



YOLO OBJECT DETECTION

InMind Final Project

Project Documentation

This is the documentation about the implementation of an object detection system for industrial environments using YOLOv5. The project focuses on detecting objects such as tuggers, cabinets, STRs, boxes, and forklifts.

Hassan Khadra
HassanKhxd@gmail.com

Table Of Contents

Executive Summary.....	5
Introduction	6
Part 1: Data Preparation & Visualization	6
1. Loading the Dataset.....	6
a. Code Implementation: Load Dataset.....	7
2. Visualization of Labeled Images	7
a. Code Implementation: Visualization.....	8
3. Dataset Augmentation	8
a. Code Implementation: Data Augmentation	8
4. Augmentation Results.....	10
5. Dataset Splitting	10
a. Code Implementation: Dataset Splitting.....	11
Part 2: Model Training & Evaluation	12
1. YOLOv5 Model Training.....	12
a. Dataset Preparation for YOLOv5	12
b. YOLOv5 Dataset Configuration.....	13
c. Training Process.....	13
d. Model Evaluation and Hyperparameter Tuning.....	14
2. TensorBoard Visualization	16
a. Model Performance Comparison	16
b. F1 Score Curves.....	16
c. Mean Average Precision (mAP).....	18
b. Loss Comparison	18
c. Learning Rate	19
d. Confusion Matrices.....	20
Part 3: Model Deployment & Inference	22
1. Export Models to ONNX Format.....	22
2. Network Visualization with Netron	22
3. REST API Implementation with FastAPI.....	23
a. API Endpoints.....	23
b. Architecture	23
c. Sample JSON Output.....	24
d. Testing FastAPI Endpoints.....	24
4. Docker Containerization	26

a. Dockerfile.....	26
b. Docker Compose	26
c. Running the Containerized API	27
Conclusion	28
References	29
Appendix.....	30
System Requirements	30
Model Configuration Details.....	30

Table Of Figures

Figure 1: Implementation of dataset loading functionality.....	7
Figure 2: Implementation of bounding box visualization.....	8
Figure 3: Implementation of data augmentation techniques.....	9
Figure 4: Example visualized image after augmenting it.	10
Figure 5: Implementation of dataset splitting functionality	11
Figure 6: Yolov5 repository setup and initial preparation steps	12
Figure 7: Dataset configuration YAML for YOLOv5	13
Figure 8: Training command execution and initial output.....	13
Figure 9: Model architecture summary for YOLOv5s.....	14
Figure 10: Training completion output with final metrics.....	14
Figure 11: Evaluation results of the trained model on test dataset	15
Figure 12: Training with modified hyperparameters	15
Figure 13: Completion output of hyperparameter-tuned training.....	16
Figure 14: F1 score curve for the standard model.....	17
Figure 15: F1 score curve for the hyperparameter-tuned model.....	17
Figure 16: Comparison of mAP@0.5 and mAP@0.5:0.95 between both models	18
Figure 17: Box loss, class loss, and object loss during training for both models	18
Figure 18: Validation losses for both models.....	19
Figure 19: Learning rate schedules for both models	19
Figure 20: Confusion matrix for the standard model.....	20
Figure 21: Confusion matrix for the hyperparameter-tuned model.....	21
Figure 22: Netron visualization of the standard YOLOv5s model architecture	22
Figure 23: Netron visualization of the hyperparameter-tuned model architecture.....	23
Figure 24: FastAPI docs page showing project endpoints	24
Figure 25: '/models' endpoint returning the available models.....	25
Figure 26: '/Inference' Endpoint showing the json output which includes the bounding boxes.....	25
Figure 27: Image inference endpoint showing the image and the returned bounding boxes	26
Figure 28: Docker Compose build process initiation	27
Figure 29: Successful completion of the Docker container build	27
Figure 30: Container logs showing API endpoint requests and responses	28

Executive Summary

This Final Project implements an object detection system designed specifically for industrial environments. This documentation details the development of a computer vision solution capable of detecting various industrial objects including tuggers, cabinets, STRs, boxes, and forklifts. The project utilizes the YOLOv5 architecture, a state-of-the-art real-time object detection framework.

The documentation covers the complete workflow from data preparation and model training to deployment and inference implementation. Performance evaluation metrics are presented, comparing standard and hyperparameter-tuned models. The final solution includes a containerized REST API for seamless integration into industrial systems.

Introduction

Object detection in industrial environments presents unique challenges due to varying lighting conditions, occlusions, and the need for high accuracy to ensure safety and efficiency. The Project addresses these challenges by implementing a customized YOLOv5-based detection system.

This project is structured in three main parts:

1. Data Preparation & Visualization
2. Model Training & Evaluation
3. Model Deployment & Inference

Each section describes the methodologies, implementation details, and results achieved throughout the development process.

Part 1: Data Preparation & Visualization

1. Loading the Dataset

A custom dataset loader was implemented using PyTorch's DataLoader to efficiently process images and annotations for training. The loader handles batch processing and ensures proper formatting of the input data.

a. Code Implementation: Load Dataset

```
Project > Part_1 > load_dataset.py > ...
1  import os
2  import json
3  from torch.utils.data import Dataset, DataLoader
4  from PIL import Image
5
6  class BMWObjectDataset(Dataset):
7      def __init__(self, image_dir, label_dir):
8          self.image_dir = image_dir
9          self.label_dir = label_dir
10         self.image_files = sorted([f for f in os.listdir(image_dir) if f.endswith(".png")])
11
12     def __len__(self):
13         return len(self.image_files)
14
15     def __getitem__(self, idx):
16         image_path = os.path.join(self.image_dir, self.image_files[idx])
17         label_path = os.path.join(self.label_dir, self.image_files[idx].replace(".png", ".json"))
18
19         image = Image.open(image_path).convert("RGB")
20         with open(label_path, "r") as f:
21             labels = json.load(f)
22
23         return image, labels
24
25 image_dir = "../dataset/data/images"
26 label_dir = "../dataset/data/labels/json"
27 dataset = BMWObjectDataset(image_dir, label_dir)
28 dataloader = DataLoader(dataset, batch_size=4, shuffle=True)
```

Figure 1: Implementation of dataset loading functionality

2. Visualization of Labeled Images

To verify the correctness of annotations, a visualization function was developed. This function overlays bounding boxes on original images, allowing for visual inspection of the labeled data.

a. Code Implementation: Visualization

```
Project > Part_1 > visualize.py > ...
1  import random
2  import matplotlib.pyplot as plt
3  import matplotlib.patches as patches
4  from load_dataset import dataset
5
6  def visualize_sample(image, labels):
7      fig, ax = plt.subplots(1)
8      ax.imshow(image)
9
10     for obj in labels:
11         rect = patches.Rectangle(
12             (obj["Left"], obj["Top"]),
13             obj["Right"] - obj["Left"],
14             obj["Bottom"] - obj["Top"],
15             linewidth=2, edgecolor='r', facecolor='none')
16         ax.add_patch(rect)
17
18     plt.show()
19
20 random_indices = random.sample(range(len(dataset)), 2)
21
22 sample_image_1, sample_labels_1 = dataset[random_indices[0]]
23 visualize_sample(sample_image_1, sample_labels_1)
24
25 sample_image_2, sample_labels_2 = dataset[random_indices[1]]
26 visualize_sample(sample_image_2, sample_labels_2)
```

Figure 2: Implementation of bounding box visualization

3. Dataset Augmentation

Data augmentation techniques were applied to improve model generalization and robustness. The Albumentations library was used to implement various transformations including horizontal flips, rotations, and contrast adjustments.

a. Code Implementation: Data Augmentation

```

Project > Part_1 > augment_dataset.py > augment_dataset
1 import albumentations as A
2 from load_dataset import BMWObjectDataset
3 import os
4 import json
5 from PIL import Image
6 import numpy as np
7
8 def augment_dataset(image_dir, label_dir, output_dir):
9     dataset = BMWObjectDataset(image_dir, label_dir)
10
11     transformations = [
12         ("horizontal_flip", A.Compose([A.HorizontalFlip(p=1.0)],
13                                     bbox_params=A.BboxParams(format='pascal_voc', label_fields=['category_ids']))),
14         ("rotate", A.Compose([A.Rotate(limit=30, p=1.0)],
15                             bbox_params=A.BboxParams(format='pascal_voc', label_fields=['category_ids']))),
16         ("brightness", A.Compose([A.RandomBrightnessContrast(p=1.0)],
17                                 bbox_params=A.BboxParams(format='pascal_voc', label_fields=['category_ids'])))
18     ]
19
20     output_image_dir = os.path.join(output_dir, "images")
21     output_label_dir = os.path.join(output_dir, "labels", "json")
22
23     os.makedirs(output_image_dir, exist_ok=True)
24     os.makedirs(output_label_dir, exist_ok=True)
25
26     image_count = 0
27
28     for _, (image, labels) in enumerate(dataset):
29         orig_image_path = os.path.join(output_image_dir, f"{image_count}.png")
30         image.save(orig_image_path)
31
32         orig_label_path = os.path.join(output_label_dir, f"{image_count}.json")
33         with open(orig_label_path, "w") as f:
34             json.dump(labels, f)
35
36         image_count += 1
37
38         image_array = np.array(image)
39         bboxes = [[label["Left"], label["Top"], label["Right"], label["Bottom"]] for label in labels]
40         category_ids = [label["ObjectClassId"] for label in labels]
41
42         for _, (transform) in enumerate(transformations):
43             augmented = transform(image=image_array, bboxes=bboxes, category_ids=category_ids)
44
45             aug_image_path = os.path.join(output_image_dir, f"{image_count}.png")
46             Image.fromarray(augmented["image"]).save(aug_image_path)
47
48             aug_labels = []
49             for bbox, class_id in zip(augmented["bboxes"], augmented["category_ids"]):
50                 original_label = next(label for label in labels if label["ObjectClassId"] == class_id)
51
52                 aug_label = original_label.copy()
53                 aug_label["Left"] = int(bbox[0])
54                 aug_label["Top"] = int(bbox[1])
55                 aug_label["Right"] = int(bbox[2])
56                 aug_label["Bottom"] = int(bbox[3])
57                 aug_labels.append(aug_label)
58
59             aug_label_path = os.path.join(output_label_dir, f"{image_count}.json")
60             with open(aug_label_path, "w") as f:
61                 json.dump(aug_labels, f)
62
63             image_count += 1
64
65 if __name__ == "__main__":
66     image_dir = "../dataset/data/images"
67     label_dir = "../dataset/data/labels/json"
68     output_dir = "../dataset/aug_data"
69
70     augment_dataset(image_dir, label_dir, output_dir)

```

Figure 3: Implementation of data augmentation techniques

4. Augmentation Results

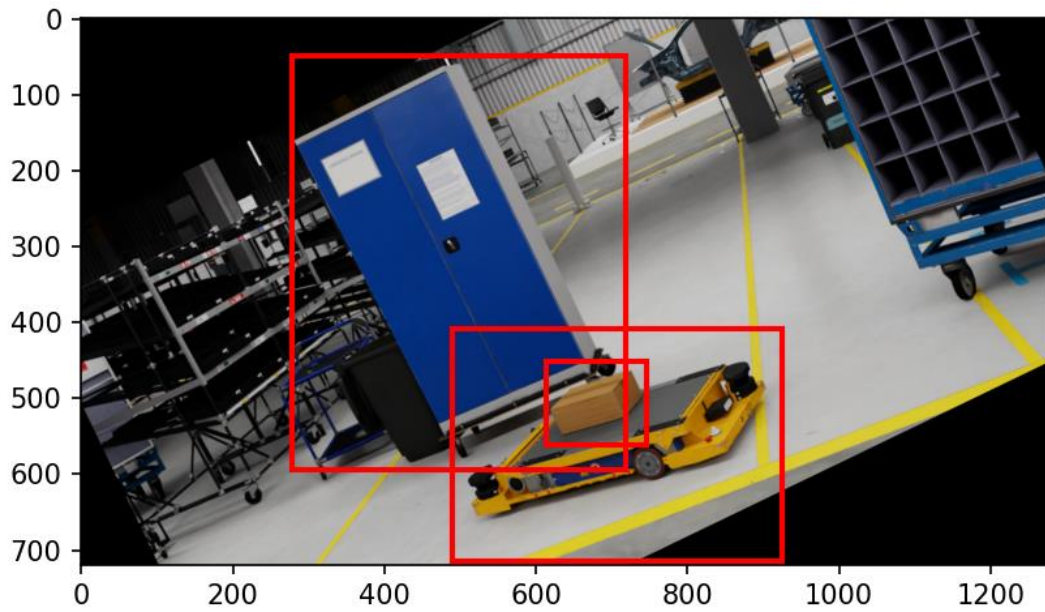


Figure 4: Example visualized image after augmenting it.

5. Dataset Splitting

The dataset was divided into training and validation sets using an 80/20 split ratio. This ensures proper evaluation of model performance on unseen data.

a. Code Implementation: Dataset Splitting

```
Project > Part_1 > split_dataset.py > split_dataset
1 import os
2 import random
3 import shutil
4 from load_dataset import BMWObjectDataset
5
6 def split_dataset(data_dir, output_dir, train_ratio=0.8):
7     image_dir = os.path.join(data_dir, "images")
8     label_dir = os.path.join(data_dir, "labels", "json")
9
10    dataset = BMWObjectDataset(image_dir, label_dir)
11
12    train_dir = os.path.join(output_dir, "train")
13    val_dir = os.path.join(output_dir, "val")
14
15    os.makedirs(os.path.join(train_dir, "images"), exist_ok=True)
16    os.makedirs(os.path.join(train_dir, "labels", "json"), exist_ok=True)
17    os.makedirs(os.path.join(val_dir, "images"), exist_ok=True)
18    os.makedirs(os.path.join(val_dir, "labels", "json"), exist_ok=True)
19
20    indices = list(range(len(dataset)))
21    random.shuffle(indices)
22    split_idx = int(len(indices) * train_ratio)
23    train_indices = indices[:split_idx]
24    val_indices = indices[split_idx:]
25
26    for idx in train_indices:
27        src_img = os.path.join(image_dir, dataset.image_files[idx])
28        dst_img = os.path.join(train_dir, "images", dataset.image_files[idx])
29        shutil.copy(src_img, dst_img)
30
31        label_file = dataset.image_files[idx].replace(".png", ".json")
32        src_label = os.path.join(label_dir, label_file)
33        dst_label = os.path.join(train_dir, "labels", "json", label_file)
34        shutil.copy(src_label, dst_label)
35
36    for idx in val_indices:
37        src_img = os.path.join(image_dir, dataset.image_files[idx])
38        dst_img = os.path.join(val_dir, "images", dataset.image_files[idx])
39        shutil.copy(src_img, dst_img)
40
41        label_file = dataset.image_files[idx].replace(".png", ".json")
42        src_label = os.path.join(label_dir, label_file)
43        dst_label = os.path.join(val_dir, "labels", "json", label_file)
44        shutil.copy(src_label, dst_label)
45
46    print(f"Split: {len(train_indices)} training, {len(val_indices)} validation samples")
47
48 if __name__ == "__main__":
49     split_dataset("../aug_data", "../split_data", 0.8)
```

Figure 5: Implementation of dataset splitting functionality

Part 2: Model Training & Evaluation

1. YOLOv5 Model Training

The YOLOv5 architecture was selected for its balance of accuracy and inference speed. The model was trained using transfer learning from pre-trained weights.

a. Dataset Preparation for YOLOv5

YOLOv5 requires a specific format for annotations, where each image has a corresponding text file containing normalized bounding box coordinates and class IDs. The dataset structure follows the YOLOv5 convention with separate directories for training and validation images and labels.

```
git clone https://github.com/ultralytics/yolov5
cd yolov5
pip install --no-deps -r requirements.txt
```



Dataset Preparation

YOLOv5 requires annotations in a specific format. Each image has a corresponding .txt file with annotations in the format:

```
<class_id> <x_center> <y_center> <width> <height>
```



My dataset was organized in the following structure:

```
/dataset
  /images
    /train
      image1.jpg
      image2.jpg
      ...
    /val
      val_image1.jpg
      val_image2.jpg
      ...
  /labels
    /train
      image1.txt
      image2.txt
      ...
    /val
      val_image1.txt
      val_image2.txt
      ...
```



Figure 6: YOLOv5 repository setup and initial preparation steps

b. YOLOv5 Dataset Configuration

Figure 7: Dataset configuration YAML for YOLOv5

c. Training Process

The model was trained using the YOLOv5 training script with the following parameters:

- Image size: 640×640 pixels
- Batch size: 16
- Number of epochs: 50
- Weights: YOLOv5s pre-trained model

```
PS C:\Users\hassa\OneDrive\Documents\GitHub\InMind-Project\Part_2\yolov5> python train.py --img 640 --batch 16 --epochs 50 --data ../dataset.yaml
--weights yolov5s.pt --cache --device 0
train: weights=yolov5s.pt, cfg=../dataset.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=50, batch_size=16, imgsz=640, rect=False, resume=False, nosave=False, noval=False, noautoanchor=False, noplots=False, evolve=None, evolve_population=data/hyps, resume_evolve=None, bucket=, cache=ram, image_weights=False, device=0, multi_scale=False, single_cls=False, optimizer=SGD, sync_bn=False, workers=8, project=runs/train, name=exp, exist_ok=False, quad=False, cos_lr=False, label_smoothing=0.0, patience=100, freeze=[0], save_period=-1, seed=0, local_rank=-1, entity=None, upload_dataset=False, bbox_interval=-1, artifact_alias=latest, ndjson_console=False, ndjson_file=False
github: up to date with https://github.com/ultralytics/yolov5
YOLOv5 v7.0-411-gf4d8a84c Python-3.9.21 torch-2.1.2+cu118 CUDA:0 (NVIDIA GeForce RTX 3070 Laptop GPU, 8192MiB)

hyperparameters: lr=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=2.0, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.0, copy_paste=0.0
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5 runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Downloading https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt to yolov5s.pt...
100% |██████████████████████████████████████████████████████████████████████████████| 14.1M/14.1M [00:12<00:00, 1.20MB/s]

Overriding model.yaml nc=80 with nc=5

      from n    params module                        arguments
   0         -1  1     3520 models.common.Conv                [3, 32, 6, 2, 2]
   1         -1  1    18560 models.common.Conv                [32, 64, 3, 2]
   2         -1  1     18816 models.common.C3                  [64, 64, 1]
   3         -1  1    73984 models.common.Conv                [64, 128, 3, 2]
   4         -1  2    115712 models.common.C3                  [128, 128, 2]
   5         -1  1   295424 models.common.Conv                [128, 256, 3, 2]
   6         -1  3   625152 models.common.C3                  [256, 256, 3]
   7         -1  1  1180672 models.common.Conv                [256, 512, 3, 2]
   8         -1  1  1182720 models.common.C3                  [512, 512, 1]
   9         -1  1   656896 models.common.SPPF                 [512, 512, 5]
  10         -1  1  131584 models.common.Conv                [512, 256, 1, 1]
  11         -1  1           0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
  12          [-1, 6]  1           0 models.common.Concat                    [1]
  13         -1  1   361984 models.common.C3                  [512, 256, 1, False]
  14         -1  1   33024 models.common.Conv                [256, 128, 1, 1]
  15         -1  1           0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
  16          [-1, 4]  1           0 models.common.Concat                    [1]
  17         -1  1    90880 models.common.C3                  [256, 128, 1, False]
  18         -1  1   147712 models.common.Conv                [128, 128, 3, 2]
  19          [-1, 14]  1           0 models.common.Concat                    [1]
```

Figure 8: Training command execution and initial output

Model architecture:

```
19      [-1, 14] 1      0 models.common.Concat      [1]
20      -1 1      296448 models.common.C3      [256, 256, 1, False]
21      -1 1      590336 models.common.Conv      [256, 256, 3, 2]
22      [-1, 10] 1      0 models.common.Concat      [1]
23      -1 1      1182720 models.common.C3      [512, 512, 1, False]
24      [17, 20, 23] 1      26970 models.yolo.Detect      [5, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]
Model summary: 214 layers, 7033114 parameters, 7033114 gradients, 16.0 GFLOPs

Transferred 343/349 items from yolov5s.pt
AMP: checks passed
optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 60 weight(decay=0.0005), 60 bias
albumentations: 1 validation error for InitSchema
size
Field required [type=missing, input_value={'scale': (0.8, 1.0), 'ra... 'mask_interpolation': 0}, input_type=dict]
For further information visit https://errors.pydantic.dev/2.10/v/missing
train: Scanning C:\Users\hasa\OneDrive\Documents\GitHub\InMind-Project\Project\dataset\yolo_data\labels\train... 80 images, 0 backgrounds, 0 corrupt
train: New cache created: C:\Users\hasa\OneDrive\Documents\GitHub\InMind-Project\Project\dataset\yolo_data\labels\train.cache
train: Caching images (0.1GB ram): 100%|██████████| 80/80 [00:00<00:00, 358.75it/s]
val: Scanning C:\Users\hasa\OneDrive\Documents\GitHub\InMind-Project\Project\dataset\yolo_data\labels\val... 20 images, 0 backgrounds, 0 corrupt: 100%|██████████| 20/20 [00:13<00:00, 1.44it/s]
val: New cache created: C:\Users\hasa\OneDrive\Documents\GitHub\InMind-Project\Project\dataset\yolo_data\labels\val.cache
val: Caching images (0.0GB ram): 100%|██████████| 20/20 [00:00<00:00, 312.50it/s]
```

Figure 9: Model architecture summary for YOLOv5s

Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size
49/49	3.61G	0.03534	0.02326	0.01079	102	640: 100% ██████████ 5/5 [00:00<00:00, 6.10it/s]
	Class	Images	Instances	P	R	mAP50 mAP50-95: 100% ██████████ 1/1 [00:00<00:00, 3.07it/s]
	all	20	47	0.961	0.817	0.913 0.638

50 epochs completed in 0.023 hours.
Optimizer stripped from runs\train\exp2\weights\last.pt, 14.4MB
Optimizer stripped from runs\train\exp2\weights\best.pt, 14.4MB

Validating runs\train\exp2\weights\best.pt...
Fusing layers...
Model summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs

Class	Images	Instances	P	R	mAP50	mAP50-95: 100% ██████████ 1/1 [00:00<00:00, 2.82it/s]
all	20	47	0.961	0.817	0.907	0.654
tugger	20	4	1	0.6	0.856	0.626
cabinet	20	13	0.973	1	0.995	0.779
str	20	13	0.998	1	0.995	0.641
box	20	13	1	0.985	0.995	0.718
forklift	20	4	0.833	0.5	0.692	0.507

Results saved to runs\train\exp2

PS C:\Users\hasa\OneDrive\Documents\GitHub\InMind-Project\Project\Part_2\yolov5>

Figure 10: Training completion output with final metrics

d. Model Evaluation and Hyperparameter Tuning

After initial training, the model was evaluated on the test dataset to assess its performance. Based on these results, hyperparameter tuning was performed to potentially improve detection accuracy.


```

PS C:\Users\hassa\OneDrive\Documents\GitHub\InMind-Project\Project\Part_2\yolov5> python val.py --weights ./runs/train/exp2/weights/best.pt --data ../../dataset/yolo_data/dataset.yaml --img 640 --batch-size 32 --device 0
val: data=../../dataset/yolo_data/dataset.yaml, weights=['./runs/train/exp2/weights/best.pt'], batch_size=32, imgsz=640, conf_thres=0.001, iou_thres=0.6, max_det=300, task=val, device=0, workers=8, single_cls=False, augment=False, verbose=False, save_txt=False, save_hybrid=False, save_conf=False, save_json=False, project=runs\val, name=exp, exist_ok=False, half=False, dnn=False
YOLOv5 v7.0-411-gf4d8a84c Python-3.9.21 torch-2.1.2+cu118 CUDA:0 (NVIDIA GeForce RTX 3070 Laptop GPU, 8192MiB)

Fusing layers...
Model summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
val: Scanning C:\Users\hassa\OneDrive\Documents\GitHub\InMind-Project\Project\dataset\yolo_data\labels\val.cache... 20 images, 0 backgrounds, 0 corrupt: 100%|██████████|

```

Class	Images	Instances	P	R	mAP50	mAP50-95: 100%	1/1 [00:07<00:00, 7.46s/it]
all	20	47	0.961	0.817	0.907	0.654	
tugger	20	4	1	0.6	0.856	0.626	
cabinet	20	13	0.973	1	0.995	0.779	
str	20	13	0.998	1	0.995	0.641	
box	20	13	1	0.985	0.995	0.718	
forklift	20	4	0.832	0.5	0.692	0.507	

```

Speed: 0.1ms pre-process, 9.3ms inference, 6.7ms NMS per image at shape (32, 3, 640, 640)
Results saved to runs\val\exp3
PS C:\Users\hassa\OneDrive\Documents\GitHub\InMind-Project\Project\Part_2\yolov5>

```

Figure 11: Evaluation results of the trained model on test dataset

A second model was trained with modified hyperparameters using the hyp.scratch-high.yaml configuration, which includes higher augmentation settings.

```

PS C:\Users\hassa\OneDrive\Documents\GitHub\InMind-Project\Project\Part_2\yolov5> python train.py --img 640 --batch 16 --epochs 100 --data ../../dataset/yolo_data/hts yolov5s.pt --cache --device 0 --hyp data/hyps/hyp.scratch-high.yaml --project runs/train --name hyper_high_train
train: weights=yolov5s.pt, cfg=, data=../../dataset/yolo_data/dataset.yaml, hyp=data/hyps/hyp.scratch-high.yaml, epochs=100, batch_size=16, imgsz=640, rect=False, noval=False, noautoanchor=False, noplots=False, evolve=None, evolve_population=data\hyps, resume_evolve=None, bucket=, cache=ram, image_weights=False, False, single_cls=False, optimizer=SGD, sync_bn=False, workers=8, project=runs/train, name=hyper_high_train, exist_ok=False, quad=False, cos_lr=False, label_smoothing=0.0, freeze=[0], save_period=-1, seed=0, local_rank=-1, entity=None, upload_dataset=False, bbox_interval=-1, artifact_alias=latest, ndjson_console=False, ndjson_github: up to date with https://github.com/ultralytics/yolov5
YOLOv5 v7.0-411-gf4d8a84c Python-3.9.21 torch-2.1.2+cu118 CUDA:0 (NVIDIA GeForce RTX 3070 Laptop GPU, 8192MiB)

hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.3, cls_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.9, shear=0.0, perspective=0.0, flipud=0.0, flip_xup=0.1, copy_paste=0.1
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5 runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Overriding model.yaml nc=80 with nc=5

```

	from	n	params	module	arguments
0	-1	1	3520	models.common.Conv	[3, 32, 6, 2, 2]
1	-1	1	18560	models.common.Conv	[32, 64, 3, 2]
2	-1	1	18816	models.common.C3	[64, 64, 1]
3	-1	1	73984	models.common.Conv	[64, 128, 3, 2]
4	-1	2	115712	models.common.C3	[128, 128, 2]
5	-1	1	295424	models.common.Conv	[128, 256, 3, 2]
6	-1	3	625152	models.common.C3	[256, 256, 3]
7	-1	1	1180672	models.common.Conv	[256, 512, 3, 2]
8	-1	1	1182720	models.common.C3	[512, 512, 1]
9	-1	1	656896	models.common.SPPF	[512, 512, 5]
10	-1	1	131584	models.common.Conv	[512, 256, 1, 1]
11	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
12	[-1, 6]	1	0	models.common.Concat	[1]
13	-1	1	361984	models.common.C3	[512, 256, 1, False]
14	-1	1	33024	models.common.Conv	[256, 128, 1, 1]
15	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']

Figure 12: Training with modified hyperparameters


```
Epoch GPU_mem box_loss obj_loss cls_loss Instances Size
98/99 3.65G 0.03207 0.01498 0.009541 103 640: 100% | 5/5 [00:00<00:00, 5.96it/s]
Class Images Instances P R mAP50 mAP50-95: 100% | 1/1 [00:00<00:00, 3.57it/s]
all 20 47 0.86 0.801 0.915 0.702

Epoch GPU_mem box_loss obj_loss cls_loss Instances Size
99/99 3.65G 0.03154 0.01427 0.008678 66 640: 100% | 5/5 [00:00<00:00, 6.11it/s]
Class Images Instances P R mAP50 mAP50-95: 100% | 1/1 [00:00<00:00, 3.36it/s]
all 20 47 0.875 0.8 0.915 0.698

100 epochs completed in 0.044 hours.
Optimizer stripped from runs\train\hyper_high_train\weights\last.pt, 14.4MB
Optimizer stripped from runs\train\hyper_high_train\weights\best.pt, 14.4MB

Validating runs\train\hyper_high_train\weights\best.pt...
Fusing layers...
Model summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
Class Images Instances P R mAP50 mAP50-95: 100% | 1/1 [00:00<00:00, 3.06it/s]
all 20 47 0.856 0.801 0.915 0.709
tugger 20 4 0.768 0.5 0.781 0.639
cabinet 20 13 0.955 1 0.995 0.806
str 20 13 0.953 1 0.995 0.695
box 20 13 0.934 1 0.995 0.763
forklift 20 4 0.669 0.507 0.808 0.643

Results saved to runs\train\hyper_high_train
```

Figure 13: Completion output of hyperparameter-tuned training

2. TensorBoard Visualization

TensorBoard was used to visualize and compare training metrics between the standard and hyperparameter-tuned models. Both models were trained with the same dataset but different hyperparameter configurations.

a. Model Performance Comparison

Various metrics were used to compare the performance of both models, including F1 score, mean average precision (mAP), and loss curves.

b. F1 Score Curves

Standard model:

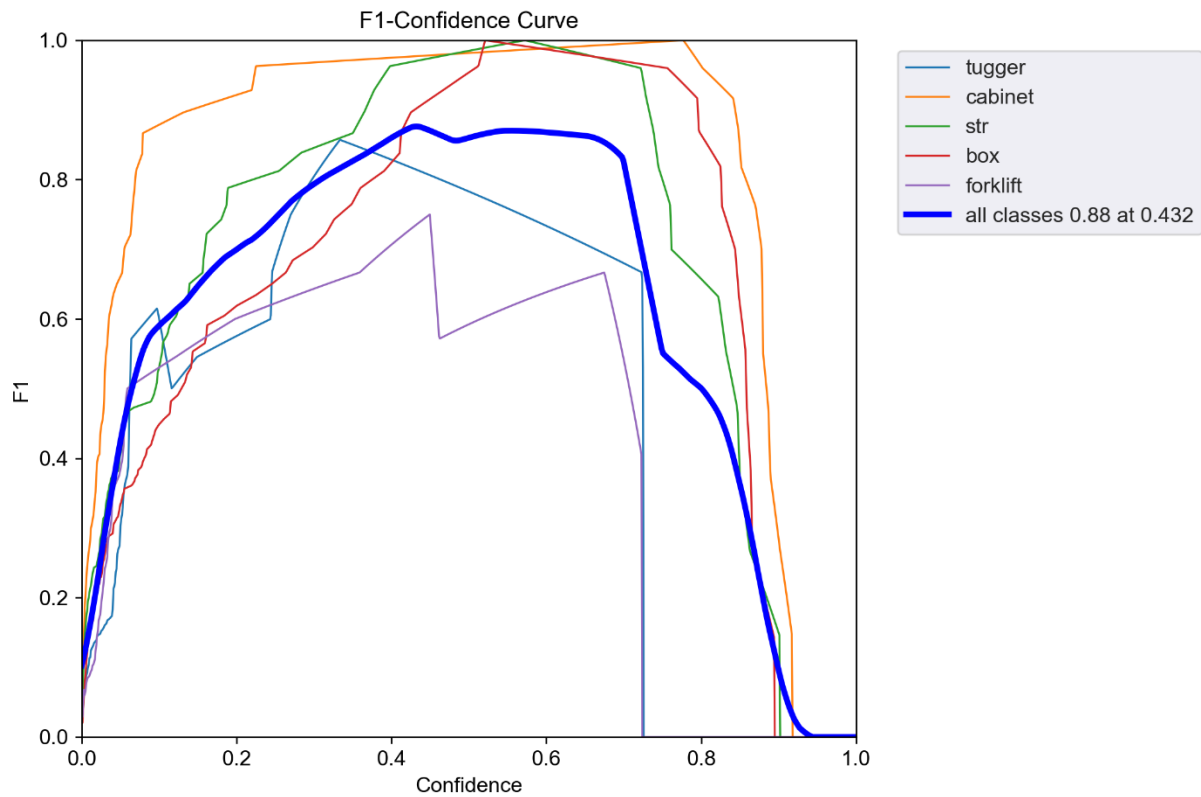


Figure 14: F1 score curve for the standard model

Hyperparameter-tuned model:

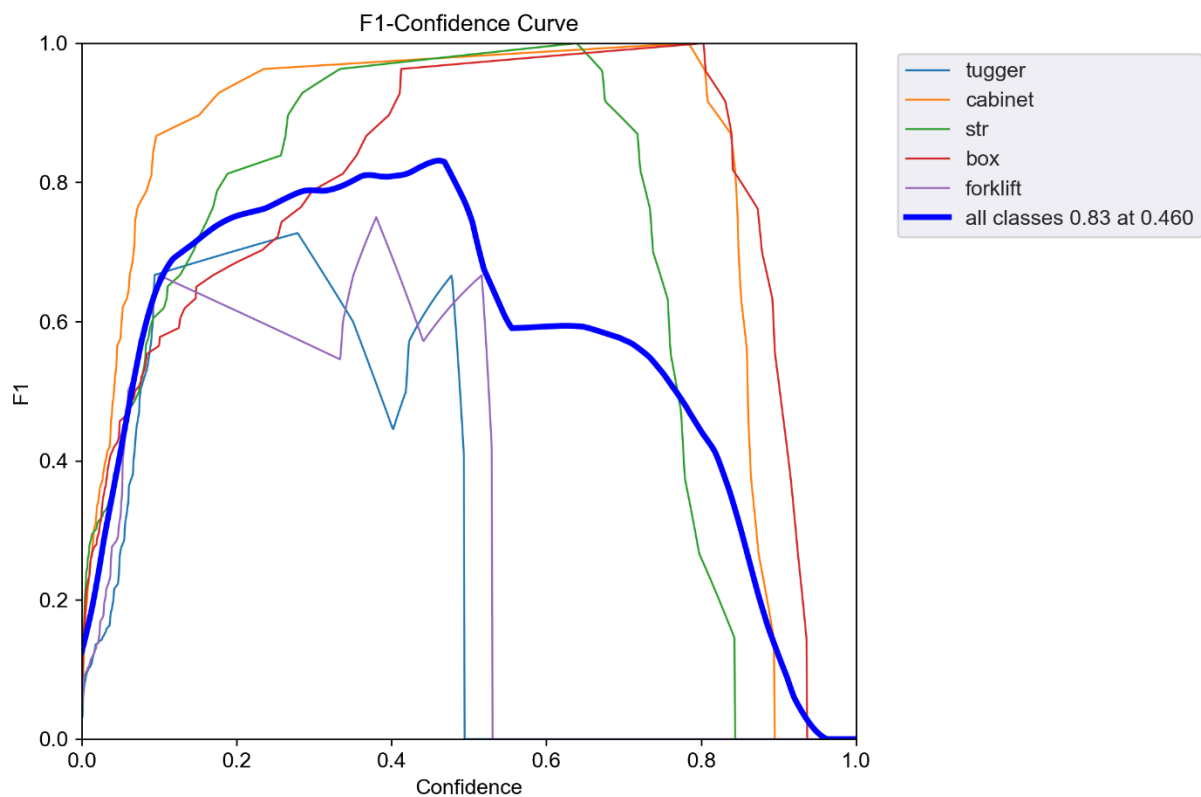


Figure 15: F1 score curve for the hyperparameter-tuned model

c. Mean Average Precision (mAP)

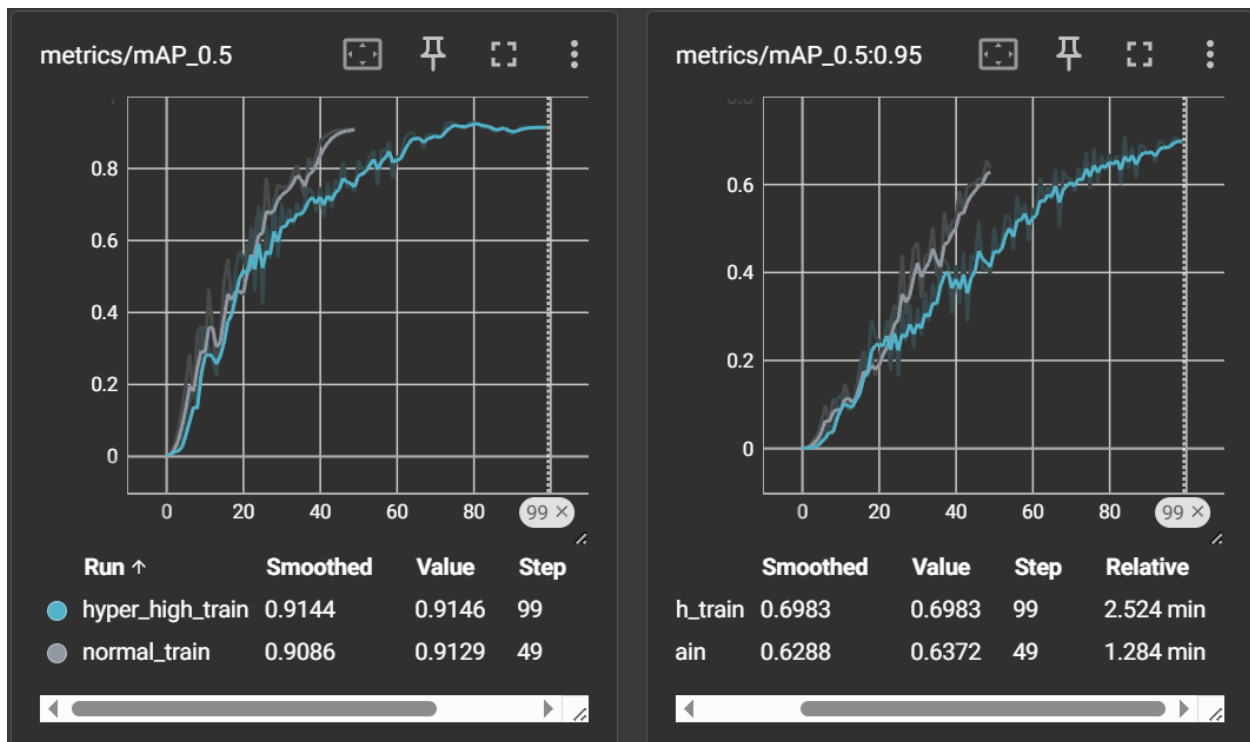


Figure 16: Comparison of mAP@0.5 and mAP@0.5:0.95 between both models

b. Loss Comparison

Loss during training for both models:

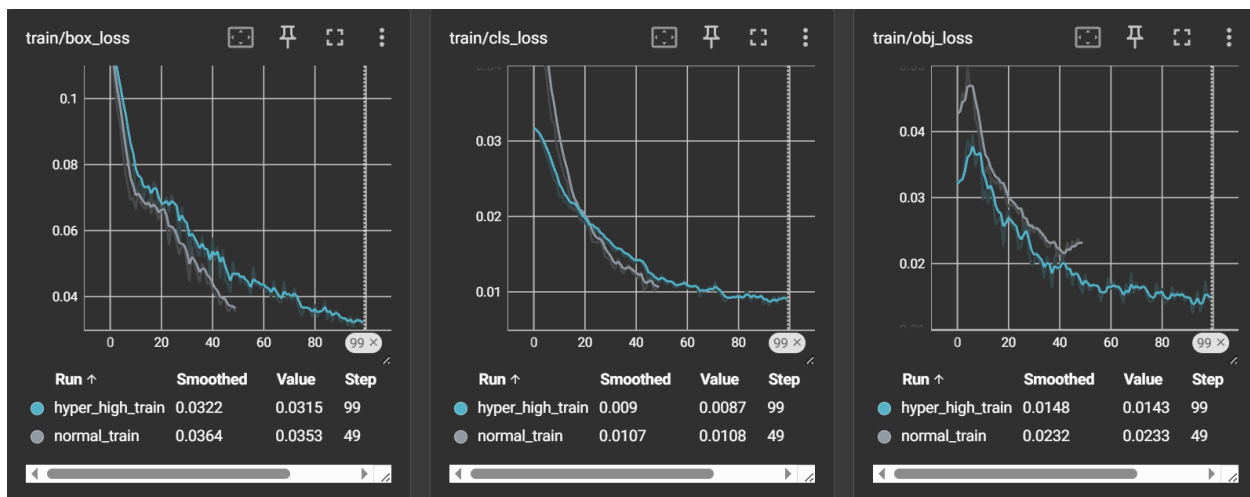


Figure 17: Box loss, class loss, and object loss during training for both models

Validation losses for both models:

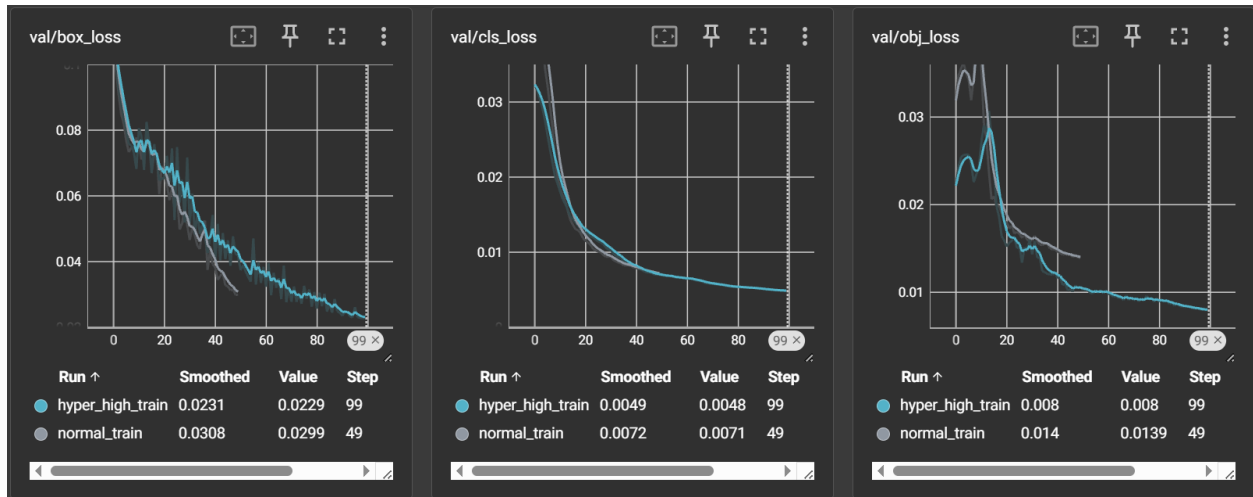


Figure 18: Validation losses for both models

c. Learning Rate

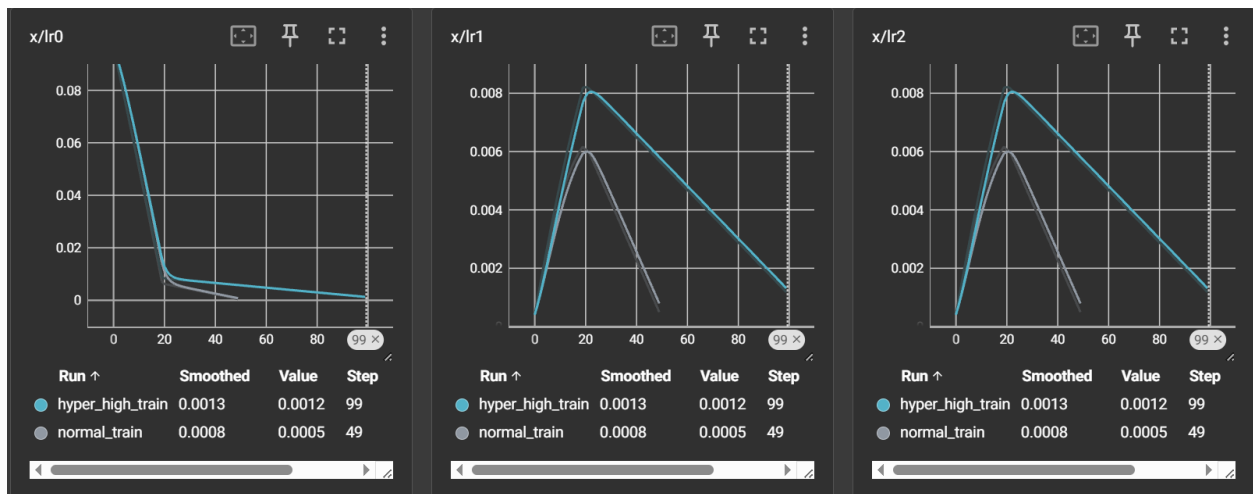


Figure 19: Learning rate schedules for both models

d. Confusion Matrices

Standard model:

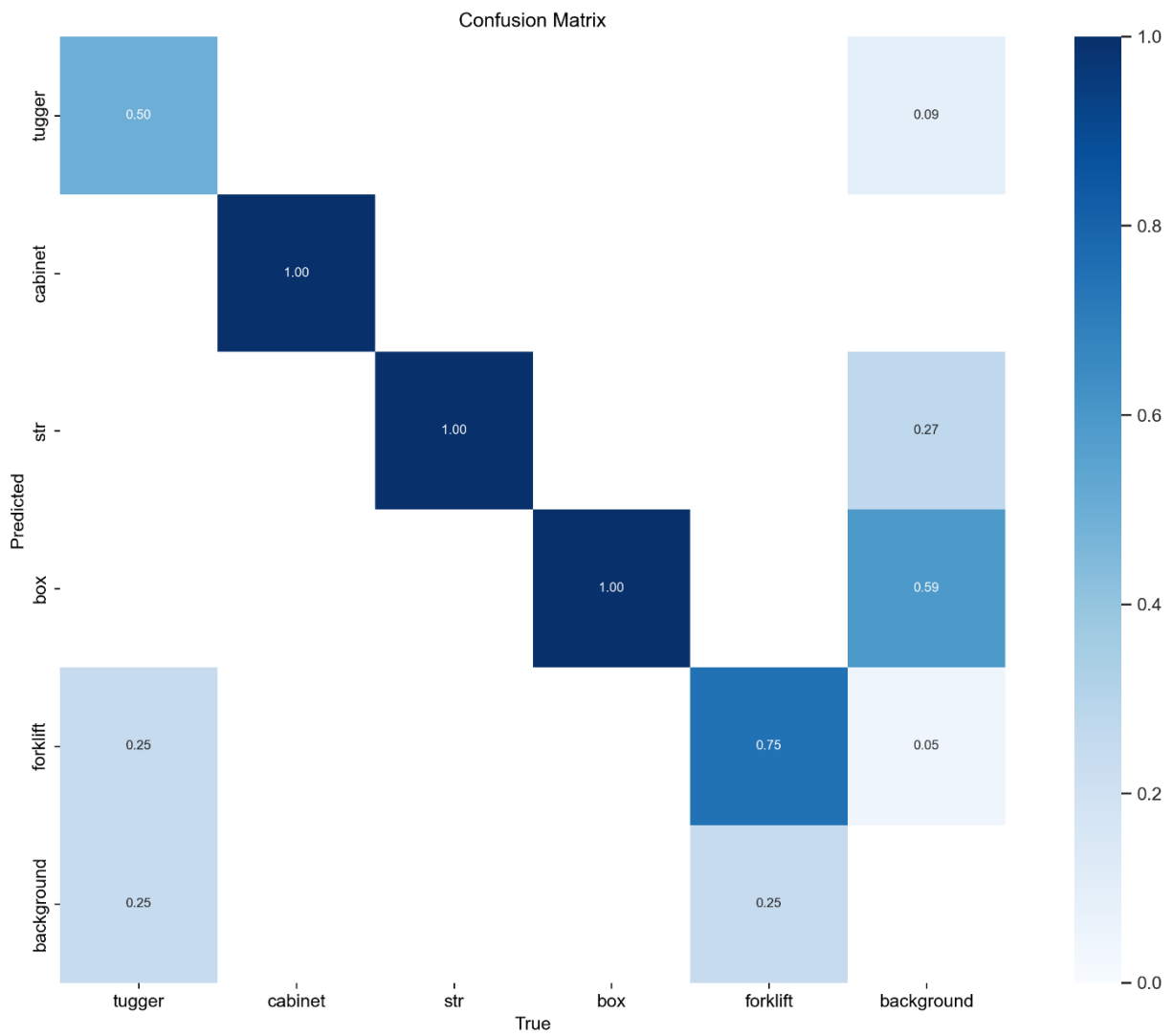


Figure 20: Confusion matrix for the standard model

Hyperparameter-tuned model:

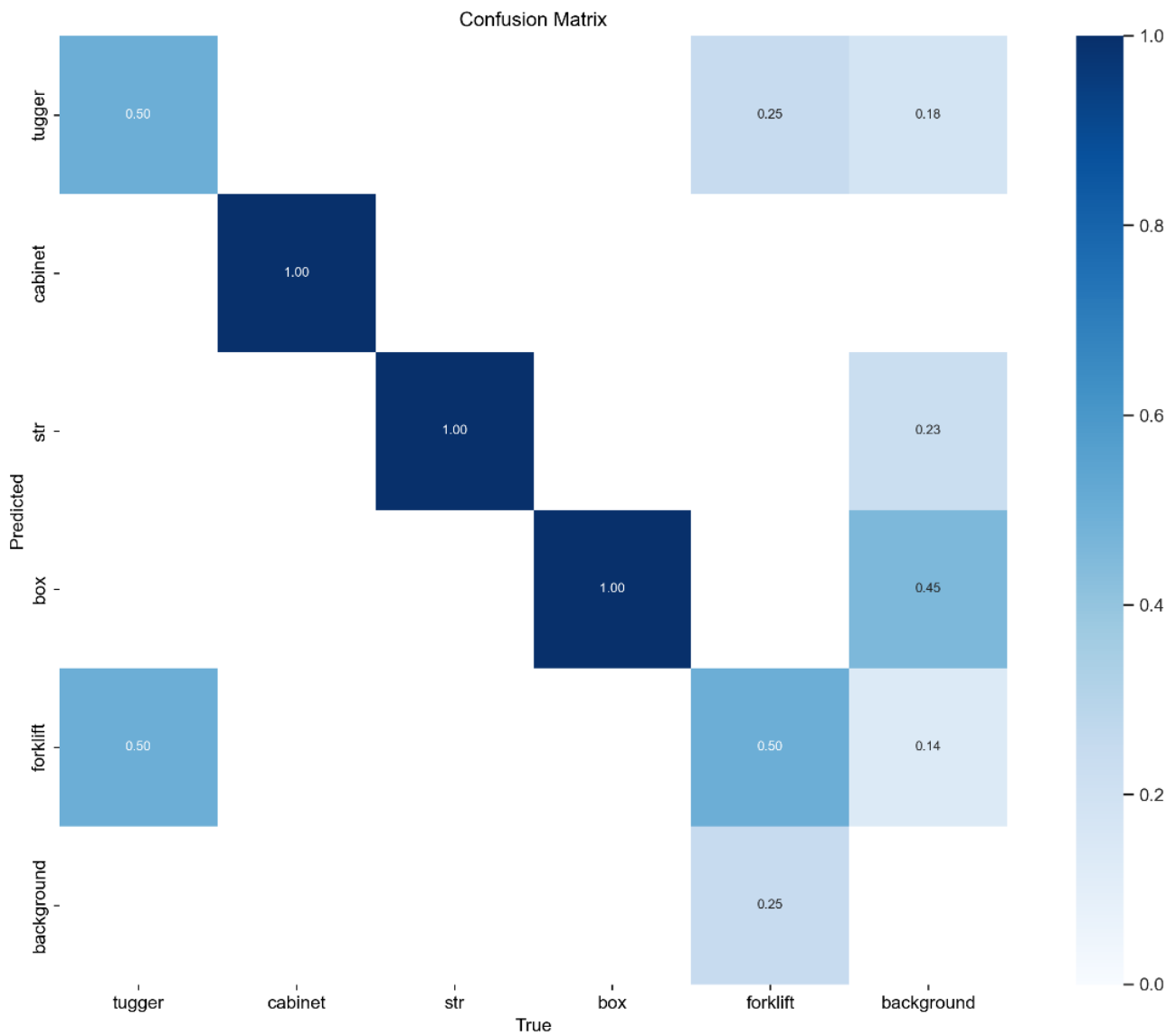


Figure 21: Confusion matrix for the hyperparameter-tuned model

Part 3: Model Deployment & Inference

1. Export Models to ONNX Format

To enable efficient inference on various platforms, both trained models were exported to the ONNX (Open Neural Network Exchange) format. This conversion allows the models to be used with a wide range of inference engines.

The export process maintained the input size of 640×640 pixels and used dynamic axes to support variable batch sizes during inference.

2. Network Visualization with Netron

The exported ONNX models were visualized using Netron, a web-based neural network visualization tool. This provides insight into the model architecture and helps verify the correct export of the models.

Standard YOLOv5s model architecture:

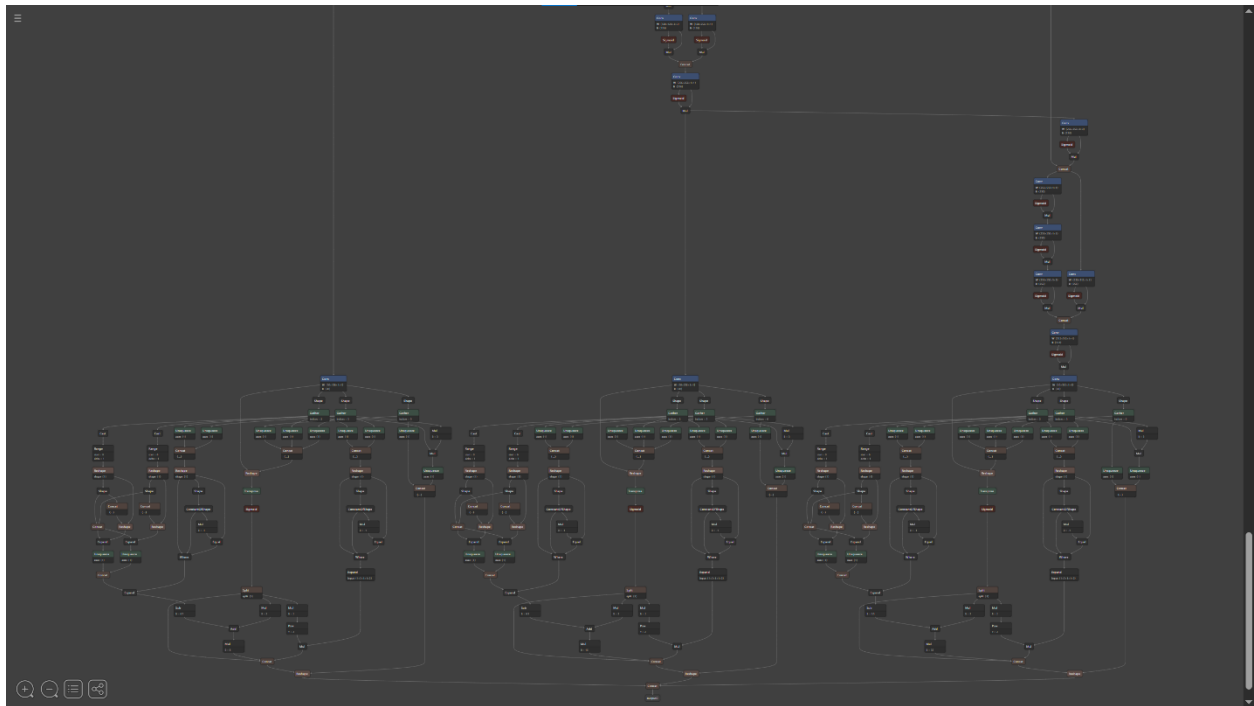


Figure 22: Netron visualization of the standard YOLOv5s model architecture

Hyperparameter-tuned model architecture:

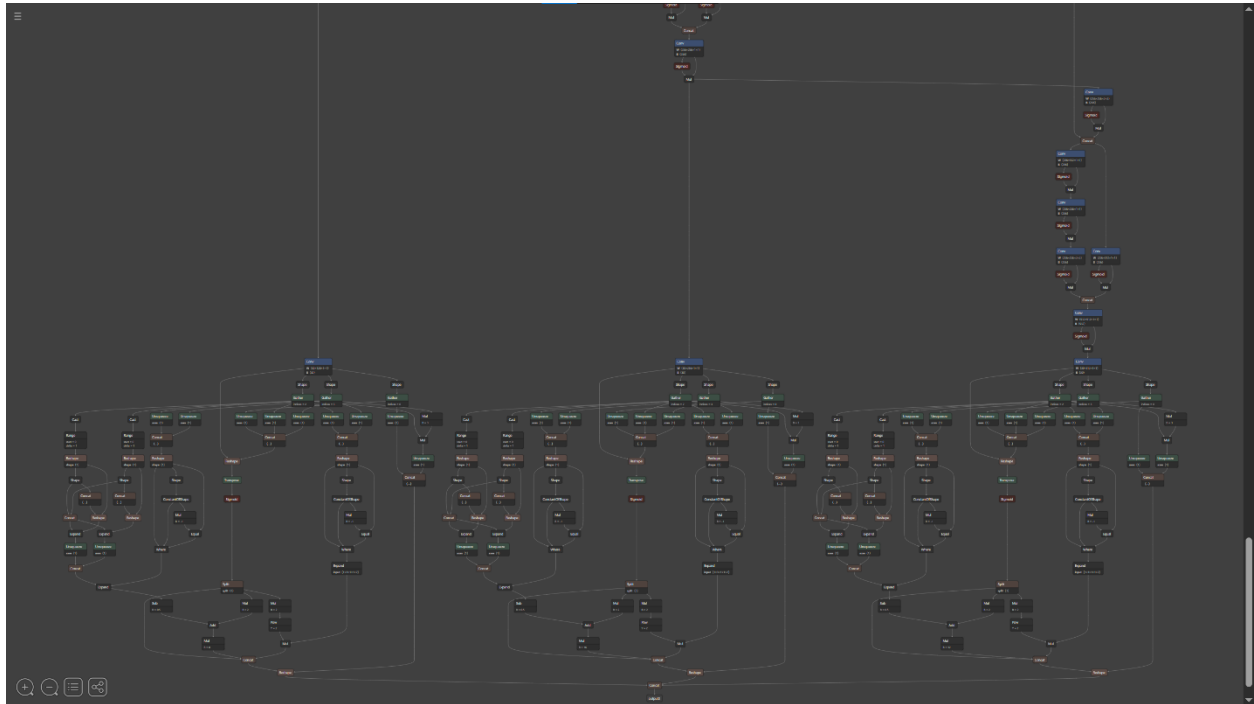


Figure 23: Netron visualization of the hyperparameter-tuned model architecture

3. REST API Implementation with FastAPI

A REST API was developed using FastAPI to serve the trained object detection models. The API provides endpoints for real-time inference on uploaded images, returning both JSON output and visualized results.

a. API Endpoints

The API offers the following endpoints:

1. **GET /:** Redirects to the API documentation
2. **GET /models:** Lists all available ONNX models
3. **POST /inference:** Accepts an image and returns detected objects as JSON
4. **POST /inference_image:** Accepts an image and returns the same image with bounding boxes drawn

b. Architecture

The API implementation includes:

- Non-Maximum Suppression (NMS) for filtering redundant detections
- Custom logic for processing YOLOv5 output tensors
- Visualization functionality for displaying detection results

c. Sample JSON Output

```
[
  {
    "Id": 172779,
    "ObjectClassName": "cabinet",
    "ObjectClassId": 2,
    "Left": 398,
    "Top": 23,
    "Right": 652,
    "Bottom": 427,
    "x_center": 0.2734,
    "y_center": 0.2083,
    "width": 0.1323,
    "height": 0.3741,
    "Confidence": 0.8976
  }
]
```

d. Testing FastAPI Endpoints

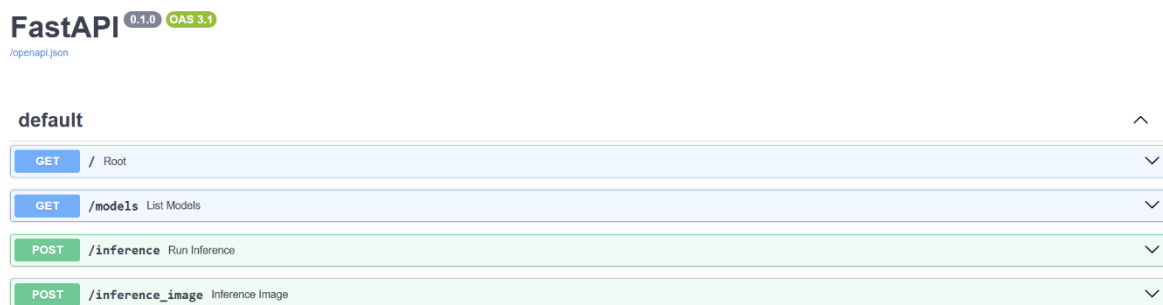


Figure 24: FastAPI docs page showing project endpoints

Models Endpoint:

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8000/models' \
  -H 'accept: application/json'
```

Request URL

http://localhost:8000/models

Server response

Code	Details
200	<p>Response body</p> <pre>{ "hyper_high_train/weights/best.onnx", "normal_train/weights/best.onnx" }</pre> <p>Response headers</p> <pre>content-length: 71 content-type: application/json date: Tue, 01 Apr 2025 14:18:27 GMT server: uvicorn</pre>

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Figure 25: '/models' endpoint returning the available models

Inference Endpoint:

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/inference' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@1.png;type=image/png'
```

Request URL

http://localhost:8000/inference

Server response

Code	Details
200	<p>Response body</p> <pre>{ { "id": 172779, "objectClassName": "box", "objectClassid": 4, "left": 194, "top": 254, "right": 526, "bottom": 369, "x_center": 0.406708288192749, "y_center": 0.48271450996398924, "width": 0.29574586391448973, "height": 0.1716749668121338, "confidence": 0.8579883715515137 }, { "id": 172780, "objectClassName": "str", "objectClassid": 3, "left": 81, "top": 222, "right": 526, "bottom": 386, "x_center": 0.4826814651489258, "y_center": 0.6319458961486817, "width": 0.6808085568695069, "height": 0.5688458919525167 } }</pre> <p>Response headers</p> <pre>content-length: 754 content-type: application/json date: Tue, 01 Apr 2025 14:19:24 GMT server: uvicorn</pre>

Figure 26: '/Inference' Endpoint showing the json output which includes the bounding boxes

Image Inference Endpoint:

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8000/inference_image' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file@52.png;type=image/png'
```

Request URL

```
http://localhost:8000/inference_image
```

Server response

Code

Details

200

Response body

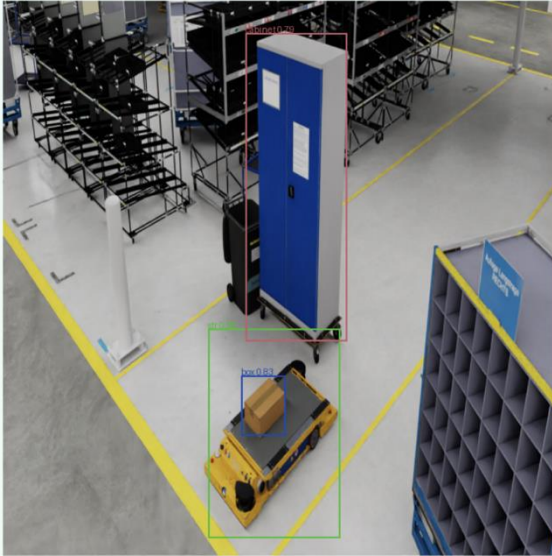


Figure 27: Image inference endpoint showing the image and the returned bounding boxes

4. Docker Containerization

The API was containerized using Docker to ensure consistent deployment across different environments. This approach simplifies installation and eliminates potential dependency issues.

a. Dockerfile

A Dockerfile was created to specify the Python environment and dependencies required for the API.

b. Docker Compose

A docker-compose.yml file was configured to simplify the deployment process, including port mapping and volume mounting for model files.

c. Running the Containerized API

The containerized API can be started using Docker Compose, providing an easy-to-deploy solution for inference.

Docker Compose build process initiation:

```
PS C:\Users\hassa\OneDrive\Documents\GitHub\InMind-Project> cd .\Project\Part_3\  
PS C:\Users\hassa\OneDrive\Documents\GitHub\InMind-Project\Project\Part_3> docker-compose up --build  
[+] Building 96.6s (13/13) FINISHED                                docker:desktop-linux  
=> [api internal] load build definition from Dockerfile            0.0s  
=> => transferring dockerfile: 396B                                0.0s  
=> [api internal] load metadata for docker.io/library/python:3.10-slim 2.9s  
=> [api auth] library/python:pull token for registry-1.docker.io    0.0s  
=> [api internal] load .dockerignore                               0.0s  
=> => transferring context: 2B                                       0.0s
```

Figure 28: Docker Compose build process initiation

Completion of the Docker container build:

```
[+] Running 3/3  
✓ api Built 0.0s  
✓ Network part_3_default Created 0.3s  
✓ Container part_3-api-1 Created 0.4s  
Attaching to api-1  
api-1 | INFO: Started server process [1]  
api-1 | INFO: Waiting for application startup.  
api-1 | INFO: Application startup complete.  
api-1 | INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)  
[+] View in Docker Desktop [o] View Config [W] Enable Watch
```

Figure 29: Successful completion of the Docker container build

Container logs in the Docker desktop app:

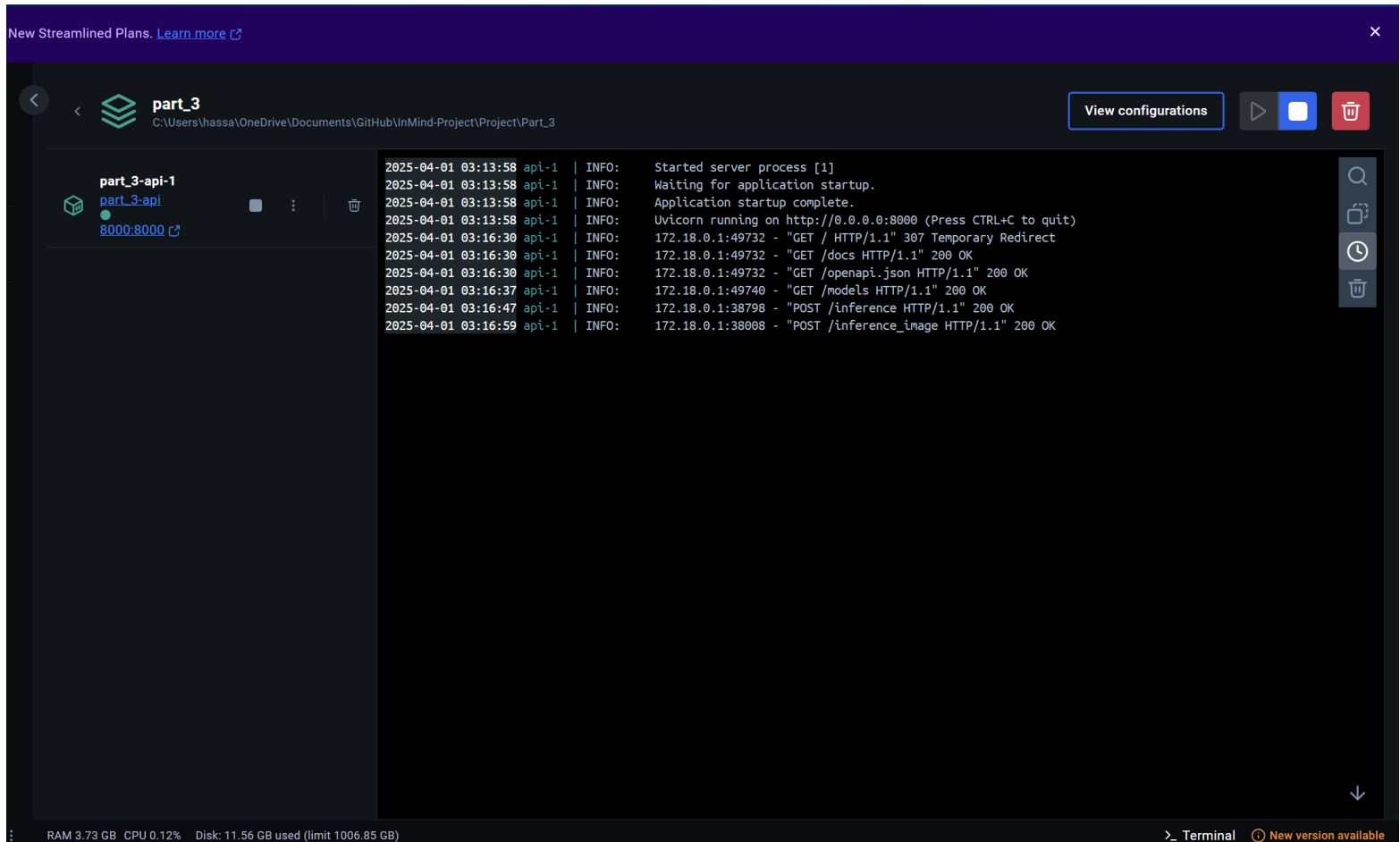


Figure 30: Container logs showing API endpoint requests and responses

Conclusion

The final object detection Project successfully implemented an object detection system for industrial environments using YOLOv5. The project encompassed the complete machine learning pipeline from data preparation to model deployment.

Performance evaluation showed that while hyperparameter tuning was applied, the improvements were minimal due to the limited size and imbalanced nature of the dataset. Future work could focus on expanding the dataset and exploring more advanced architectures.

The final solution provides a containerized API that can be easily integrated into existing industrial systems for real-time object detection.

References

1. Ultralytics YOLOv5. (2022). GitHub repository. <https://github.com/ultralytics/yolov5>
2. ONNX Runtime. (2022). GitHub repository. <https://github.com/microsoft/onnxruntime>
3. FastAPI. (2022). Documentation. <https://fastapi.tiangolo.com/>
4. Albumentations. (2022). Documentation. <https://albumentations.ai/docs/>

Appendix

System Requirements

The project dependencies include:

- torch==2.1.2+cu118
- torchvision==0.16.2+cu118
- matplotlib
- albumentations
- numpy
- tensorboard
- onnx
- onnxruntime
- fastapi
- uvicorn
- python-multipart

Model Configuration Details

The detailed configuration of the YOLOv5 models including layer architecture and hyperparameters can be found in the model summary output and the hyperparameter configuration files.