARV
            <interface uuid="125f2007-1bf9-421d-9367-fbdc1210d006" name="IC
            <interface uuid="5DDC7DF0-8377-437F-9C81-3643F7676A5B" name="IC
        </component>
    </module>

    <factory>
        <bindings>
            <bind interface="ICamera" to="SolARCameraOpencv"/>
            <!--bind interface="ICamera" to="SolARImagesAsCameraOpencv"/-->
            <!--bind interface="ICamera" to="SolARVideoAsCameraOpencv"/-->
        </bindings>
    </factory>

    <properties>
        <configure component="SolARCameraOpe
                        <property name="cali rationFile" type="string" valu
            <property name="deviceID" type="  int" value="0"/>
        </configure>
        <configure component="SolARImagesAsC
            <property name="calibrationFile" type="string" value="camera_ca
                        <property name="imagesDirectoryPath" type="string"
```

Set camera index

Use software to compute camera parameters

Calibration

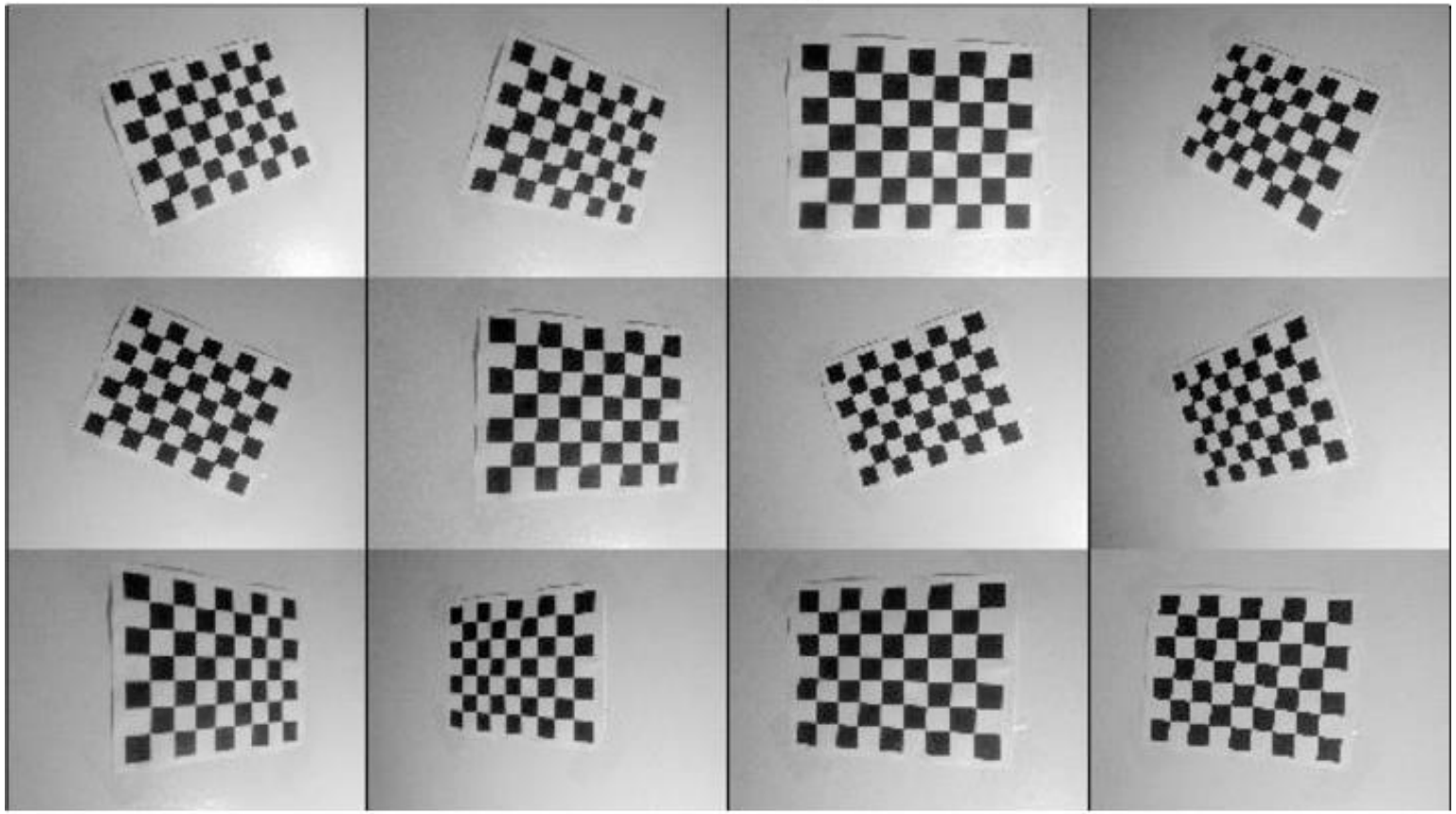
File : SolArTool_CameraCalibration



Calibrate Image

calibration_config.yml
chessboard_width, height ⇒ numbers of crossed point (not wdth x height)
square_size ⇒ size of box

Calibration images



Required image data:
- Various scale
- Various skewness
- Various Translation
- Various Rotation

SolAR_Tool camcal_conf

program confg

Calibration Results

Look for the result in camera_calibration.json

536.223	0	312.731
0	535.986	237.458
0	0	1

$$\begin{pmatrix} f_x & 0 & W/2 \\ 0 & f_y & H/2 \\ 0 & 0 & 1 \end{pmatrix}$$

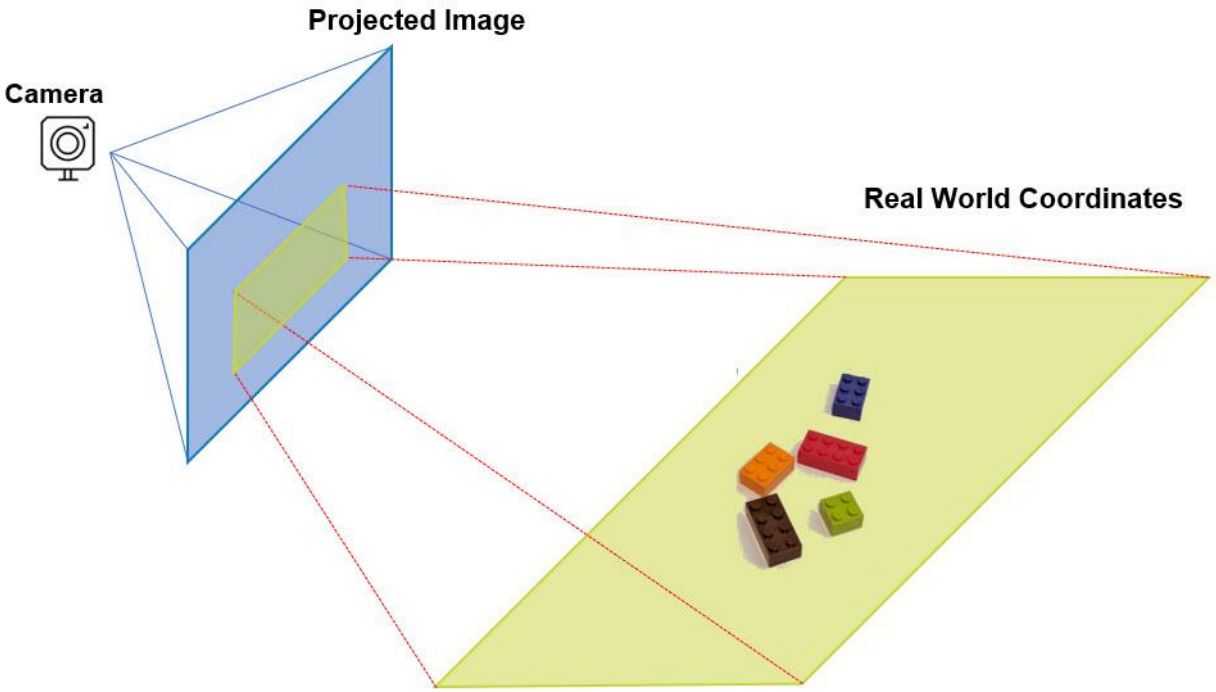
d = [0.1521, -0.592, -0.0009, - 0.0023, 0.6627]

camera_calibration.json

D: > RobotLibrary > CameraCalibration > CameraCalibration_win > camera_calibra


```
{
    "CameraParameters": {
        "name": "Camera",
        "id": 0,
        "type": 0,
        "resolution": {
            "width": 640,
            "height": 480
        },
        "intrinsic": [
            536.2233276367188,
            0.0,
            312.73138427734375,
            0.0,
            535.986083984375,
            237.45838928222656,
            0.0,
            0.0,
            1.0
        ],
        "distortion": [
            0.15210852026939392,
            -0.5921238660812378,
            -0.0009657841874286532,
            -0.002313214587047696,
            0.6627073287963867
        ]
    }
}
```

Camera Coordinate Transformation



Given this — Find This

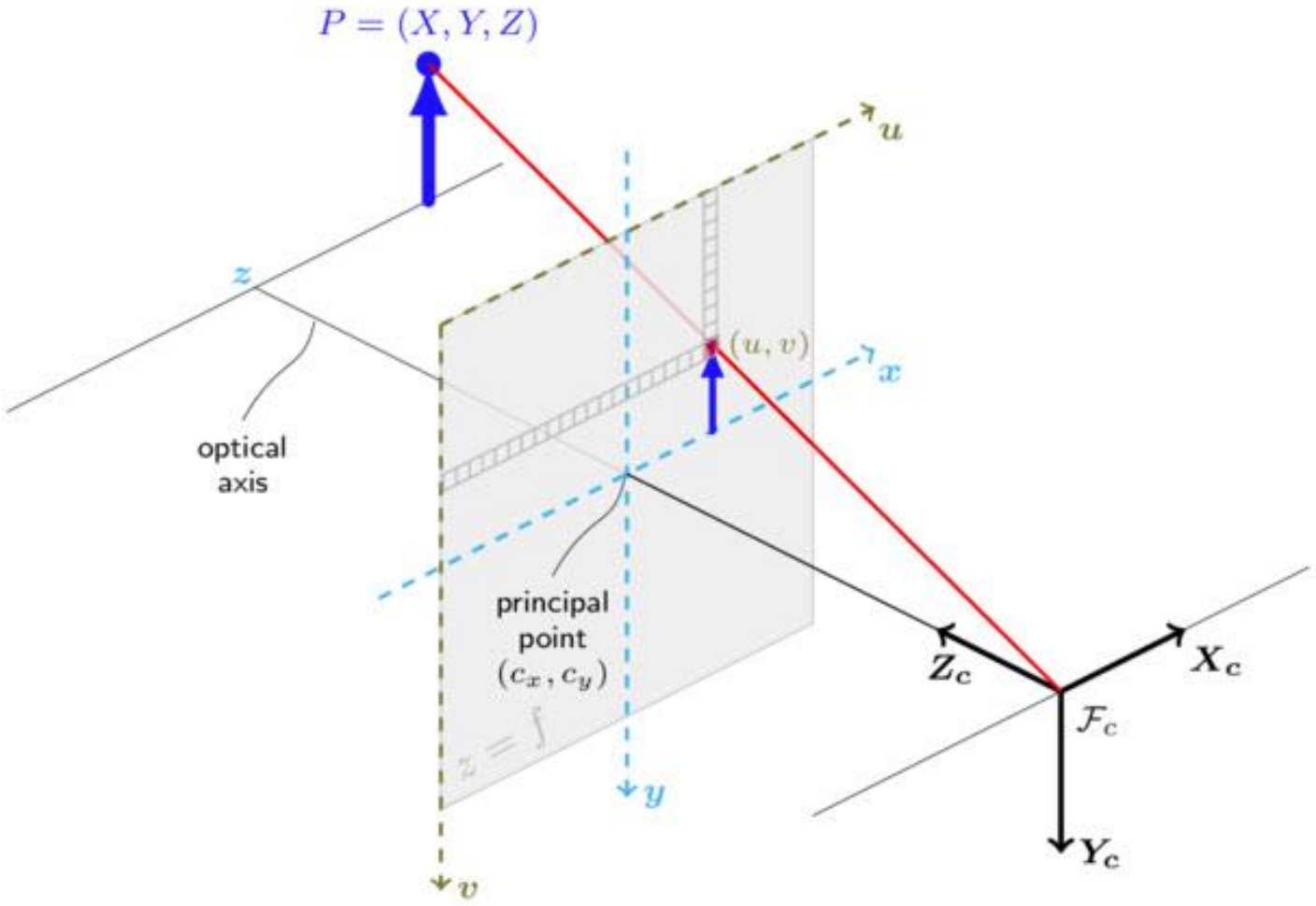
$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where:

- (X, Y, Z) are the coordinates of a 3D point in the world coordinate space
- (u, v) are the coordinates of the projection point in pixels
- A is a camera matrix, or a matrix of intrinsic parameters
- (cx, cy) is a principal point that is usually at the image center
- fx, fy are the focal lengths expressed in pixel units.

https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

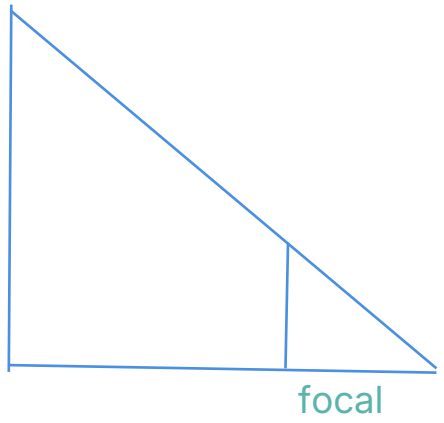
Camera Coordinate Transformation

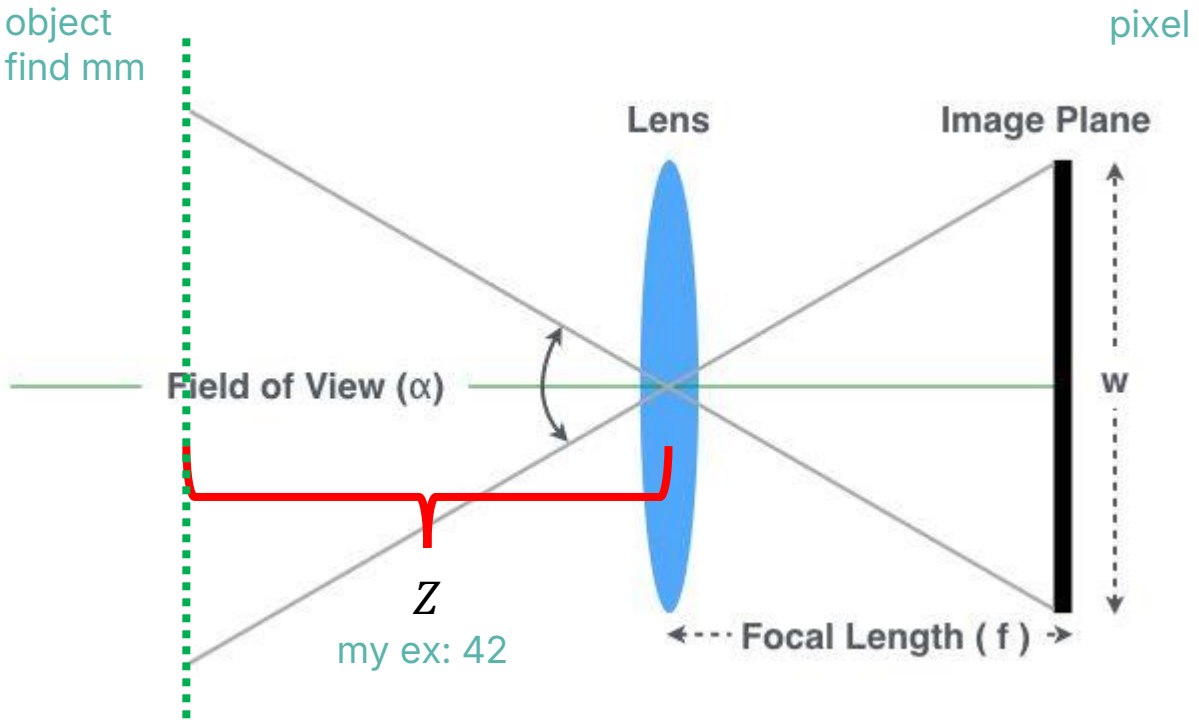


Camera Coordinate Transformation

$$\frac{X_{camera}}{Z} = \frac{X_{image}}{f_x}$$

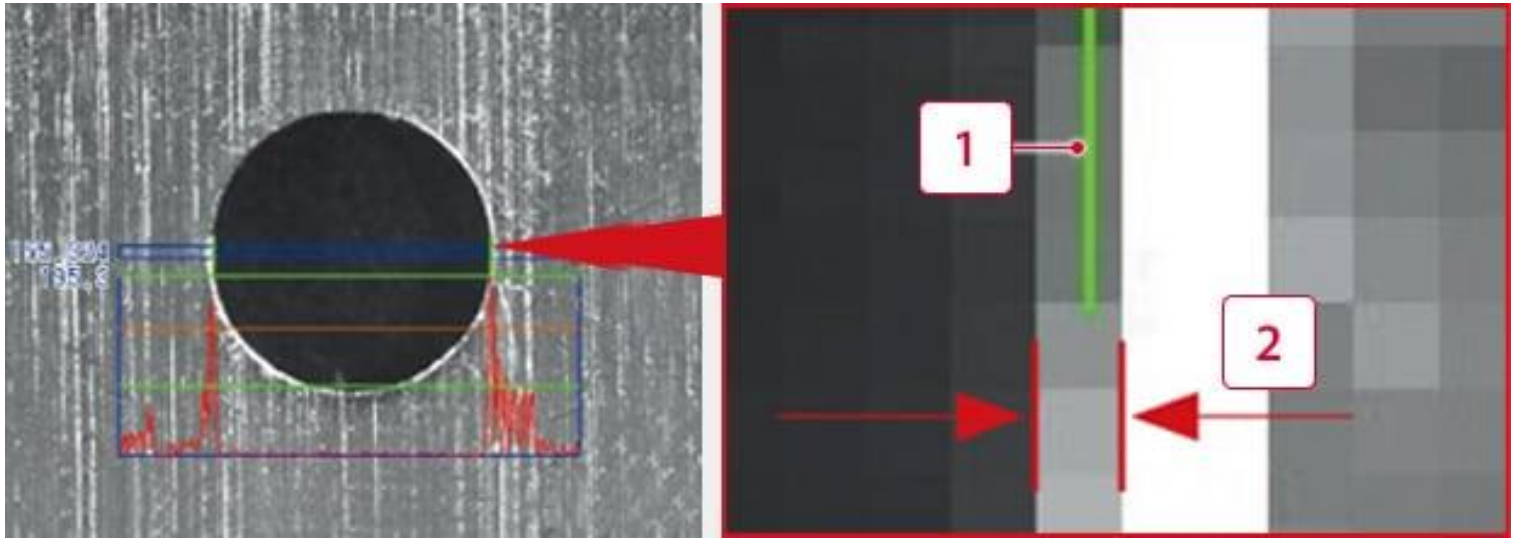
$$\frac{Y_{camera}}{Z} = \frac{Y_{image}}{f_y}$$

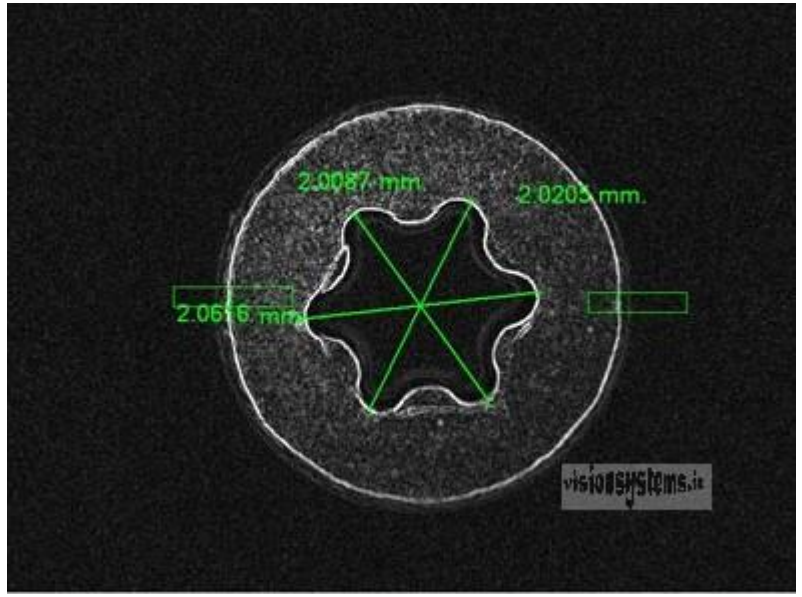




Coordinate Transformation for Inspection

Dimension visual inspection



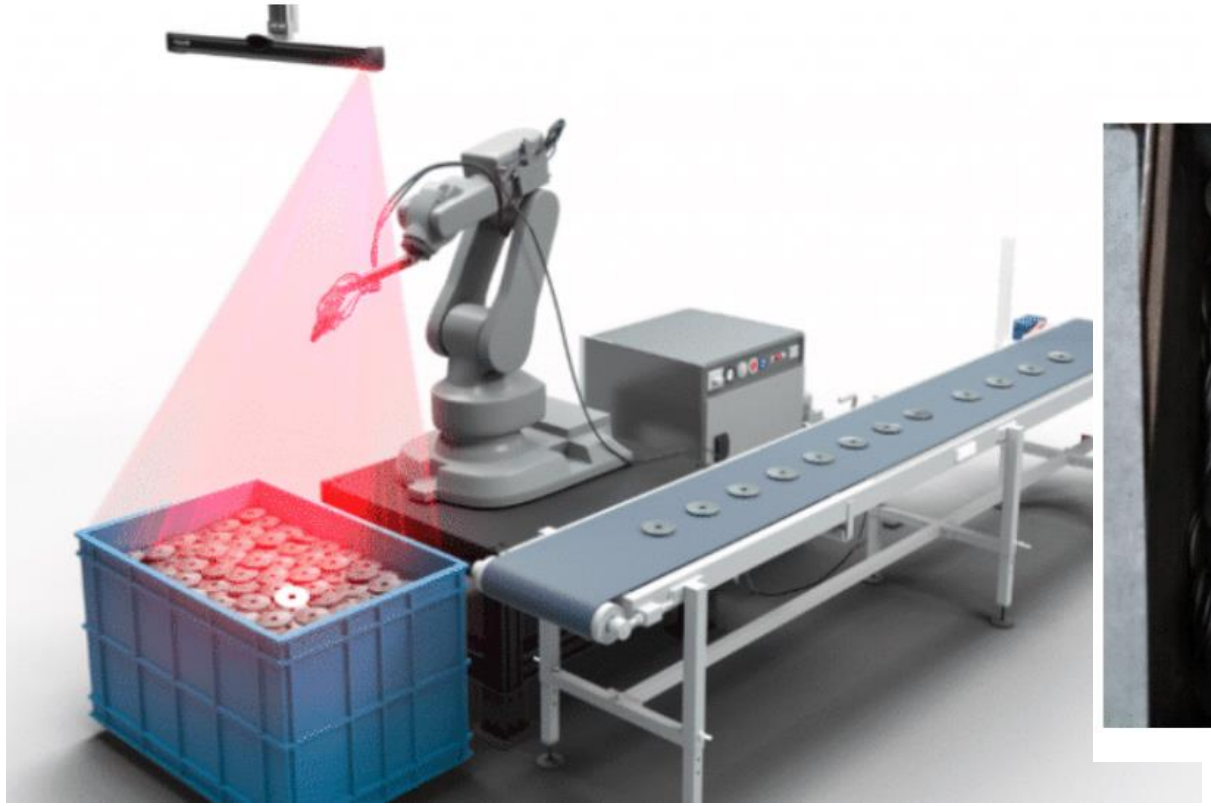


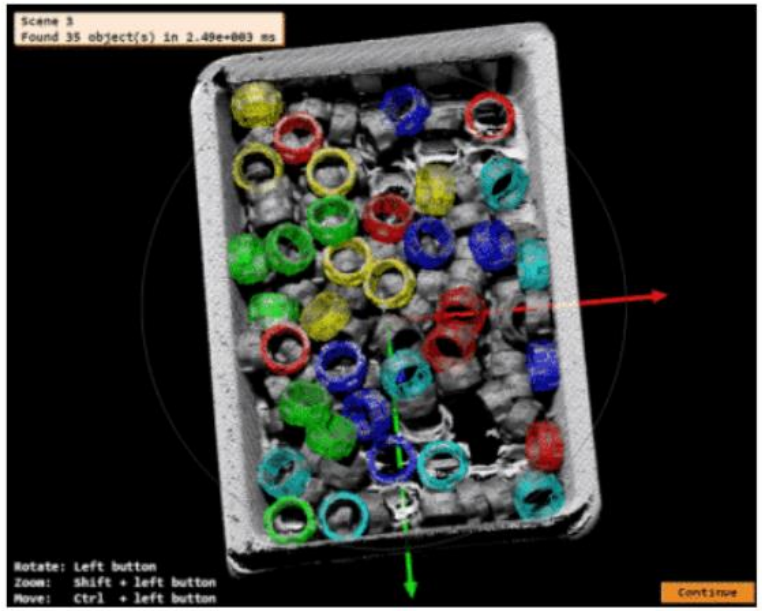
y cam = x robot
x cam = y robot

then add offset

Coordinate Transformation for Inspection

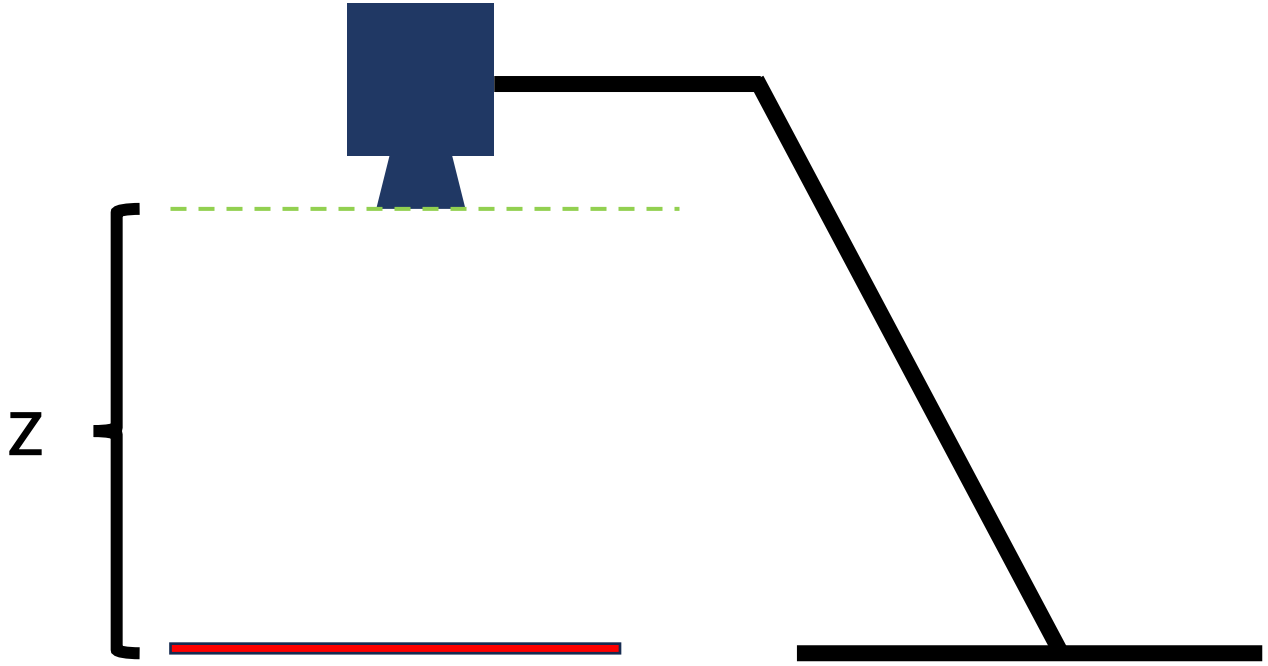
Bin Picking





Example: Dimension Measurement

Create a program to measure the distance of object by the camera and YOLO v11 object detection.



Example: Dimension Measurement

Camera matrix setup

```python
import cv2
import numpy as np
from ultralytics import YOLO

cameraMatrix = np.array([[536.223, 0, 312.731],
                         [0, 535.986, 237.4583],
                         [0,  0,  1]])
distCoeffs = np.array([0.1521, -0.59212, -0.00096, -0.0023132, 0.6627])

fx = cameraMatrix[0, 0]
fy = cameraMatrix[1, 1]
cx = cameraMatrix[0, 2]
cy = cameraMatrix[1, 2]
```

Example: Dimension Measurement

Load YOLO standard model (trained by COCO dataset) and capture image.

```
model = YOLO("yolo11n.pt")

Z = 445
cap = cv2.VideoCapture(1)

cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    undistorted_frame = cv2.undistort(frame, cameraMatrix, distCoeffs)
    results = model(undistorted_frame)
```

Example: Dimension Measurement

Calculate the distance of bounding boxes.

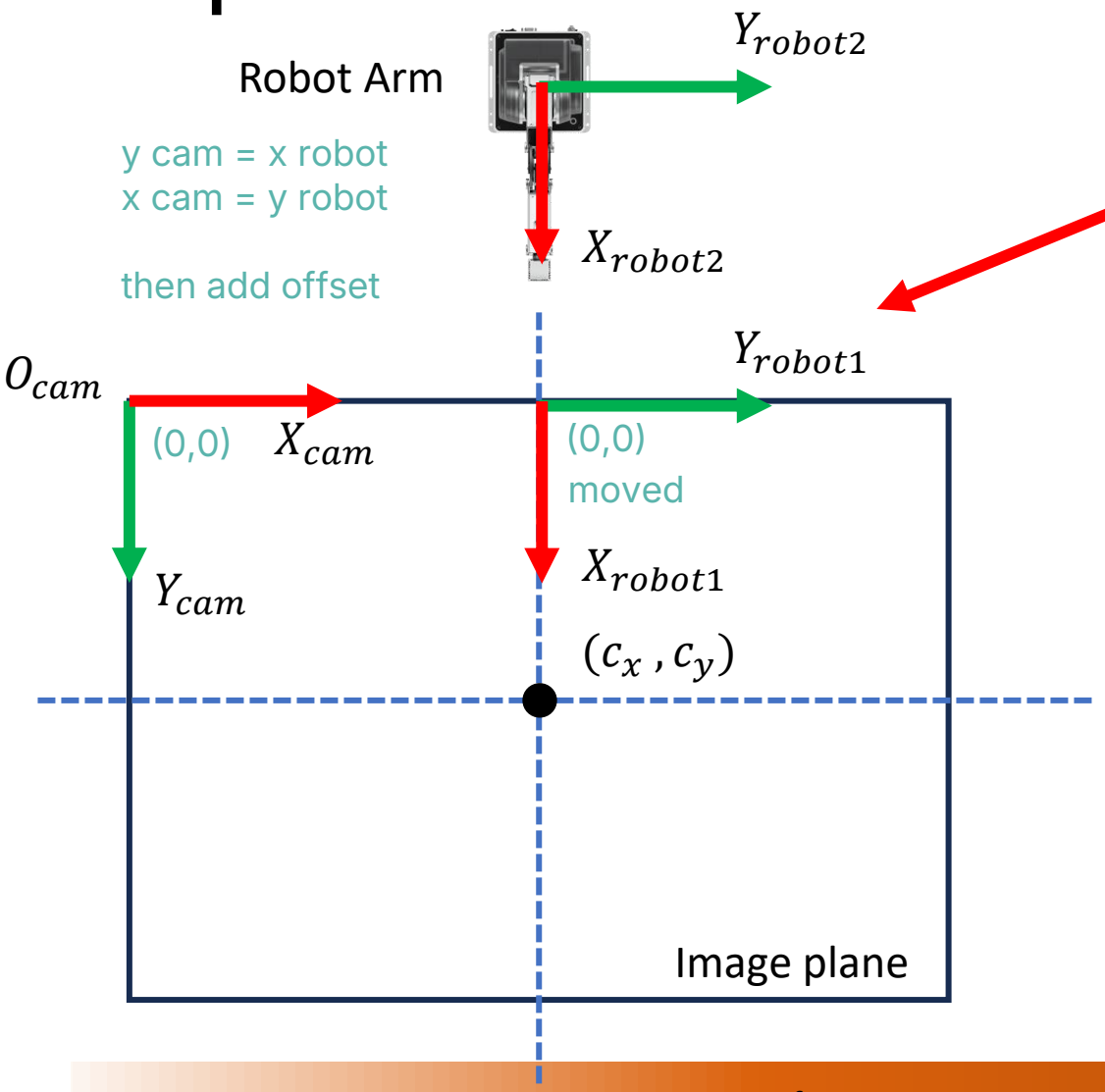
```
for result in results:
    for box in result.boxes:
        x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()

        x_center = (x1 + x2) / 2
        y_center = (y1 + y2) / 2

        distX_cam = abs(x1 - x2)
        distY_cam = abs(y1 - y2)

        D1 = distX_cam / fx * Z
        D2 = distY_cam / fx * Z
```

Example: Dimension Measurement

Display the results

```
            cv2.rectangle(undistorted_frame, (int(x1), int(y1)), (int(x2), int(y2)), (0,0, 255), 2)
            text = f"D1: {D1:.2f} mm, D2: {D2:.2f} mm"
            cv2.putText(undistorted_frame, text, (int(x1), int(y1) - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

    cv2.imshow("Object Detection and Coordinate Estimation", undistorted_frame)
    cv2.waitKey(1)

cap.release()
cv2.destroyAllWindows()
```

Example: Find Coordinate



Create a program to measure the X,Y coordinate of the object in millimeters unit for the robot arm

Example: Find Coordinate

Edit calculation part of the program

```
32    for result in results:
33        for box in result.boxes:
34            x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
35
36            x_center = (x1 + x2) / 2
37            y_center = (y1 + y2) / 2
38
39            X_cam = ((x_center) / fx) * Z
40            Y_cam = ((y_center) / fy) * Z
41
42            X_real = Y_cam
43            Y_real = (X_cam - cx)
44
45            cv2.rectangle(undistorted_frame, (int(x1), int(y1)), (int(x2), int(y2)), (0,0, 255), 2)
46            text = f"X: {X_real:.2f} mm, Y: {Y_real:.2f} mm"
47            cv2.putText(undistorted_frame, text, (int(x1), int(y1) - 10),
48                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
```

SUT
SURANAREE UNIVERSITY OF TECHNOLOGY

INSTITUTE OF
ENGINEERING

Mecha tronics

Modern Automotive