Container Inspection

Computer Vision AI Bootcamp

5 September 2025

VisionX

Budi Kawira

Developer

CNR Extractor

Dana Putra Kembara

Developer

Damage Detector

Maruli Tyson

Team Member

Mustafa

Team Member

Efenus Valentino

Team Member

Background & Problem Statement

Background

The global supply chain relies on shipping containers, which must be inspected for damage to ensure they are structurally sound and safe for transit. This traditional manual process is slow, labor-intensive, and prone to human error, creating a significant bottleneck in logistics due to the rapid growth of global trade.

Problem Statement

The manual inspection of shipping containers presents several critical issues:

- **Operational Inefficiency and Cost:** The manual process is slow and a significant source of operational costs in logistics and maritime industries.
- **Inconsistency and Human Error:** Inspections vary in accuracy, leading to inconsistent results and the potential for critical defects to be overlooked.
- **Safety and Risk:** Overlooked damage can lead to cargo damage, structural failure, and safety hazards during transit.
- **Lack of Scalability:** The high volume of global trade makes a fully manual approach unsustainable, causing significant operational delays.

There is a clear need for an automated, scalable, and highly accurate solution to streamline the inspection process

Reference:
https://tcc-industry.innovation-challenge.sg/problem-statement/automated-inspection-of-shipping-containers-at-container-depots.html

Objectives & Scope

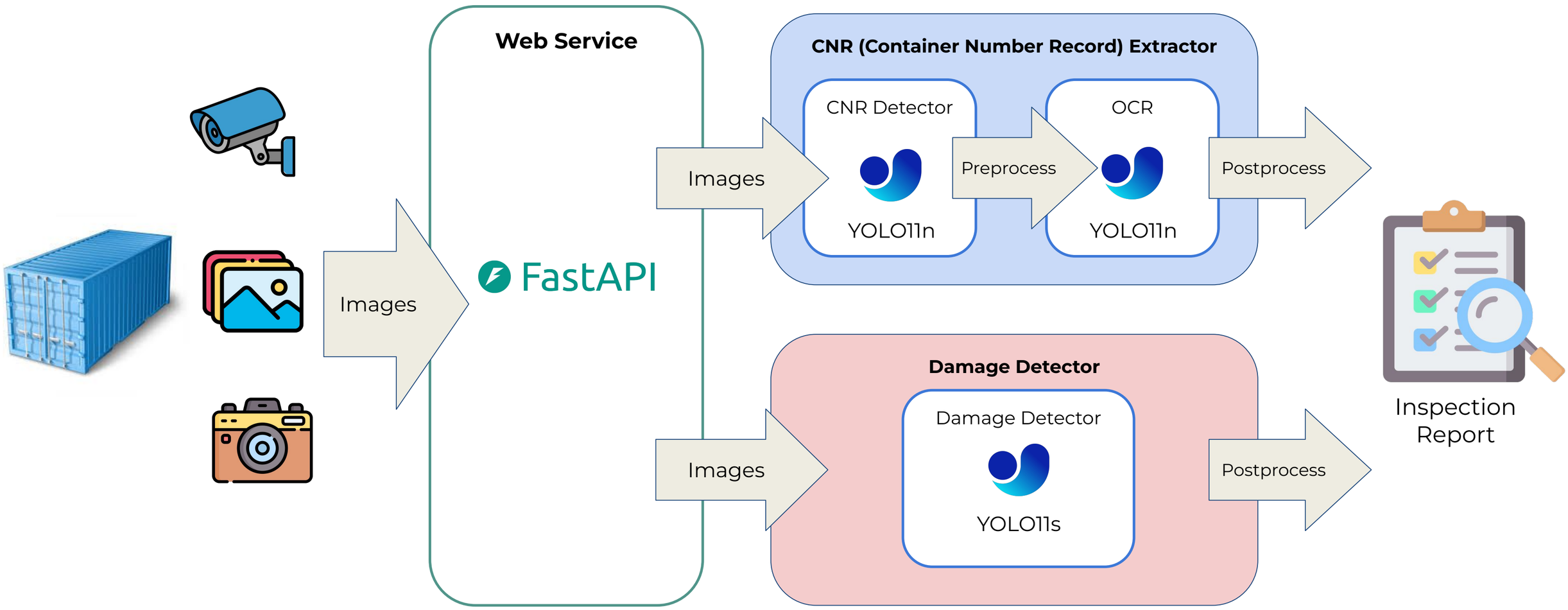
Objectives

- Develop and implement a computer vision-based AI system for automated container inspection.
- The system will accurately extract and register Container Number Records (Owner Prefix, Serial Number, and ISO6346)
- The system will also detect various types of damage, including dents, rust, and structural defects.
- The goal is to replace the slow, inconsistent human process with a fast, reliable, and scalable automated solution.

Scope

- Significant improvements in operational efficiency and cost savings for logistics companies.
- Reduced inspection times, increased throughput, and minimized labor costs.
- Enhanced accuracy and consistency from the AI, leading to a reduction in overlooked defects and improved overall safety.
- A scalable and standardized approach to container inspection capable of meeting the demands of a growing global supply chain

System Architecture



Data Collection & Preparation

CNR Detector

- Source images are from Internet Search Engine and subset of Roboflow Final Container Project

- Images are annotated using CVAT
- There are total 6 classes:
  - Owner Prefix
  - Serial Number
  - ISO6346
  - V Owner Prefix
  - V Serial Number
  - V ISO6346

OCR

- Source images are from object detected by CNR detector
- Images are cropped and preprocessed for OCR purpose
- Preprocess steps:
  - Resize (Upscale)
  - Denoising
  - Enhance contrast
  - Sharpening

- Preprocessed Images are annotated using CVAT for training Letter Detection Object Detection
- There are total 35 classes:
  - ‘0’ to ‘9’
  - ‘A’ to ‘P’
  - ‘R’ to ‘Z’

Damage Detector

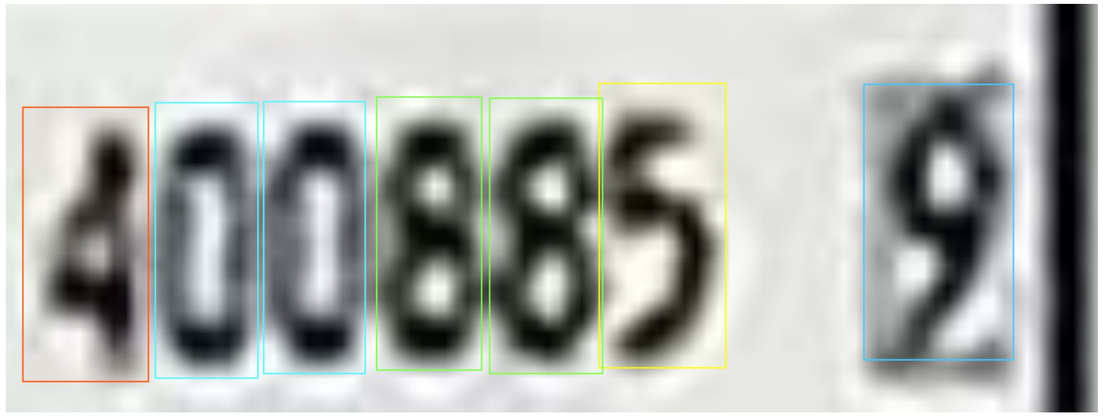
- Source images are from Roboflow Container Damage Detection
- Contains 7731 images, size 416 x 416 pixels, and 1 class [Damaged]
- Augmentations using vertical and horizontal flip, and 90 degree rotation

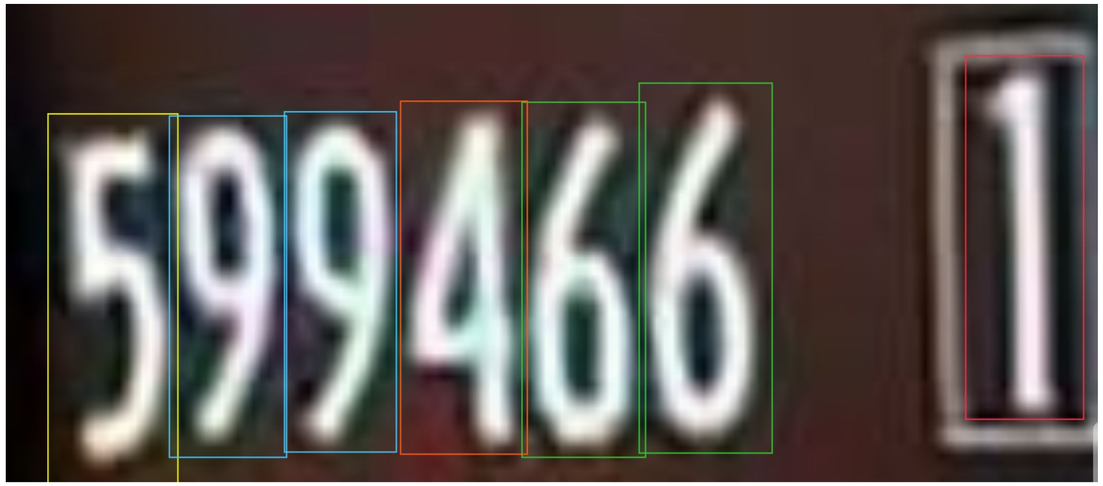
CNR Dataset

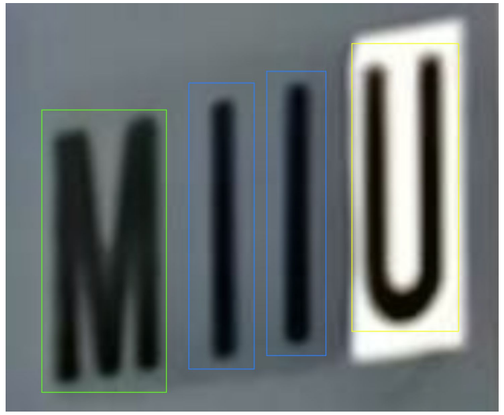
Annotation result of CNR Dataset is uploaded to Kaggle:
https://www.kaggle.com/datasets/budikawira/container-cnr

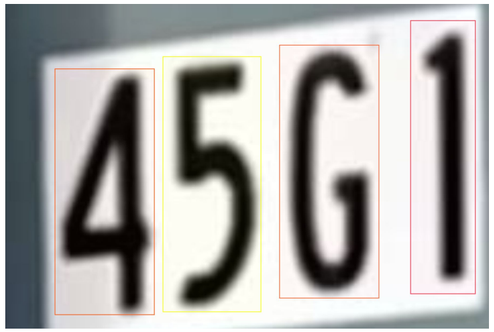
OCR Dataset

Annotation result of OCR Dataset is uploaded to Kaggle:
https://www.kaggle.com/datasets/budikawira/container-cnr-letter-detection





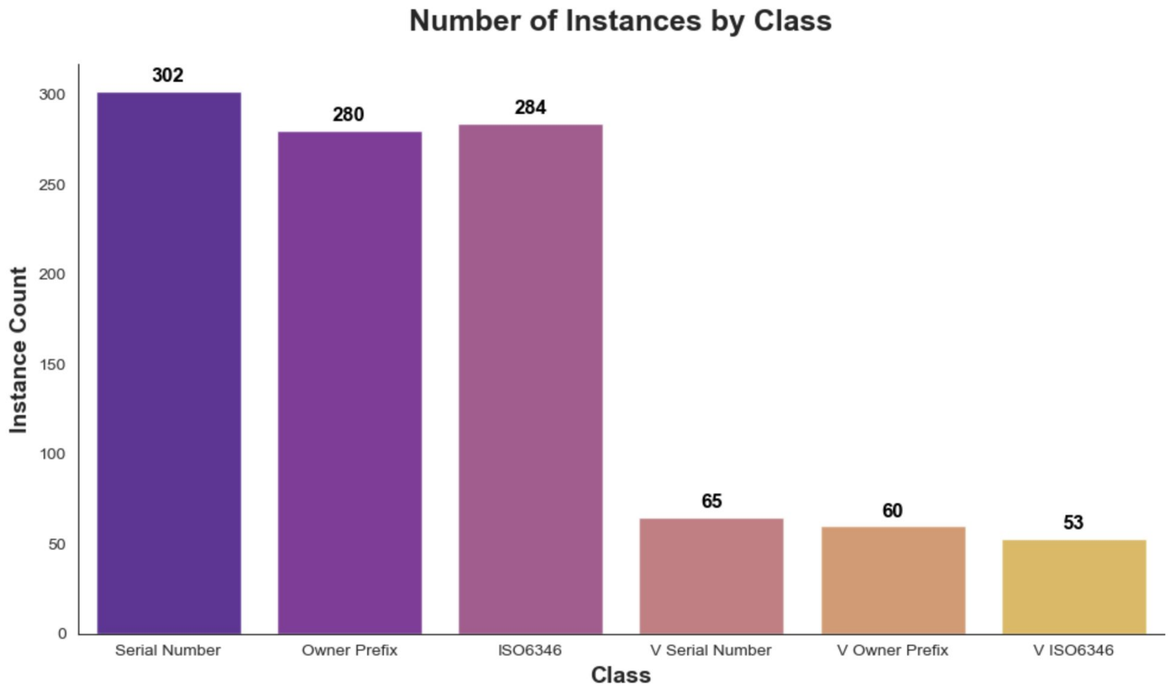




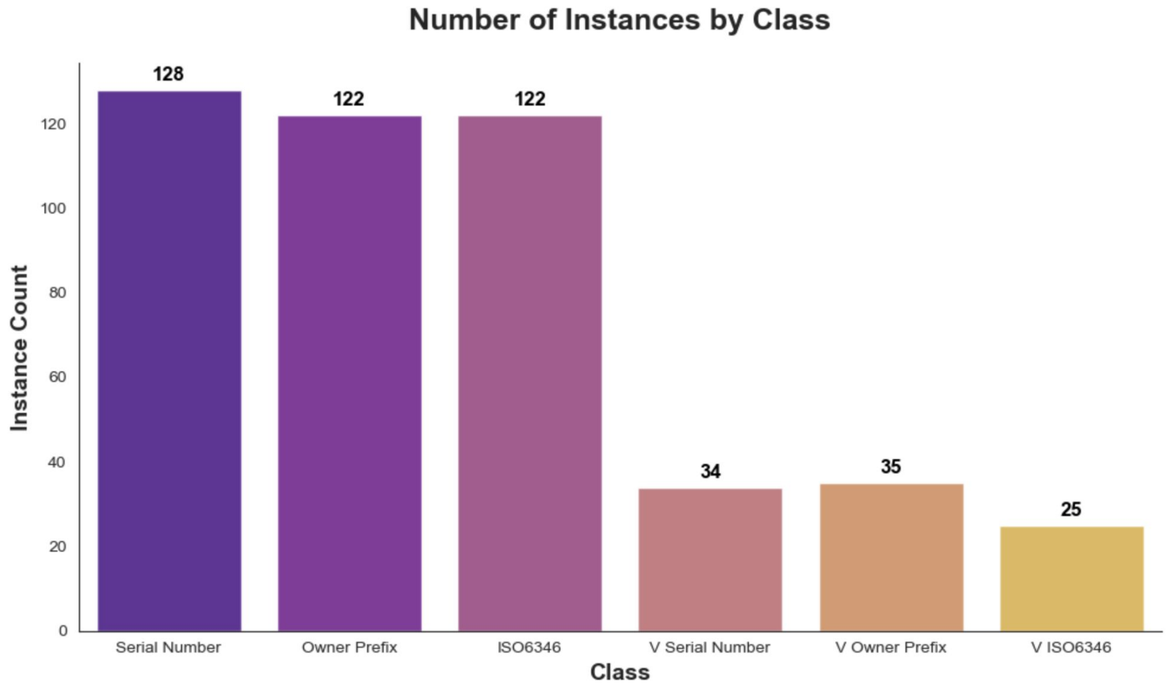
CNR Detector EDA

Total Image Files : 450
Train Image Files (70%) : 315
Validation Image Files (30%) : 135

Training Dataset



Validation Dataset

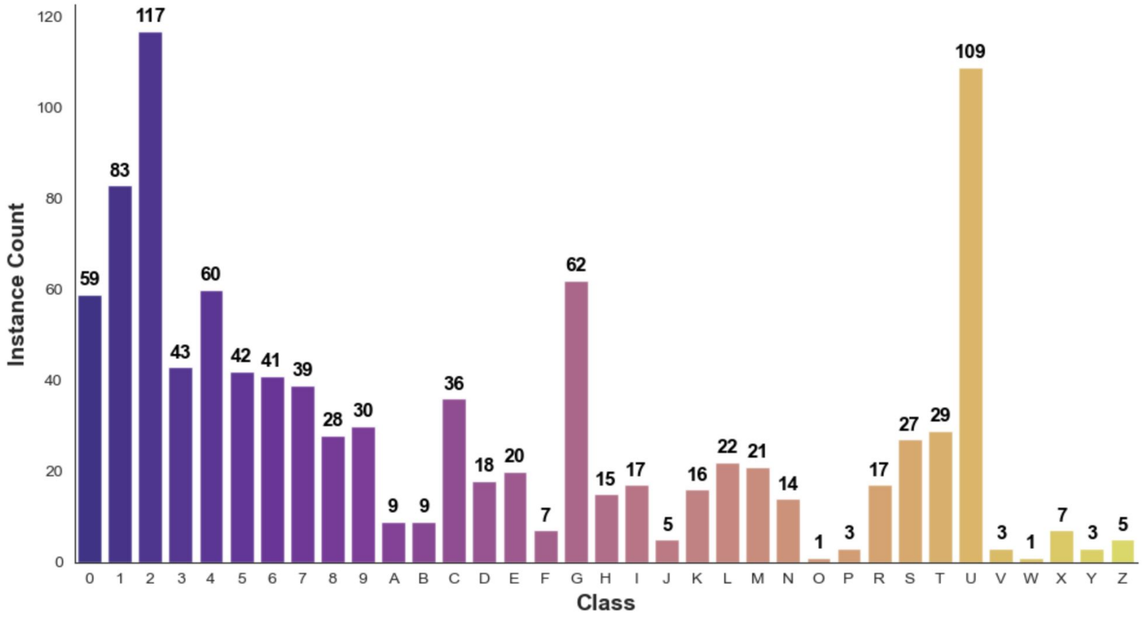


Letter Detector (OCR) EDA

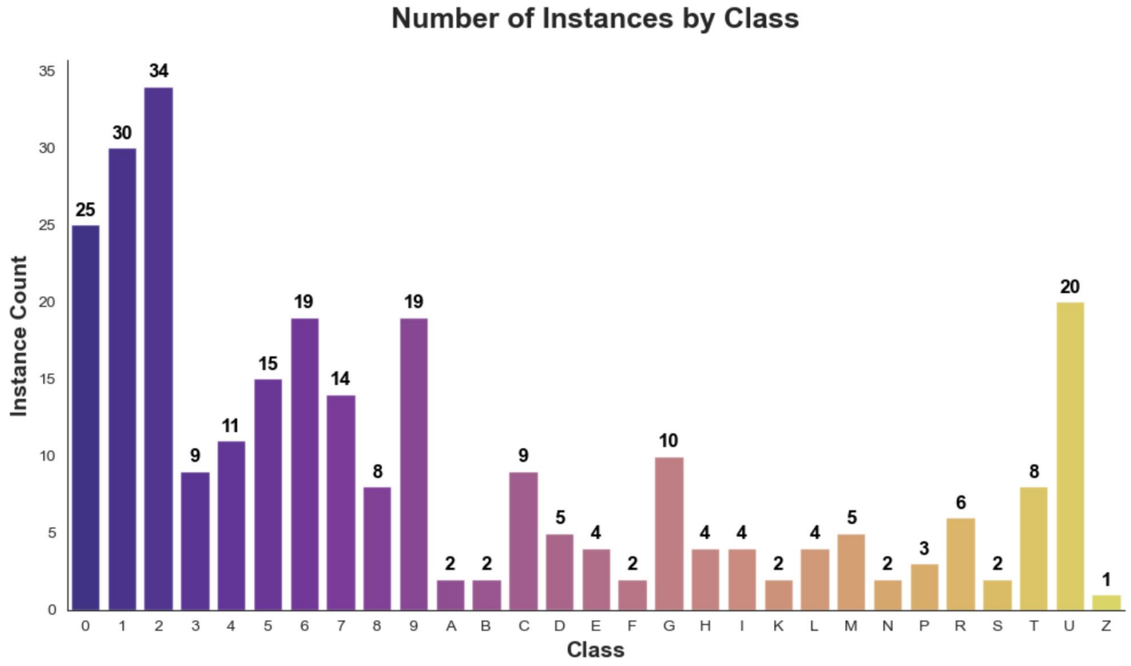
Total Image Files : 268
Train Image Files (80%) : 214
Validation Image Files (20%) : 54

Number of instance of some classes are still too small. Additional image files should be able to improve the model accuracy

Training Dataset



Validation Dataset



Training Hyperparameters (CNR Detector)

Parameters	Value
Architecture	Yolo11n
Epoch	80
Batch	16
Image Size	640x640
Optimizer	AdamW
Starting LR	0.001
Final LR After Decay	0.01
Momentum	0.9
Weight Decay	0.0005
Warmup Epoch	3
Warmup Momentum	0.8
Warmup Biar LR	0.1

Augmentation	Value
Horizontal Flip	Disabled
Vertical Flip	Disabled
Random Rotation	30
Translate	0.1
Scale	0.5
Hue Variation	0.015
Saturation Variation	0.7
Brightness Variation	0.4
Mosaic	Enabled
Copy Paste	Disabled

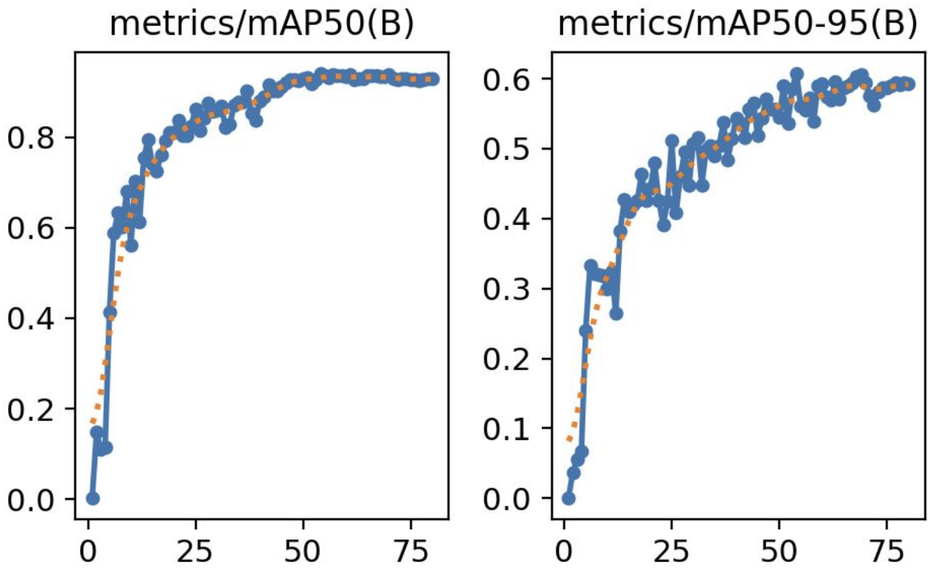
Hardware: NVIDIA GeForce RTX 4060 Laptop GPU

Training Result

CNR Object Detection

mAP50: 0.94, mAP50-95: 0.61
Best Epoch: 54 of 80

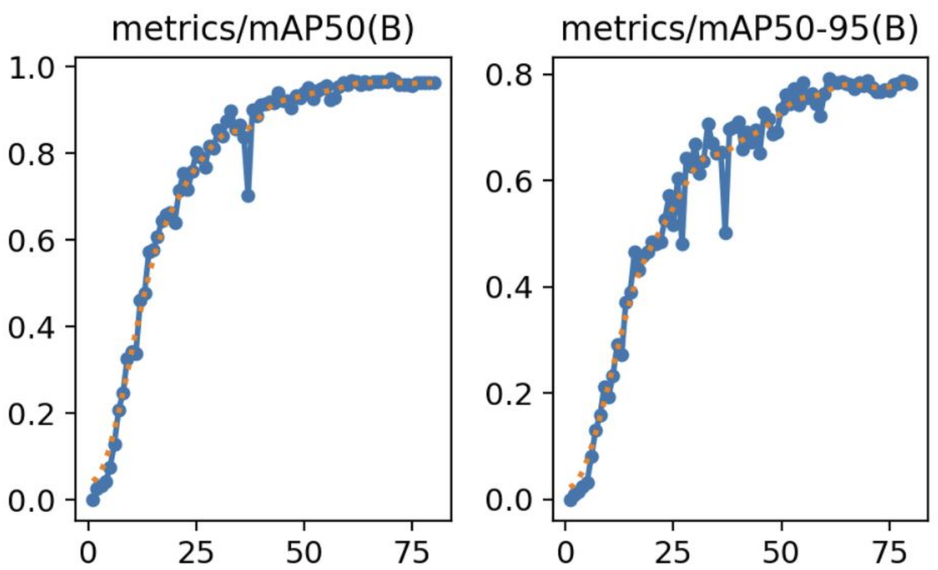
Best Model Criteria: Highest mAP50



Letter Object Detection

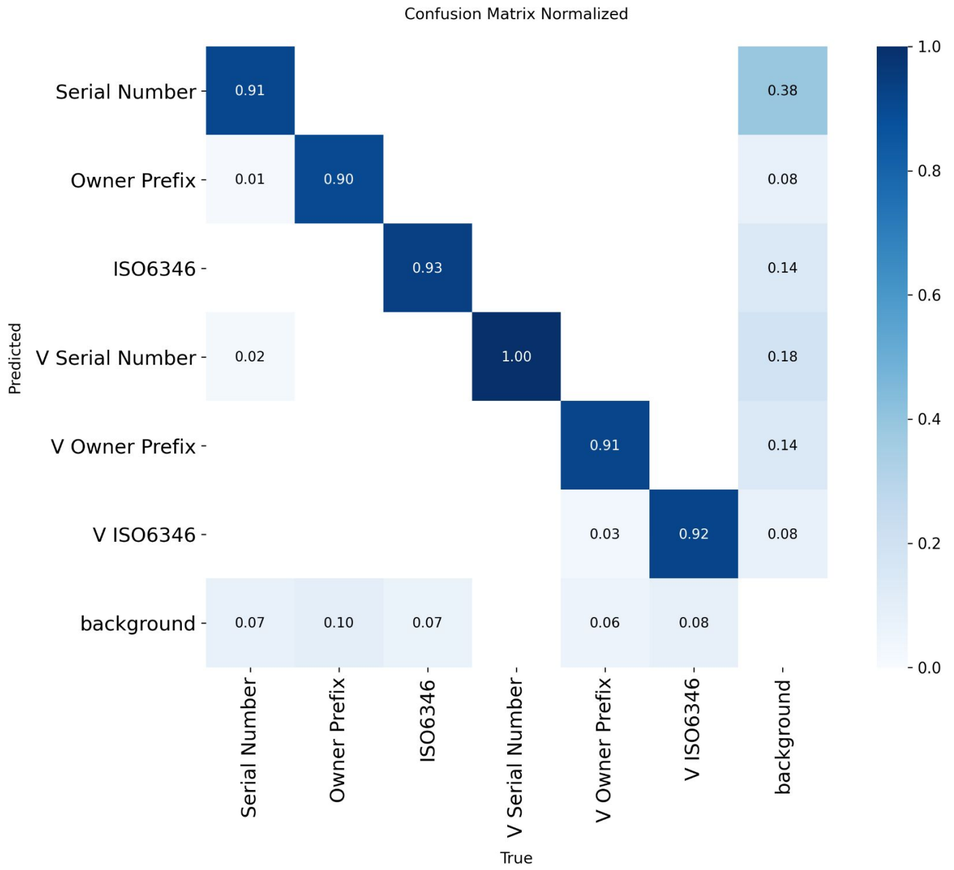
mAP50: 0.97, mAP50-95: 0.79
Best Epoch: 70 of 80

Best Model Criteria: Highest mAP50

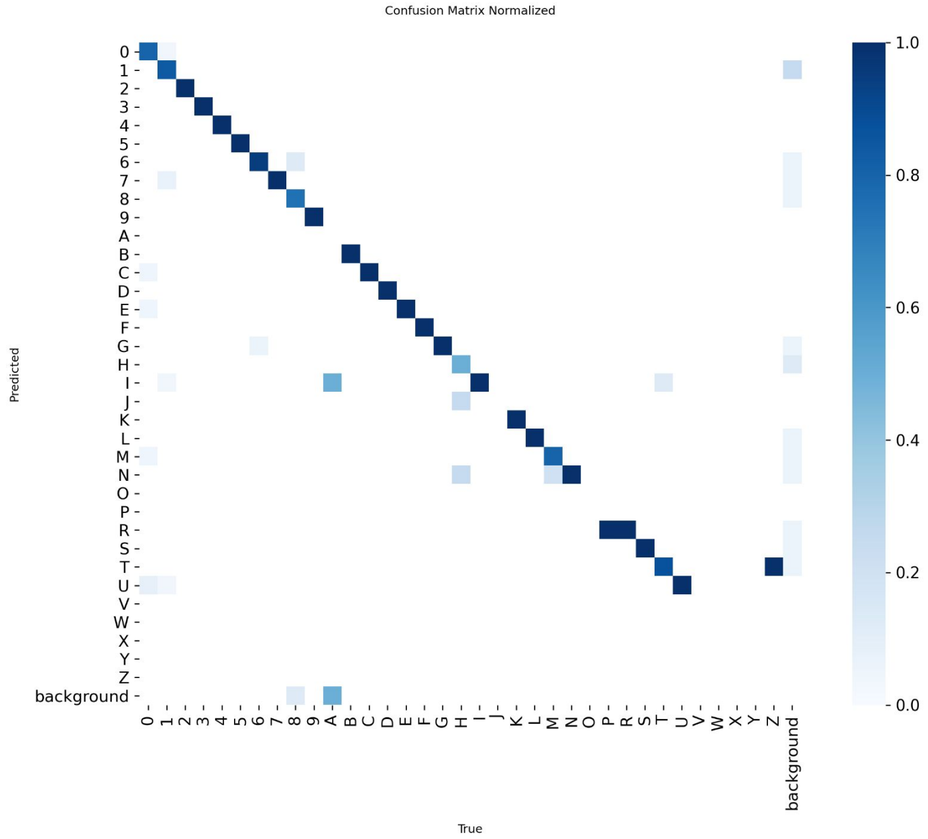


Training Result - Confusion Matrix

CNR Object Detection

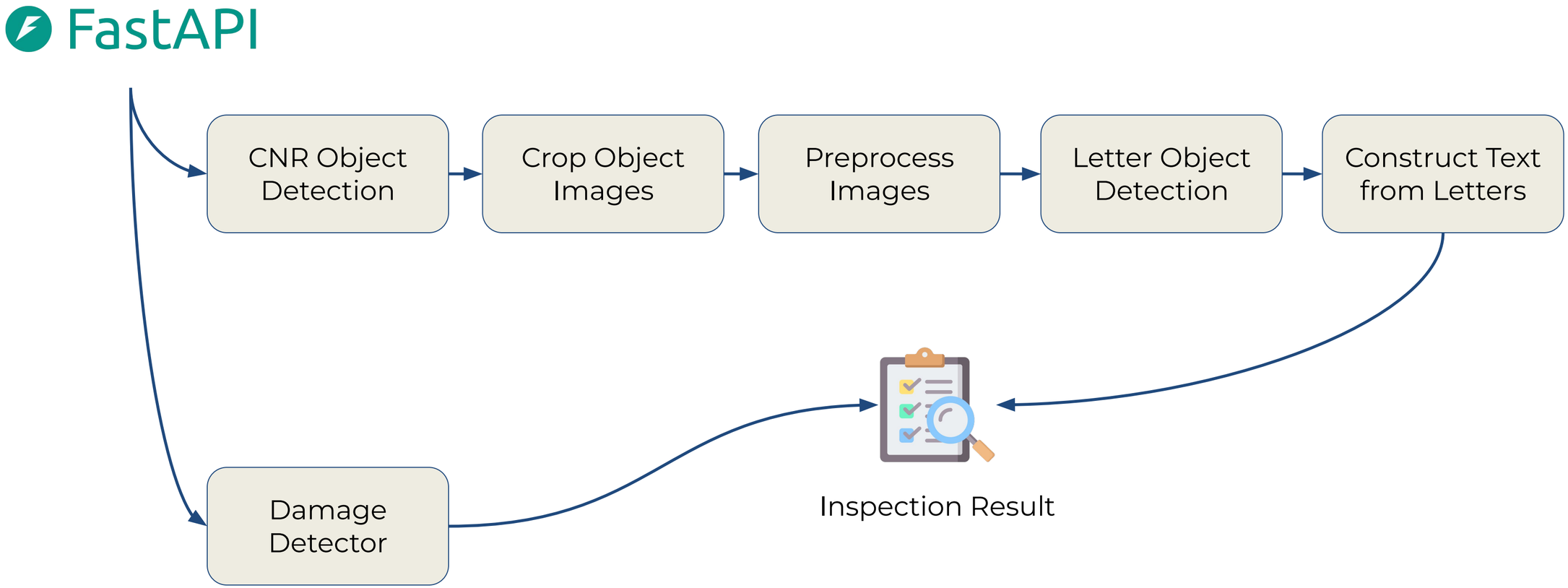


Letter Object Detection



Deployment

Container Inspection System is deployed based on FastAPI Framework



Deployment (CNR Extractor)

FastAPI

CNR Object Detection

Crop Object Images

```
#Object Detection: Detect CNR
cnr_model = app.state.models["cnr"]
results = cnr_model.predict(Image.open(image_path),conf=0.4)
result_path = os.path.join(folder, f"{label}-cnr.jpg")
results[0].save(filename=result_path)
cnr.save_yolo_predictions(results, result_path, folder)
crop_filenames = cnr.crop_predicted_objects(cnr_model, image_path, results, folder)
```

Preprocess Images

Letter Object Detection

```
#Object Detection: Detect Letters (OCR)
ocr_model = app.state.models["ocr"]
for crop_filename in crop_filenames:
    params = crop_filename.split('_')
    print(f"params : {params}")
    crop_filepath = os.path.join(folder, crop_filename)
    preproc_img = ocr.preprocess_image_for_ocr(Image.open(crop_filepath))
    ocr_results = ocr_model.predict(preproc_img,conf=0.4)
    ocr_results = ocr.remove_overlapping_detections(ocr_results, iou_threshold=0.5)
```

Construct Text from Letters

```
    #Check vertical or horizontal
    orientation = 'x'
    if (params[3].startswith("V ")):
        orientation = 'y'
    text, text_lbls, avg_conf = ocr.get_sorted_label_string(ocr_model, ocr_results, sort_by=orientation)
    print(f"letters predictions : {text_lbls}")
    utils.save_text(f"{text}\n{avg_conf}", os.path.join(folder, utils.remove_extension(crop_filename) + ".txt"))
    print(f"text : {text}")
```

Inference Results (CNR Detector)

Source Image

CNR Result

Crop Object Images

Owner Prefix, Confidence: 0.82

Serial Number, Confidence: 0.83

ISO6346, Confidence: 0.79

Inference Results (CNR Detector)

Preprocess Images

Owner Prefix

Serial Number

ISO6346

OCR

1. Letter Object Detection, Conf=0.4
2. Detect IoU, if IoU between objects are > 0.5 then remove letter object with smaller confidence
3. Convert letter to text by ‘y’ coordinate if object is vertical text (Class name start with ‘V ‘) and convert by ‘x’ coordinate if object is horizontal text

TTNU
Average Confidence: 0.87
Owner Prefix

5994661
Average Confidence: 0.88
Serial Number

42G1
Average Confidence: 0.93
ISO6346

Inference Results (CNR Detector)

Source Image

9019270
Average Confidence: 0.89
Serial Number

HMCU
Average Confidence: 0.68
Owner Prefix

45G1
Average Confidence: 0.89
ISO6346

Inference Results (CNR Detector)

Source Image

2706366
Average Confidence: 0.86
V Serial Number

TDCU
Average Confidence: 0.93
V Owner Prefix

Training on the Damage Detector

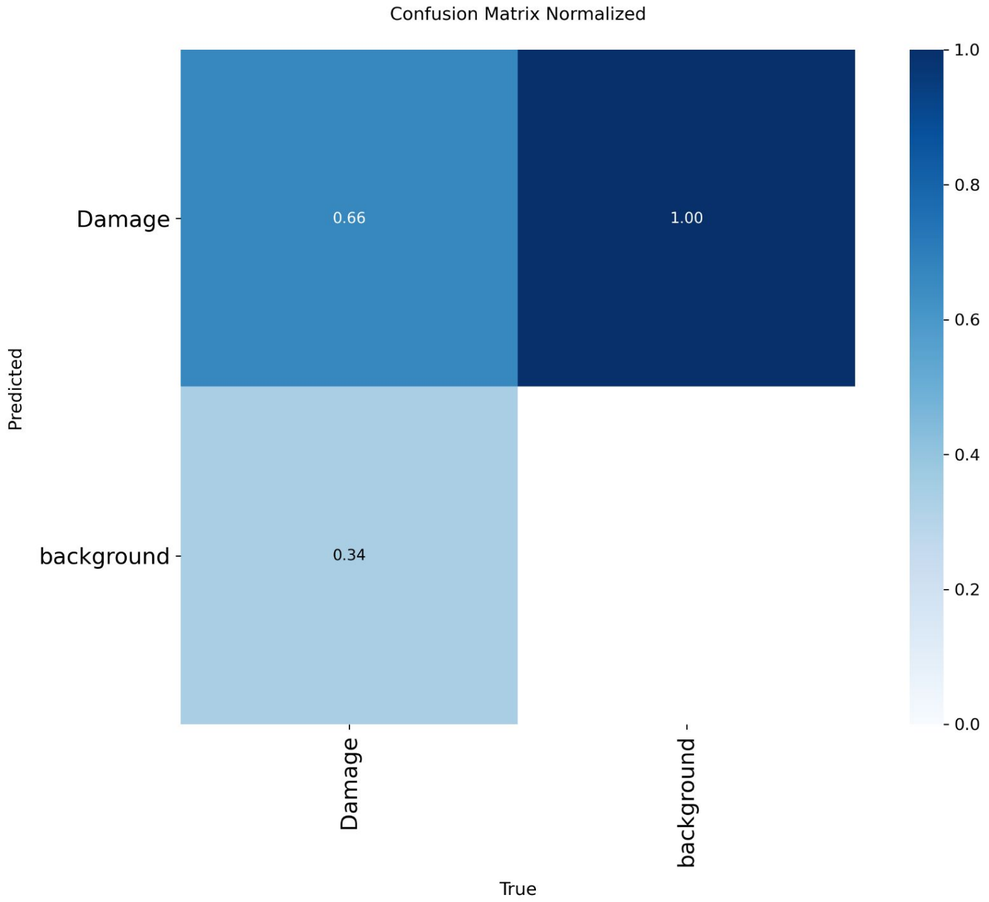
Dataset partition:

Partition	# of images	Ratio
Training	6331	82%
Validation	933	12%
Test	467	6%
Total	7731	

Training protocol:

- Pre-trained YOLO11s
- Input size = 416 x 416
- Training epochs = 100
- Post processing:
  - Grid search on the validation set for confidence threshold [0.1 - 0.95, step 0.05]
  - Grid search on the validation set for IoU threshold in NMS [0.1 - 0.95, step 0.05]
  - Best setting is achieved with confidence = 0.1 and IoU = 0.4
- Other hyperparameters use the default YOLO11s

Results on the Damage Detector



Performance on the test set:

- Precision = 55.5% ~ half of detected damages are correct
- Recall = 51.2% ~ misses many actual damages
- mAP50 = 51.9% ~ moderate localization accuracy
- Inference time = 33.81 ms/image ~ real-time capable

Conclusion

The model is fast and real-time capable, but accuracy remains moderate, which limiting reliability for full automation

Future improvement

- Expand training data (more damaged containers, diverse lighting and angles)
- Improve annotation quality and consistency

Results on the Damage Detector

Conclusion

Computer vision has emerged as a powerful tool for automating and improving the accuracy of container inspection. By leveraging cameras and image processing algorithms, it automates the detection of damages, defects, and other issues, leading to faster, more reliable, and cost-effective inspections. This technology significantly reduces the reliance on manual labor, which is prone to human error and can be slow.

Continuous improvement of accuracy can be done by continuously improving dataset and analyze the performance periodically.

Thank You