SEEDLING LEAKAGE DETECTION

BASED ON YOLOV5

—— Machine Learning and Computer Vision in Industrial

CHENG HU

ORIGINAL PROJECT IN 2022—2023

WORK SAMPLE

FOR WORKSAFE BC

[SENSITIVE DATA AND PROCESSES HAVE BEEN REMOVED OR MODIFIED]

Table of Contents

1 OVERVIEW

1.1 Project Requirements:

When the transplanting machine manipulator takes the seedlings from the seedling tray, there is often the phenomenon of missing seedlings, resulting in no seedlings in some areas, and the seedlings need to be manually replenished in the later stage, thereby increasing the labor cost. The machine vision correlation model is used to accurately detect whether there is leakage in the seedling cup of the transplanter, and the recognition results are displayed in the video image.

1.2 Purpose & Significance

Use image processing and machine vision.

The above situation involves the use of machine vision and image processing technology to solve the problem of missing seedlings from the transplanter and reduce the subsequent cost of manual seedling replenishment.

The specific analysis is as follows:

（1）Seedling leakage problem of transplanting machine: When the transplanter takes seedlings from the seedling tray, some seedlings may be missed, resulting in no seedlings in some areas. This will increase the labor cost in the later stage, as additional manual seedling replenishment operations are required.

（2）Solution: Through the application of image processing and machine vision technology, it can accurately detect whether there is a leakage in the transplanter seedling cup, so as to find the leakage of seedlings in time and take measures.

（3）Image processing and machine vision technology: These technologies can be applied to video images, and through the analysis and processing of the images, they can identify whether there are leaks in the seedling cups of the transplanter. Common image processing and machine vision technologies include image segmentation, feature extraction, object detection, and classification.

（4）Accurate detection and identification results show: Through the design and training of the algorithm and model, the video image of the transplanting machine can be processed to accurately detect whether there is a leakage of seedlings in the seedling cup. Recognition results can be displayed by annotation, marqueeping, or other means in the video image, allowing the operator to observe and take remedial action in a timely manner.

（5）Benefits & Applications: Accurately detecting leakage in the transplanter's seedling cup can help identify problems in advance and take measures to reduce the cost of subsequent manual seedling replenishment. This technology can be applied to fields such as seedling production and agricultural planting to improve production efficiency and reduce costs.

In short, the use of image processing and machine vision technology to achieve the accurate detection of whether there is a leakage in the transplanter seedling cup, and display the recognition results in the video image, can help solve the problem of transplanter leakage, improve production efficiency, and reduce labor costs.

2 PROPOSAL

2.1 Innovation Idea

According to the relevant demonstration example of YOLOv5, it is feasible to use the design idea of YOLOv5 to solve the problem of transplanter leakage, and use your own customized dataset to train a model for specific purposes.

2.2 Innovative Implementation Guide

Here's a brief step-by-step guide：

(1) Data collection and annotation： Image data of transplanter seedling cups were collected and labeled. The label should include the location of the seedling cup and the bounding box of the missing seedling area. You can use annotation tools such as LabelMe, LabelImg, RectLabel, etc.

(2) Data preparation: Divide the labeled dataset into a training set and a validation set. Make sure that the dataset has the correct category labels and that you have prepared the data format required by YOLOv5.

(3) YOLOv5 Installation: To install YOLOv5, you can get the code and related documentation from its official GitHub repository (https://github.com/ultralytics/yolov5).

(4) Network configuration: Configure the YOLOv5 model according to your own dataset and requirements

(5) Model training: Use the prepared dataset and the configured model for training. By running the training script, you can start training a YOLOv5 model on your own dataset. The training process iteratively adjusts the weights of the model so that it gradually learns to detect the transplanter's seedling cups and missing seedlings.

(6) Model evaluation: During training, you can use validation sets to evaluate the model to understand its performance. The results of the evaluation can help you determine if the model needs further tweaking and improvement.

(7) Model deployment and application: After the training is completed, the trained YOLOv5 model can be applied to the embedded image recognition module (such as Jstson Nano) and imported video images for seedling leakage detection. Or integrate the model into the relevant application or system and display the inspection results in a video image so that the operator can observe and take appropriate remedial action.

3 MODEL TRAINING AND TESTING

3.1 Process

To use YOLOv5 to train a model for leakage detection, you can follow these steps:

（1）Data collection and annotation: Collect image data related to the seedling cups of the transplanting machine and annotate them. The label should include the location of the seedling cup and the bounding box of the leaky seedling area. Annotation tools such as LabelImg, RectLabel, etc. can be used for annotation, but in order to facilitate the direct generation of label files in .txt format, this project uses https://www.makesense.ai/ online labeling website to directly generate label files that meet the YOLOv5 format, as shown in Figure 1.

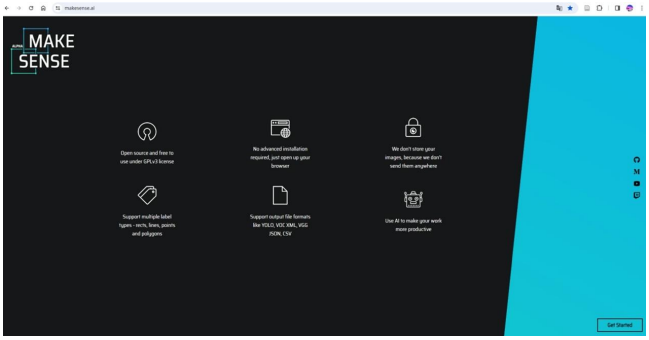


Figure 1

Import the prepared dataset into the website, a total of 1200 images, and define two labels, Ture for normal seedlings and False for missing seedlings. The labeling process is shown in Figure 2.

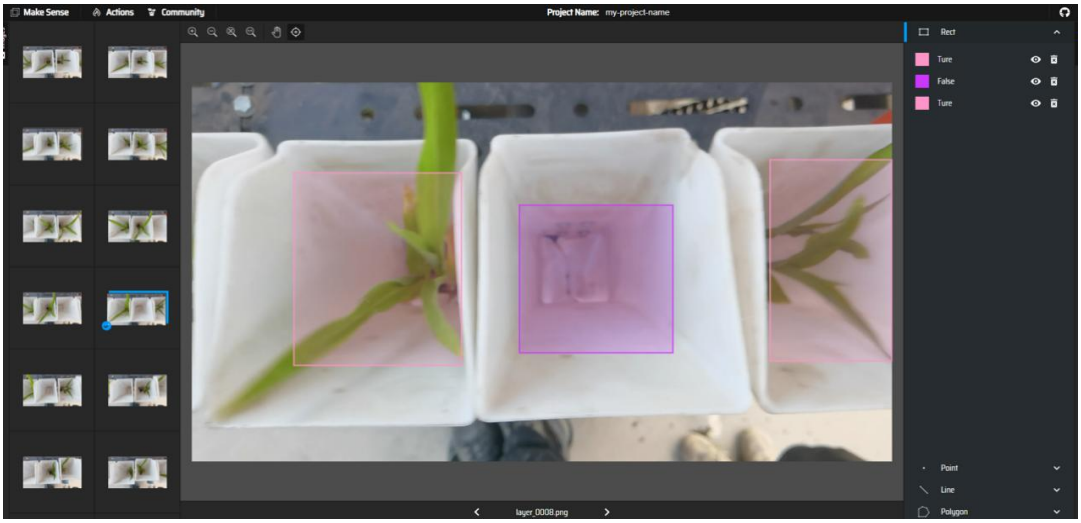


Figure 2

（2）Data preparation: Divide the labeled dataset into a training set, a test set, and a validation set, according to the 8:1:1 division, YOLOv5 requires that the images and the corresponding annotated information be placed in a specific folder, and the file names should be matched one by one.

The label files are also divided into corresponding files.

（3）YOLOv5 Installation: To install YOLOv5, you can get the code and related documentation from its official GitHub repository (https://github.com/ultralytics/yolov5). Follow the instructions in the documentation to install the YOLOv5 dependency library and environment, and this project directly uses pip install -r requirements .txt environment configuration as follows for convenience.

```
# YOLOv5 requirements
# Usage: pip install -r requirements.txt
# Base ------------------------------------------------------------------------
gitpython>=3.1.30
matplotlib>=3.3
numpy>=1.18.5
opencv-python>=4.1.1
Pillow>=7.1.2
psutil  # system resources
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
thop>=0.1.1  # FLOPs computation
torch>=1.7.0  # see https://pytorch.org/get-started/locally (recommended)
torchvision>=0.8.1
tqdm>=4.64.0
ultralytics>=8.0.111
# protobuf<=3.20.1  # https://github.com/ultralytics/yolov5/issues/8012
# Logging ---------------------------------------------------------------------
# tensorboard>=2.4.1
# clearml>=1.2.0
# comet
# Plotting --------------------------------------------------------------------
pandas>=1.1.4
seaborn>=0.11.0
# Export ----------------------------------------------------------------------
# coremltools>=6.0  # CoreML export
# onnx>=1.10.0  # ONNX export
# onnx-simplifier>=0.4.1  # ONNX simplifier
# nvidia-pyindex  # TensorRT export
# nvidia-tensorrt  # TensorRT export
# scikit-learn<=1.1.2  # CoreML quantization
# tensorflow>=2.4.0  # TF exports (-cpu, -aarch64, -macos)
# tensorflowjs>=3.9.0  # TF.js export
# openvino-dev  # OpenVINO export
# Deploy ----------------------------------------------------------------------
setuptools>=65.5.1 # Snyk vulnerability fix
# tritonclient[all]~=2.24.0
# Extras ----------------------------------------------------------------------
# ipython  # interactive notebook
# mss  # screenshots
# albumentations>=1.0.3
# pycocotools>=2.0.6  # COCO mAP
```

This project uses anaconda3 to create the environment and configure it (as shown in Figure 5), the compilation software uses pycharm, the two are used together, the architecture is pytorch, where pytorch should correspond to the cuda version, so first check the cuda version of the system, and then download the corresponding pytorch version on the pytorch official website, as shown in figure 6.

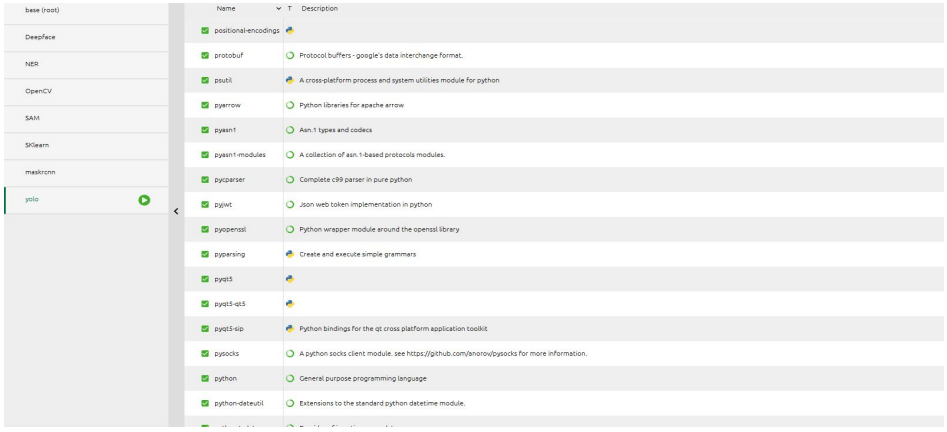


Figure 5

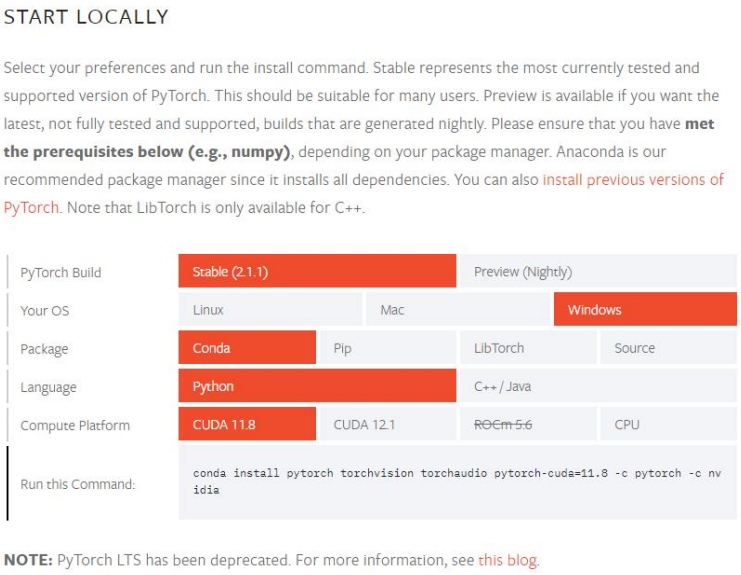


Figure 6

（4）Model/network configuration and Train: Unlike the .cfg configuration file in Yolov3/Yolov4, the Yolov5 parameter configuration file is in .yaml format, and in YOLOv5, the model configuration file is 'yolov5/models/yolov5.yaml'. Depending on your dataset and needs, configure the model's parameters, such as network structure, input resolution, training hyperparameters, and more. Make sure that the number of categories and category labels in your profile match your own dataset.

（5）Model application: After the training is completed, the trained YOLOv5 model can be applied to the video image stream transplanted to Jstson Nano or similar industrial computer for seedling leakage detection.

3.2 Optimize and Train

（1）Modify the data/training configuration xxx.yaml file: To facilitate modification, we can copy a copy of the original data/coco128.yaml, rename it to xian.yaml, and change the parameters in it to the custom training parameters, as shown in Figure 7：

```
# YOLOv5 🚀 by Ultralytics, AGPL-3.0 license
# COCO128 dataset https://www.kaggle.com/ultralytics/coco128 (first 128 images from COCO
train2017) by Ultralytics
# Example usage: python train.py --data coco128.yaml
# parent
# ├── yolov5
# └── datasets
#     └── coco128  ← downloads here (7 MB)


# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list:
[path/to/imgs1, path/to/imgs2, ..]
path: ./datasets  # dataset root dir
train: images/train  # train images (relative to 'path') 128 images
val: images/val  # val images (relative to 'path') 128 images
test:  images/test      # test images (optional)

# Classes
names:
  0: Ture
  1: False



# Download script/URL (optional)
download: https://ultralytics.com/assets/coco128.zip
```

Figure 7

（2）Modify the models model configuration xxx.yaml file, go to the models/ directory, you can see the yaml file with the model configuration, configure your own file, and simply configure 5m, 5s, and 5x, as shown in Figure 8：

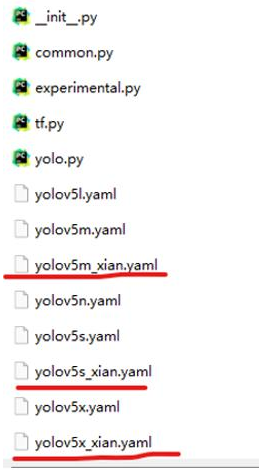


Figure 8

Take 5m_xian.yaml as an example, which is a medium-sized model in the YOLOv5 series. "m" stands for "medium". YOLOv5m offers a good balance between speed and accuracy, making it suitable for devices with some computing power. nc to 2, Figure 9:

```yaml
# YOLOv5 🚀 by Ultralytics, AGPL-3.0 license

# Parameters
nc: 2  # number of classes
depth_multiple: 0.67  # model depth multiple
width_multiple: 0.75  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]

# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Figure 9

For example, the depth_multiple in NC refers to the depth of the network, the width_multiple is the width of the network, the anchors is the anchor mark (the box that marks the object), and the backbone is the backbone network. Against

（3）Model training: Use the prepared dataset and the configured model for training.

You can start training a YOLOv5 model on your own dataset by running the training script 'train.py'. Use the following command to train:

```
python train.py --cfg models/yolov5m_tstat.yaml --data data/Tstat.yaml --weights yolov5m.pt --epoch 1200 --batch-size 32
```

The --data parameter specifies the configuration file path of the dataset, the --cfg parameter specifies the model configuration file path, and the --weights parameter specifies the pre-trained weight file path (e.g., 'yolov5s.pt'). Adjust hyperparameters during training, including:

- - lr0: 0.01 # learning rate
- - lrf: 0.13 # Cosine Annealing hyperparameter (CosineAnnealing)
- - momentum: 0.846 # learning rate momentum
- - weight_decay: 0.00036 # weight decay coefficient
- - warmup_epochs: 2.0 # warmup learning epochs
- - warmup_momentum: 0.5 # warmup learning rate momentum
- - warmup_bias_lr: 0.04 # warmup learning rate
- - box: 0.0296 # coefficient of giou loss
- - cls: 0.243 # coefficient of classification loss
- - cls_pw: 0.631 # weight of positive samples in classification BCELoss
- - obj: 0.301 # Coefficient of loss with and without objects
- - obj_pw: 0.911 # weight of positive samples in BCELoss with or without objects
- - iou_t: 0.2 # iou threshold for labeling with anchors iou training threshold

（4）Model evaluation: After the model is trained, you can see that the weights of the trained model are saved in runs/train/exp17/weights/last.pt and best.pt, which also includes training process data such as PR Curve, Confusion matric, results.png/txt, etc., as shown in Figure 10.

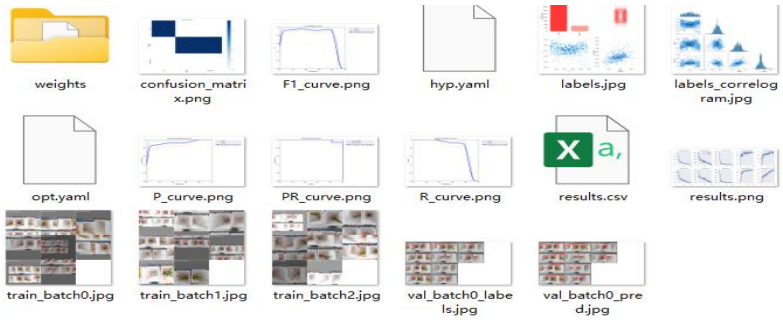


Figure 10

During training, the model can be evaluated using a validation set to understand its performance. YOLOv5 provides an evaluation script 'val.py', which can be evaluated by running the following command：

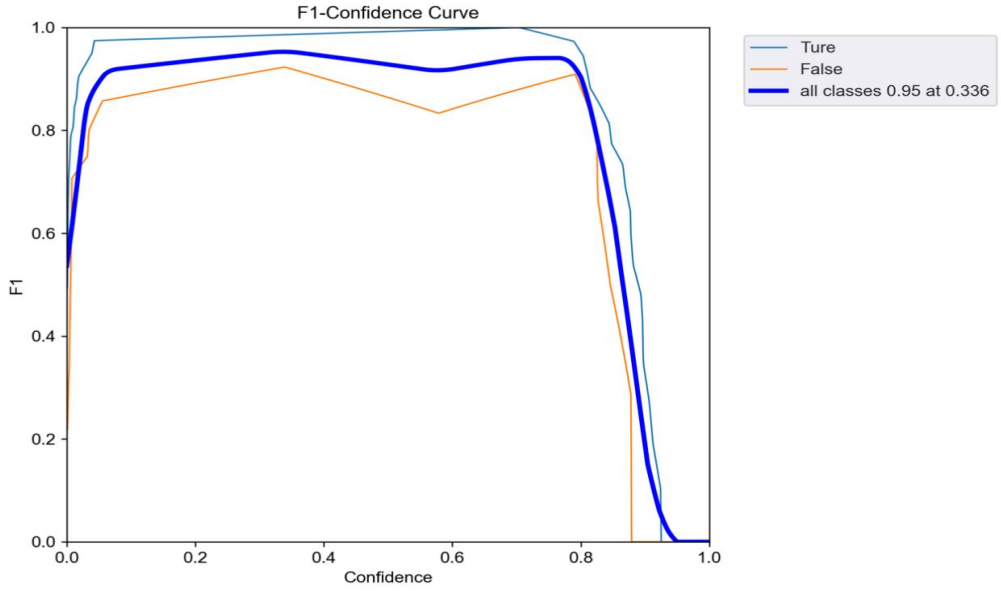
```
python val.py --img 640 --batch 16 --data path/to/your/data.yaml --weights path/to/your/weights.pt
```

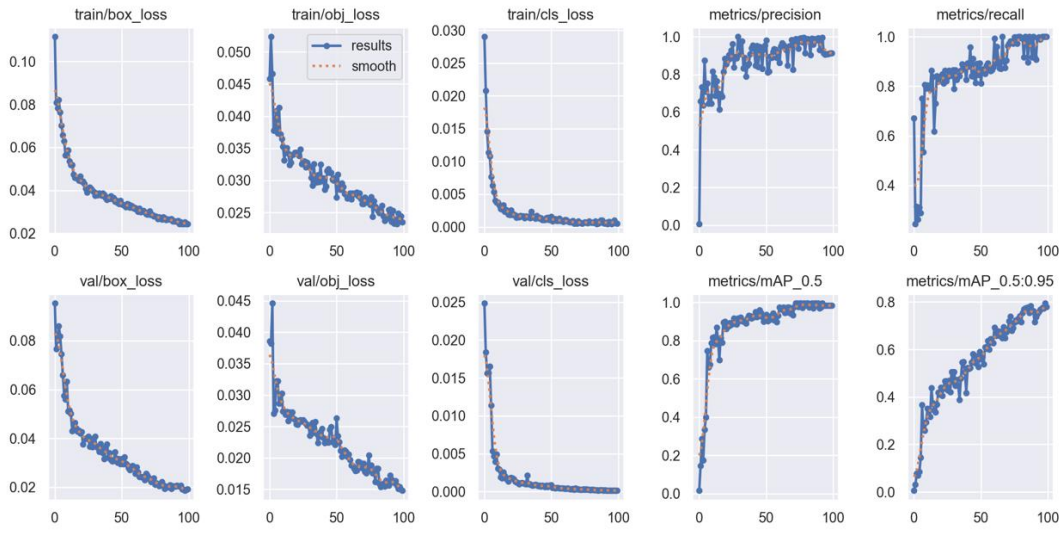
The '--weights' parameter here specifies the path of the trained model's weights file. The evaluation results will show the accuracy of the model, the recall rate, and other metrics.

3.3 Tests & Results

For non-leaking seedlings, "Ture" was used to mark and "False" was used for missing seedlings, and the training results were as follows:

[Confusion matrix REMOVED]





- mAP: [The higher, the better]

It comprehensively considers the accuracy and recall of the model in different categories, and averages all classes.

- Loss: [The lower the better, (on the premise that it doesn't fit)]

Key metrics during the training phase, representing the difference between the model's predicted value and the real label. Through the change of Loss, the fit degree of the model can be judged to a certain extent (whether it is fully trained and whether it is overfitted).

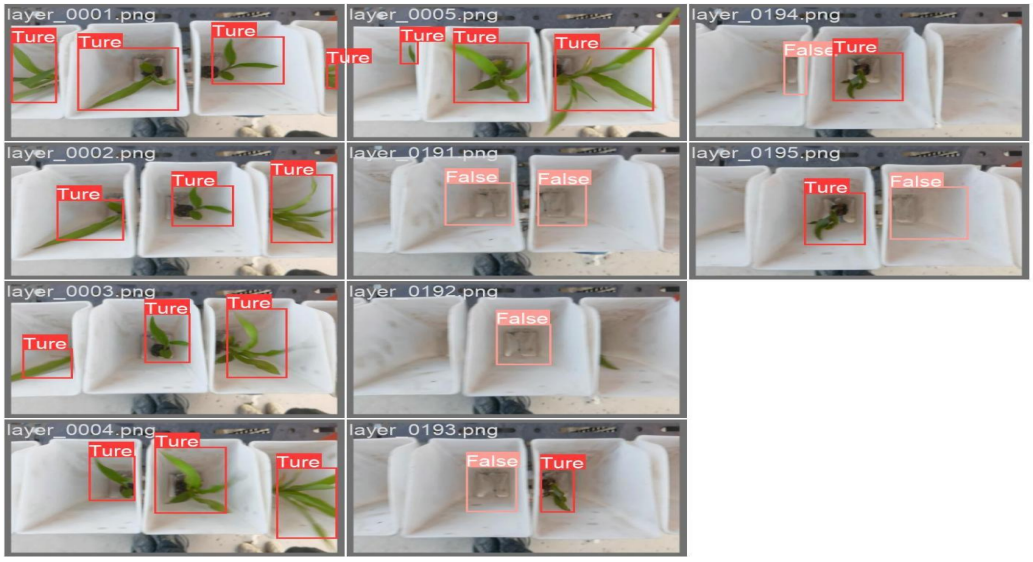
- Recall: [The higher, the better]

It reflects the model's ability to discover real targets.

- Confusion matrix: [The higher the TP and TN, the better; the lower the FP and FN, the better]

These include TP, TN, FP, FN, which can help analyze how the model performs on each category.

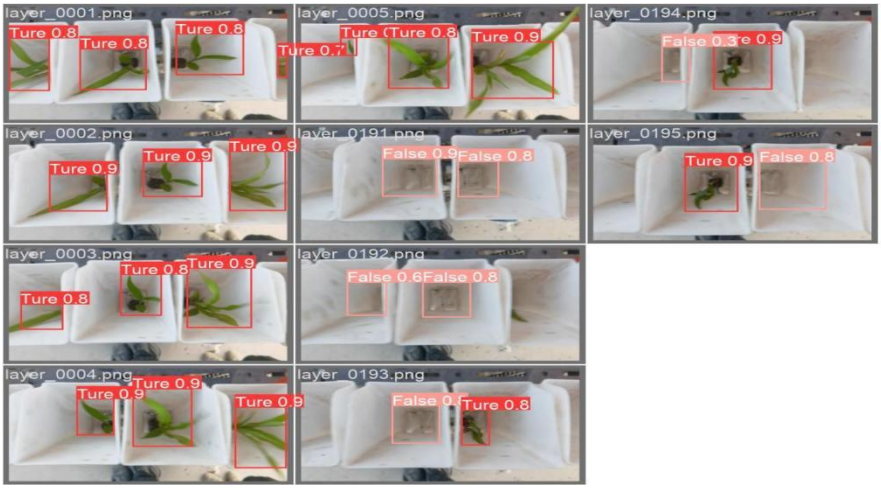
Validation Set Results:



***Here Ture represents normal seedlings, and False represents leaked seedlings.**

4 FINAL RESULT

4.1 Test Set Results:



4.2 Video detection results:

*The confidence level of the detection is above 0.6, and some even reach more than 0.9, which is very good, and the recognition rate is also maintained at a relatively high level.