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

Table Of Figures

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

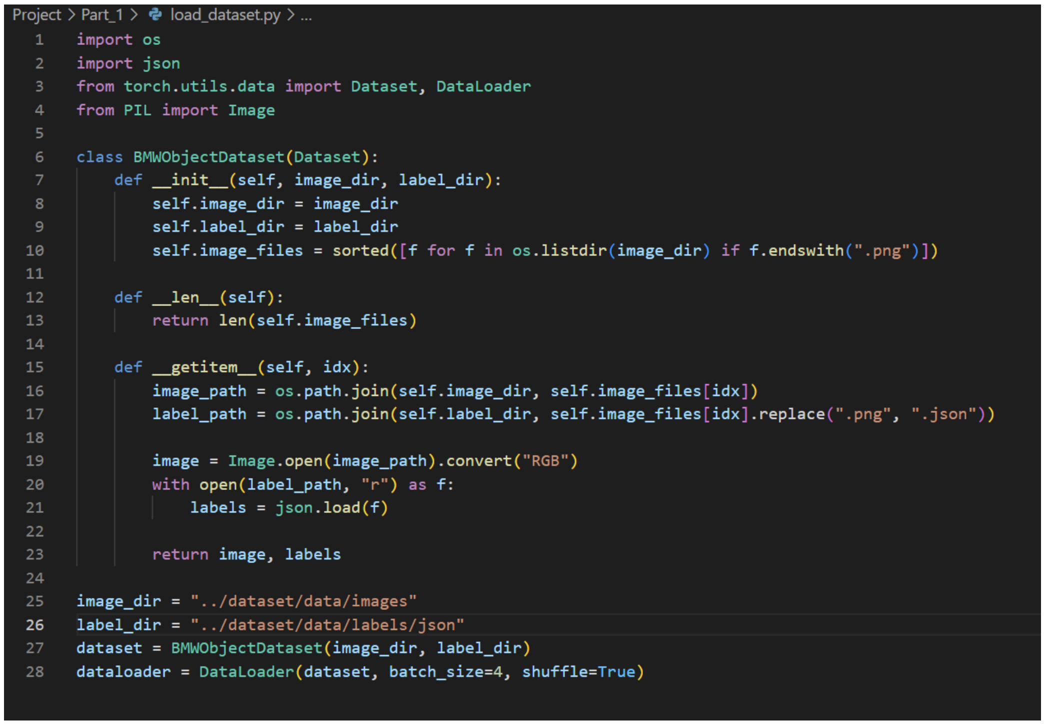
Each section describes the methodologies, implementation details, and results achieved throughout the development process.

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 > ...

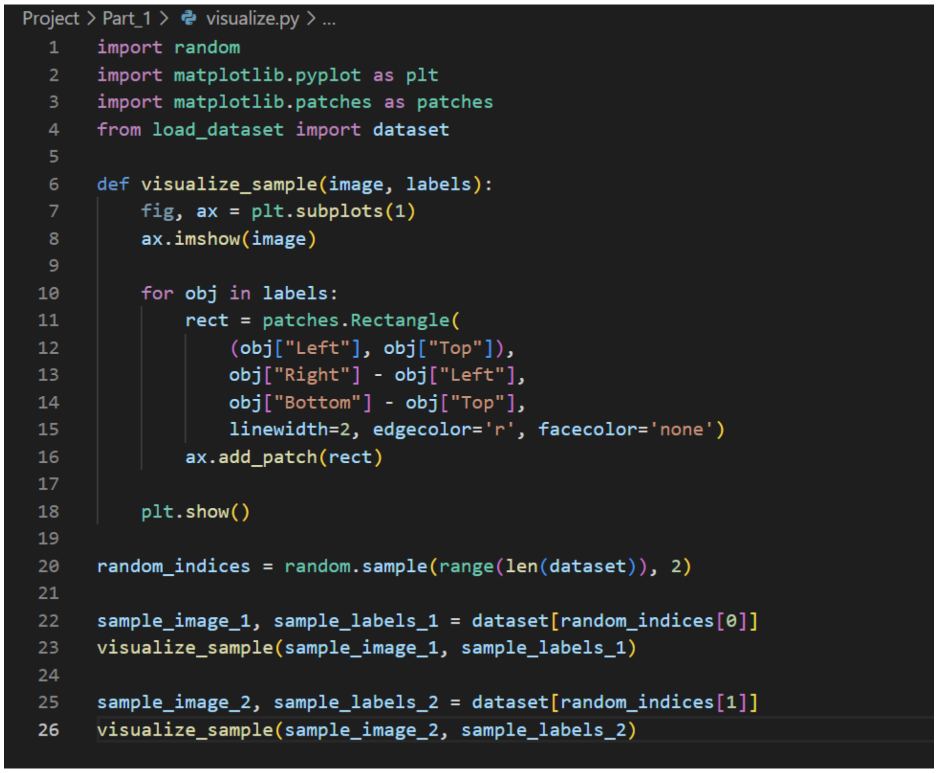
```python
import os
import json
from torch.utils.data import Dataset, DataLoader
from PIL import Image

class BMWObjectDataset(Dataset):
    def __init__(self, image_dir, label_dir):
        self.image_dir = image_dir
        self.label_dir = label_dir
        self.image_files = sorted([f for f in os.listdir(image_dir) if f.endswith(".png")])

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        image_path = os.path.join(self.image_dir, self.image_files[idx])
        label_path = os.path.join(self.label_dir, self.image_files[idx].replace(".png", ".json"))

        image = Image.open(image_path).convert("RGB")
        with open(label_path, "r") as f:
            labels = json.load(f)

        return image, labels

image_dir = "../dataset/data/images"
label_dir = "../dataset/data/labels/json"
dataset = BMWObjectDataset(image_dir, label_dir)
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



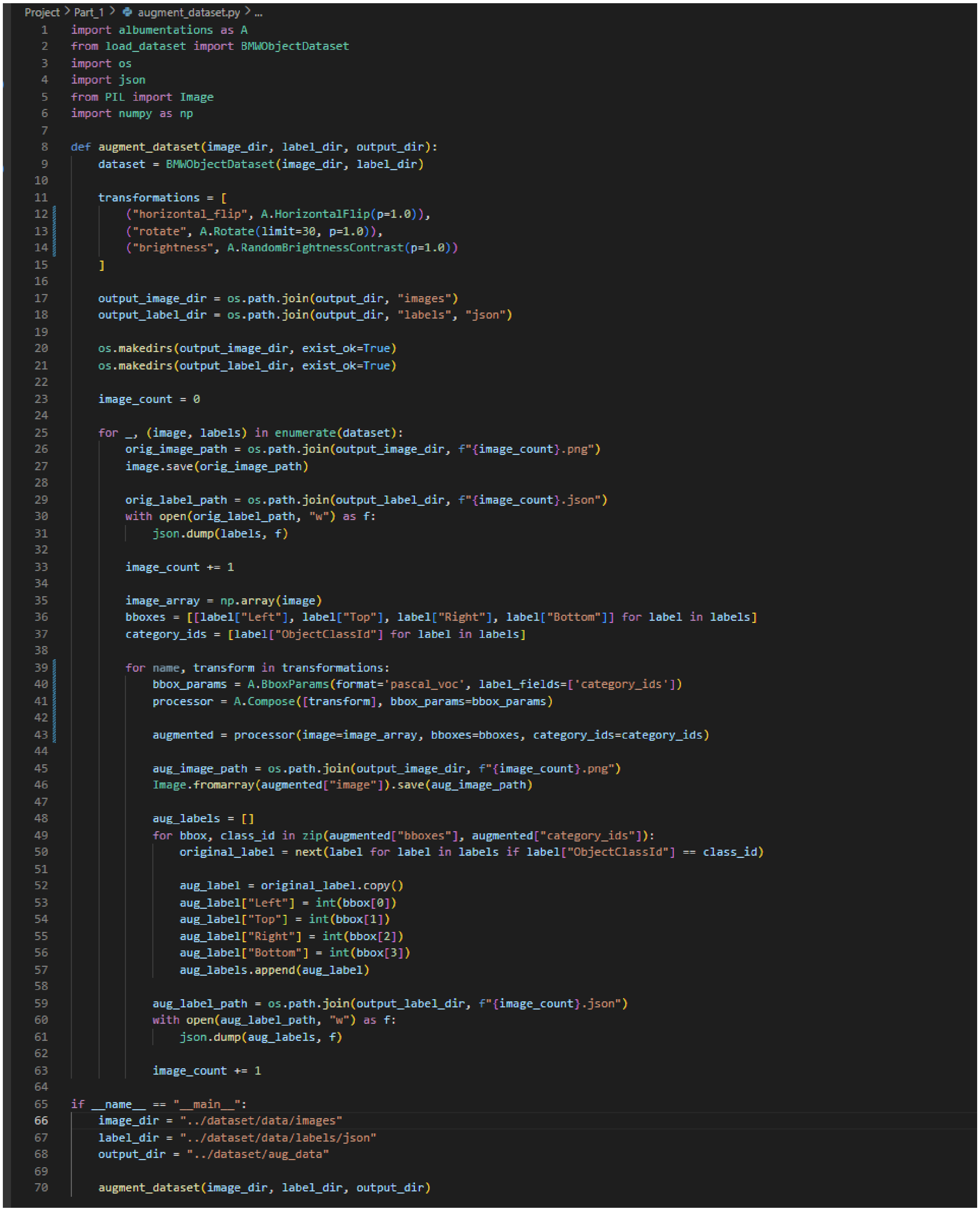
```
Project > Part_1 > visualize.py > ...
import random
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from load_dataset import dataset

def visualize_sample(image, labels):
    fig, ax = plt.subplots(1)
    ax.imshow(image)

    for obj in labels:
        rect = patches.Rectangle(
            (obj["Left"], obj["Top"]),
            obj["Right"] - obj["Left"],
            obj["Bottom"] - obj["Top"],
            linewidth=2, edgecolor='r', facecolor='none')
        ax.add_patch(rect)

    plt.show()

random_indices = random.sample(range(len(dataset)), 2)

sample_image_1, sample_labels_1 = dataset[random_indices[0]]
visualize_sample(sample_image_1, sample_labels_1)

sample_image_2, sample_labels_2 = dataset[random_indices[1]]
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



```python
import albumentations as A
from load_dataset import BMWObjectDataset
import os
import json
from PIL import Image
import numpy as np

def augment_dataset(image_dir, label_dir, output_dir):
    dataset = BMWObjectDataset(image_dir, label_dir)

    transformations = [
        ("horizontal_flip", A.HorizontalFlip(p=1.0)),
        ("rotate", A.Rotate(limit=30, p=1.0)),
        ("brightness", A.RandomBrightnessContrast(p=1.0))
    ]

    output_image_dir = os.path.join(output_dir, "images")
    output_label_dir = os.path.join(output_dir, "labels", "json")

    os.makedirs(output_image_dir, exist_ok=True)
    os.makedirs(output_label_dir, exist_ok=True)

    image_count = 0

    for _, (image, labels) in enumerate(dataset):
        orig_image_path = os.path.join(output_image_dir, f"{image_count}.png")
        image.save(orig_image_path)

        orig_label_path = os.path.join(output_label_dir, f"{image_count}.json")
        with open(orig_label_path, "w") as f:
            json.dump(labels, f)

        image_count += 1

        image_array = np.array(image)
        bboxes = [[label["Left"], label["Top"], label["Right"], label["Bottom"]] for label in labels]
        category_ids = [label["ObjectClassId"] for label in labels]

        for name, transform in transformations:
            bbox_params = A.BboxParams(format='pascal_voc', label_fields=['category_ids'])
            processor = A.Compose([transform], bbox_params=bbox_params)

            augmented = processor(image=image_array, bboxes=bboxes, category_ids=category_ids)

            aug_image_path = os.path.join(output_image_dir, f"{image_count}.png")
            Image.fromarray(augmented["image"]).save(aug_image_path)

            aug_labels = []
            for bbox, class_id in zip(augmented["bboxes"], augmented["category_ids"]):
                original_label = next(label for label in labels if label["ObjectClassId"] == class_id)

                aug_label = original_label.copy()
                aug_label["Left"] = int(bbox[0])
                aug_label["Top"] = int(bbox[1])
                aug_label["Right"] = int(bbox[2])
                aug_label["Bottom"] = int(bbox[3])
                aug_labels.append(aug_label)

            aug_label_path = os.path.join(output_label_dir, f"{image_count}.json")
            with open(aug_label_path, "w") as f:
                json.dump(aug_labels, f)

            image_count += 1

if __name__ == "__main__":
    image_dir = "../dataset/data/images"
    label_dir = "../dataset/data/labels/json"
    output_dir = "../dataset/aug_data"

    augment_dataset(image_dir, label_dir, output_dir)
```

Figure 3: Implementation of data augmentation techniques

4. Augmentation Results

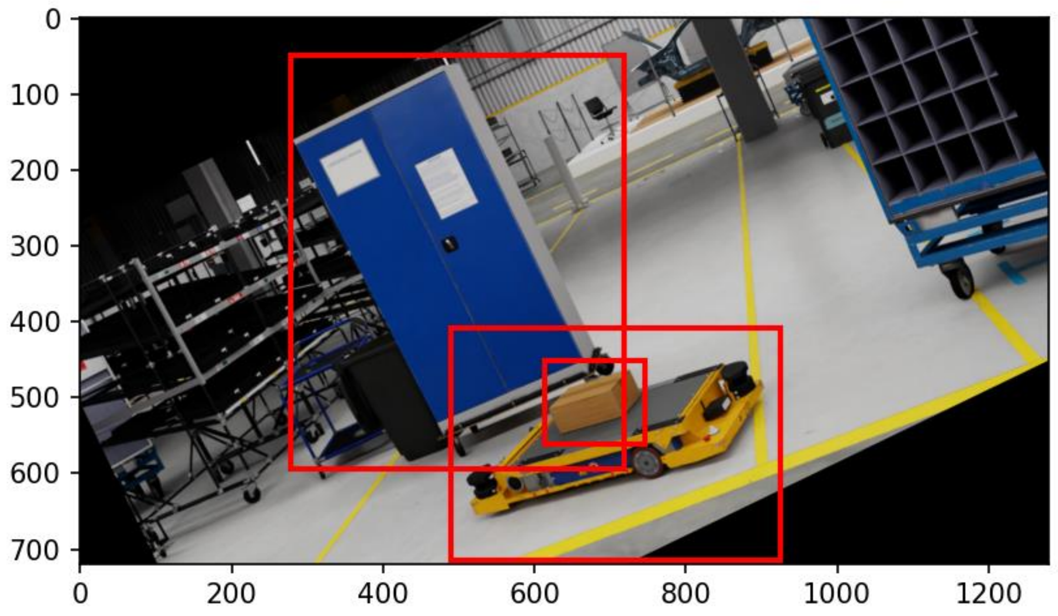
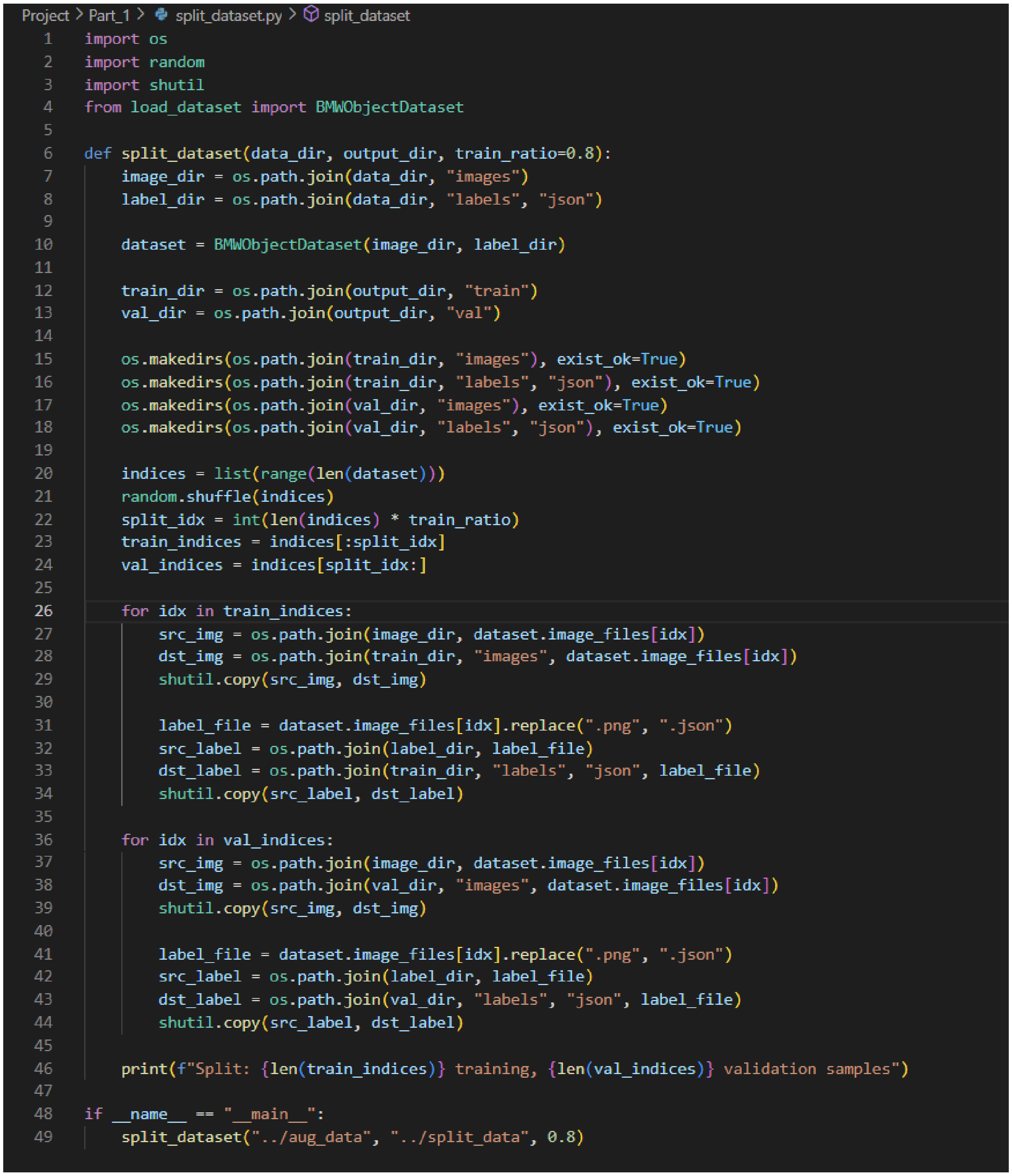


Figure 4: Example visualized image after augmenting it.

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

```python
import os
import random
import shutil
from load_dataset import BMWObjectDataset

def split_dataset(data_dir, output_dir, train_ratio=0.8):
    image_dir = os.path.join(data_dir, "images")
    label_dir = os.path.join(data_dir, "labels", "json")

    dataset = BMWObjectDataset(image_dir, label_dir)

    train_dir = os.path.join(output_dir, "train")
    val_dir = os.path.join(output_dir, "val")

    os.makedirs(os.path.join(train_dir, "images"), exist_ok=True)
    os.makedirs(os.path.join(train_dir, "labels", "json"), exist_ok=True)
    os.makedirs(os.path.join(val_dir, "images"), exist_ok=True)
    os.makedirs(os.path.join(val_dir, "labels", "json"), exist_ok=True)

    indices = list(range(len(dataset)))
    random.shuffle(indices)
    split_idx = int(len(indices) * train_ratio)
    train_indices = indices[:split_idx]
    val_indices = indices[split_idx:]

    for idx in train_indices:
        src_img = os.path.join(image_dir, dataset.image_files[idx])
        dst_img = os.path.join(train_dir, "images", dataset.image_files[idx])
        shutil.copy(src_img, dst_img)

        label_file = dataset.image_files[idx].replace(".png", ".json")
        src_label = os.path.join(label_dir, label_file)
        dst_label = os.path.join(train_dir, "labels", "json", label_file)
        shutil.copy(src_label, dst_label)

    for idx in val_indices:
        src_img = os.path.join(image_dir, dataset.image_files[idx])
        dst_img = os.path.join(val_dir, "images", dataset.image_files[idx])
        shutil.copy(src_img, dst_img)

        label_file = dataset.image_files[idx].replace(".png", ".json")
        src_label = os.path.join(label_dir, label_file)
        dst_label = os.path.join(val_dir, "labels", "json", label_file)
        shutil.copy(src_label, dst_label)

    print(f"Split: {len(train_indices)} training, {len(val_indices)} validation samples")

if __name__ == "__main__":
    split_dataset("../aug_data", "../split_data", 0.8)
```

Figure 5: Implementation of dataset splitting functionality

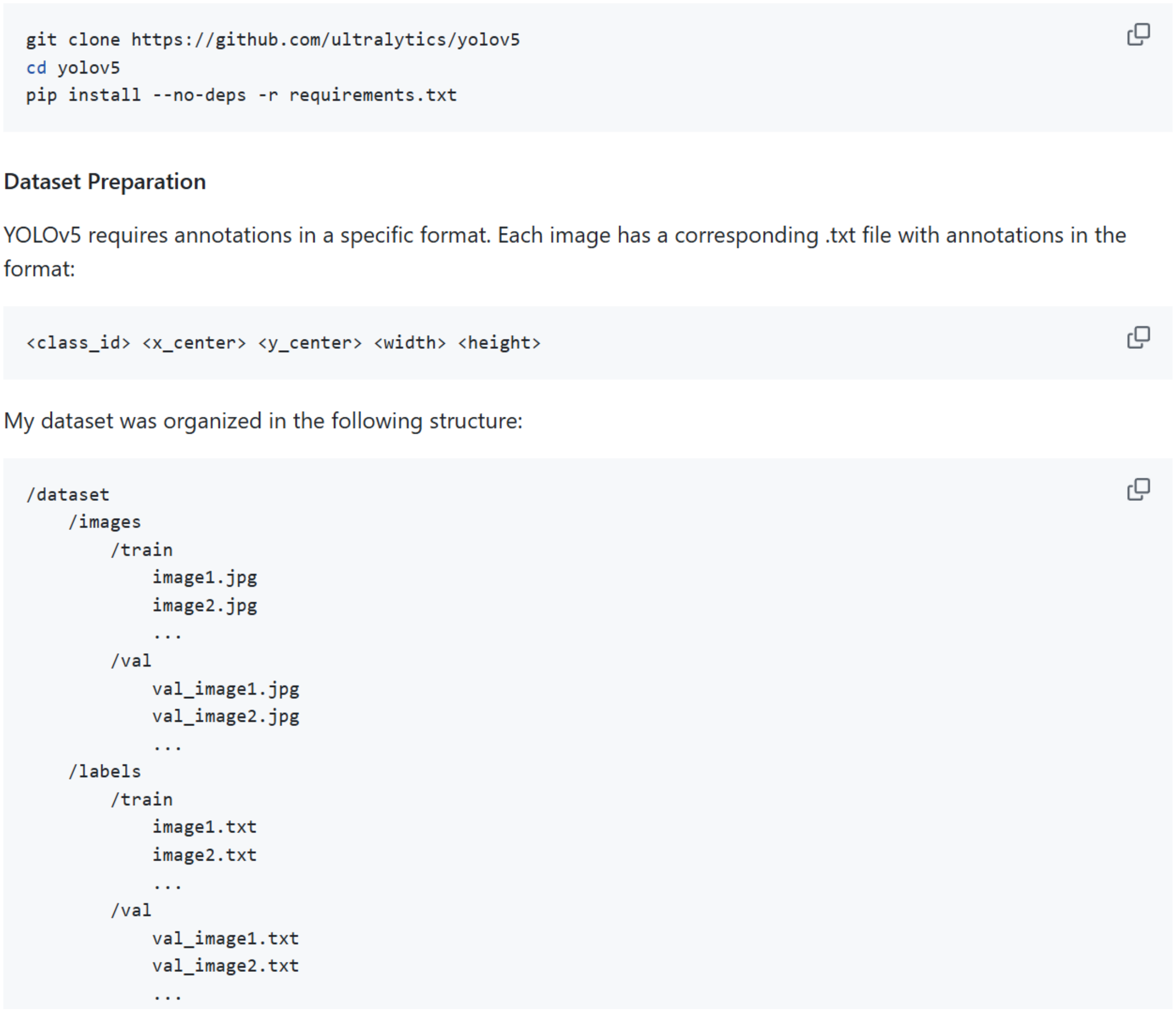
Part 2: Model Training & Evaluation

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:

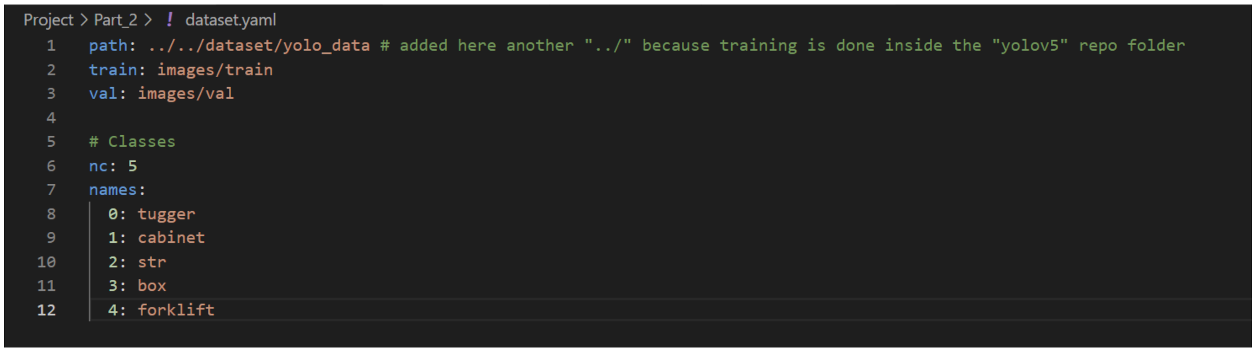
```
<class_id> <x_center> <y_center> <width> <height>
```

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



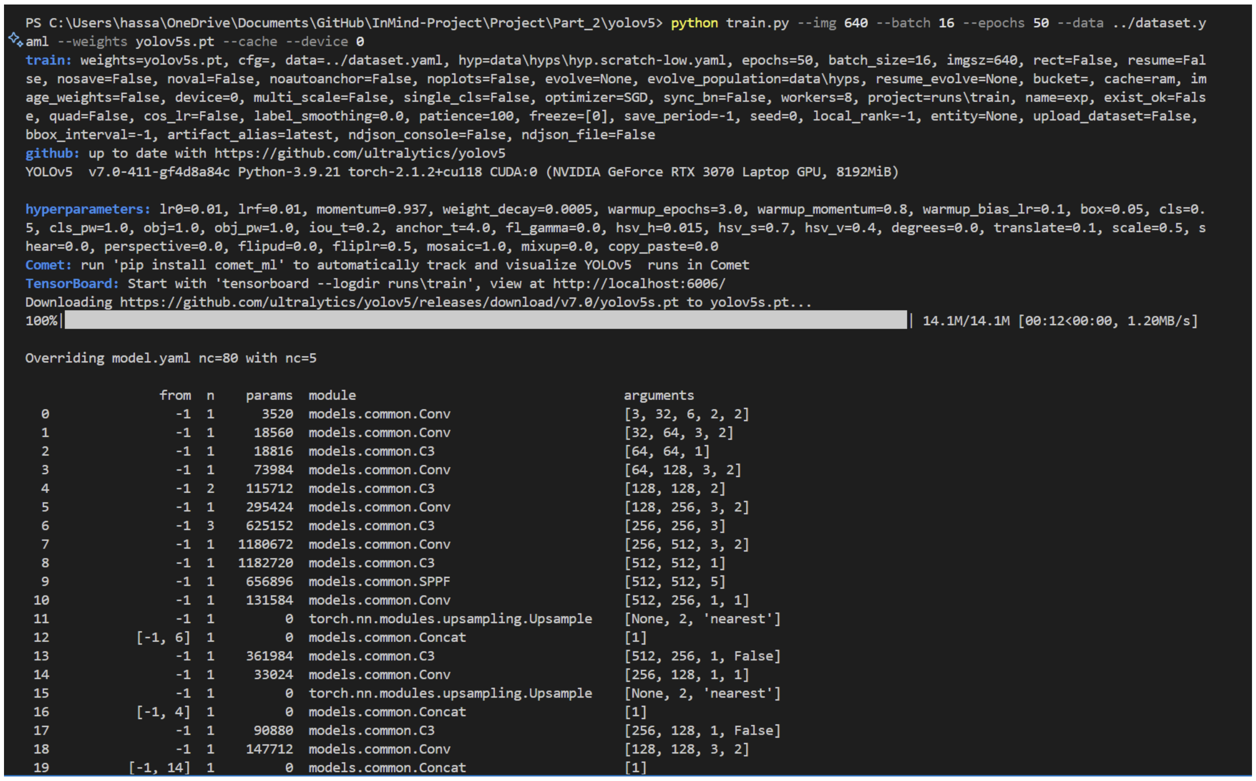
```yaml
Project > Part_2 > ! dataset.yaml
path: ../../dataset/yolo_data # added here another "../" because training is done inside the "yolov5" repo folder
train: images/train
val: images/val

# Classes
nc: 5
names:
  0: tugger
  1: cabinet
  2: str
  3: box
  4: forklift
```

Figure 7: Dataset configuration YAML for YOLOv5

c. Training Process

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

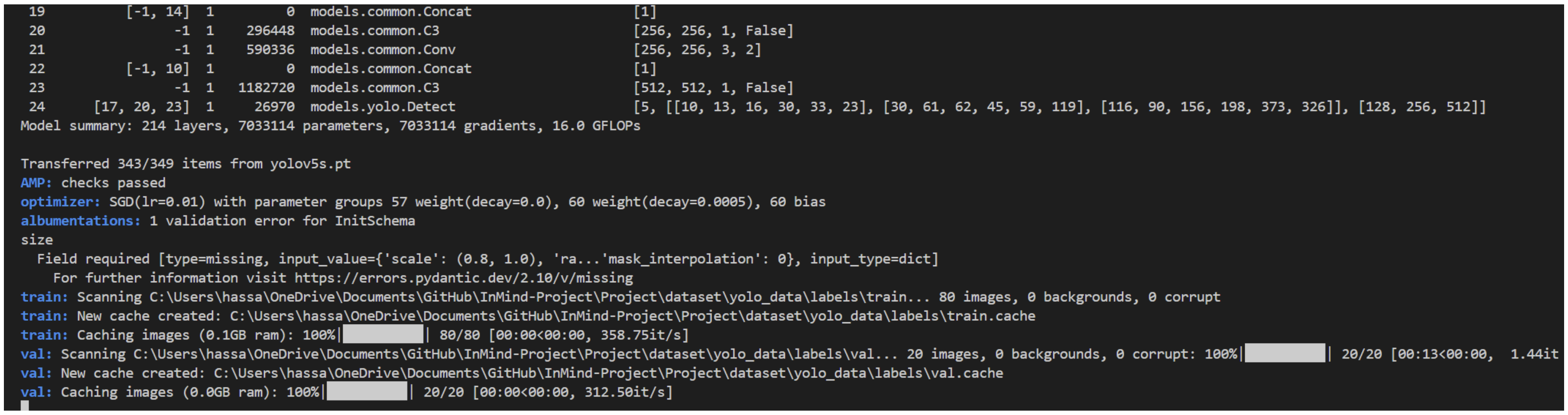
- Image size: 640×640 pixels
- Batch size: 16
- Number of epochs: 50
- Weights: YOLOv5s pre-trained model



```
PS C:\Users\hassa\OneDrive\Documents\GitHub\InMind-Project\Project\Part_2\yolov5> python train.py --img 640 --batch 16 --epochs 50 --data ../dataset.y
aml --weights yolov5s.pt --cache --device 0
train: weights=yolov5s.pt, cfg=, data=../dataset.yaml, hyp=data\hyps\hyp.scratch-low.yaml, epochs=50, batch_size=16, imgsz=640, rect=False, resume=Fal
se, nosave=False, noval=False, noautoanchor=False, noplots=False, evolve=None, evolve_population=data\hyps, resume_evolve=None, bucket=, cache=ram, im
age_weights=False, device=0, multi_scale=False, single_cls=False, optimizer=SGD, sync_bn=False, workers=8, project=runs\train, name=exp, exist_ok=Fals
e, 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: lr0=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=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=0.5, s
hear=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]
```

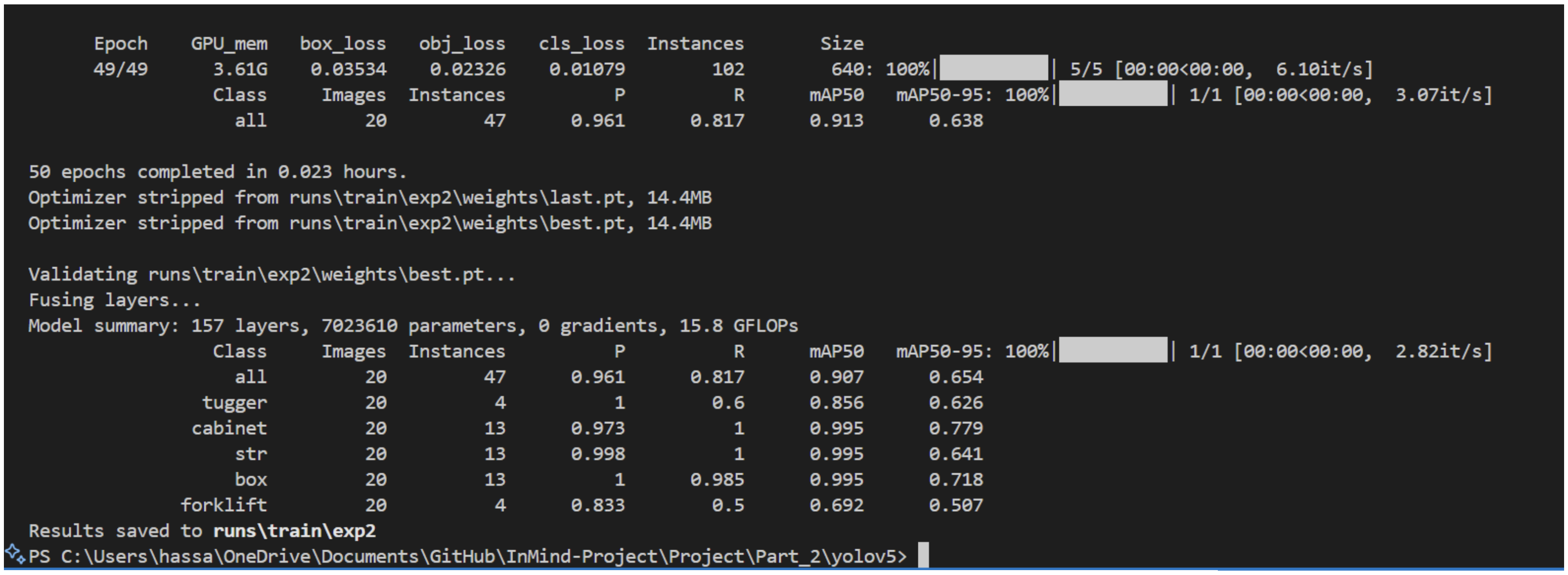
Figure 8: Training command execution and initial output

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\hassa\OneDrive\Documents\GitHub\InMind-Project\Project\dataset\yolo_data\labels\train... 80 images, 0 backgrounds, 0 corrupt
train: New cache created: C:\Users\hassa\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\hassa\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
val: New cache created: C:\Users\hassa\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

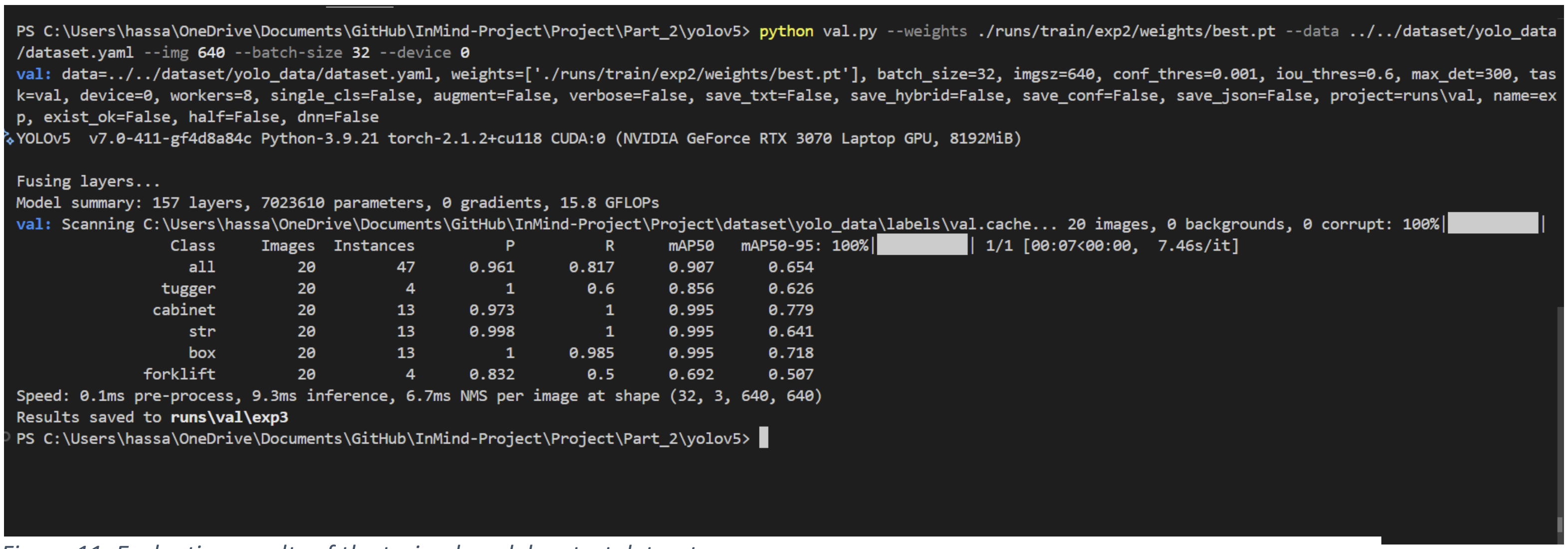


```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
      49/49      3.61G    0.03534    0.02326    0.01079        102        640: 100%|██████████| 5/5 [00:00<00:00,  6.10it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 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\hassa\OneDrive\Documents\GitHub\InMind-Project\Project\Part_2\yolov5>
```

Figure 10: Training completion output with final metrics

d. Model Evaluation and Hyperparameter Tuning

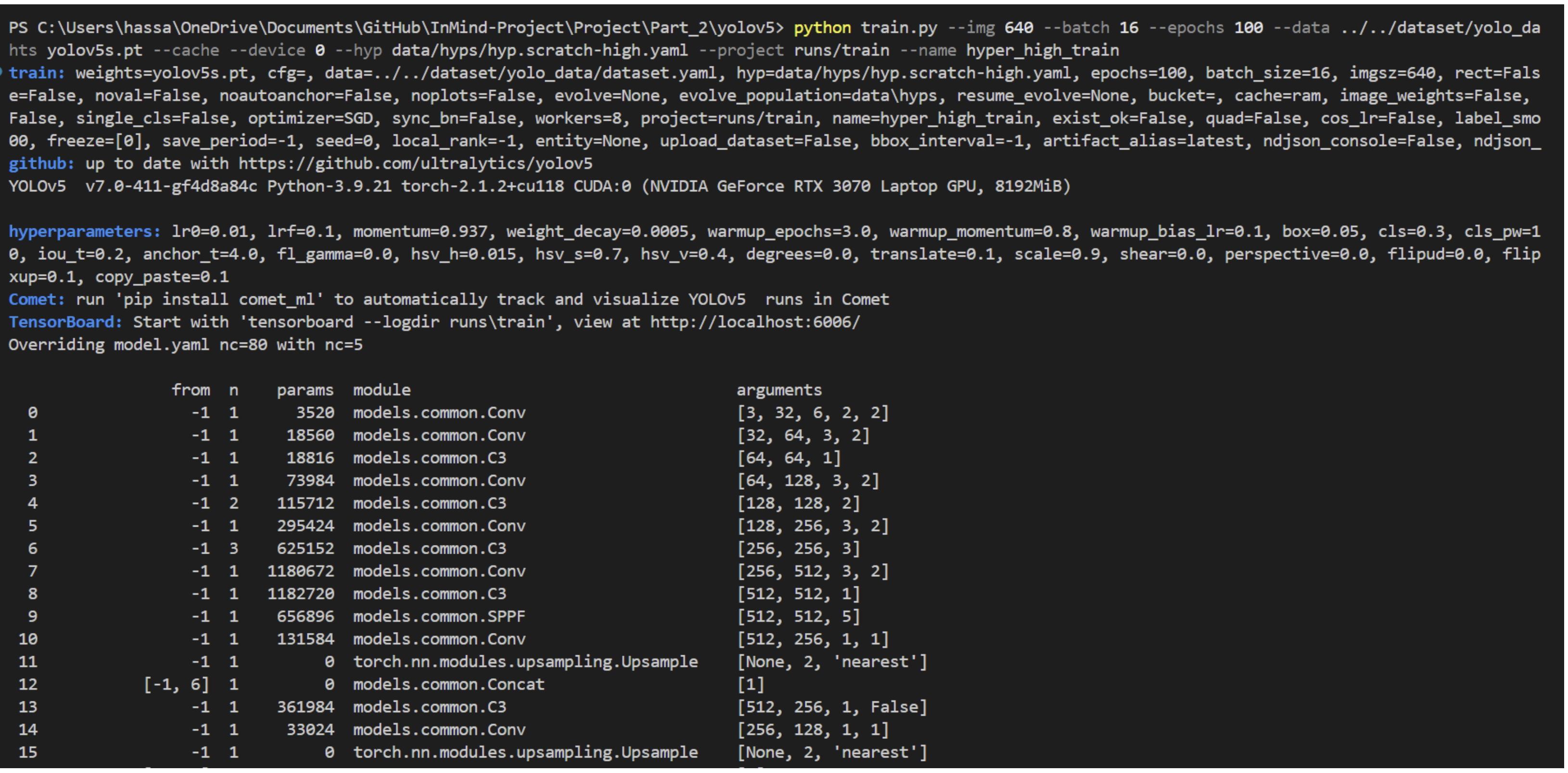
After initial training, the model was evaluated on the test dataset to assess its performance. Based on these results, hyperparameter tuning was performed to potentially improve detection accuracy.



```
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, tas
k=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=ex
p, 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> 
```

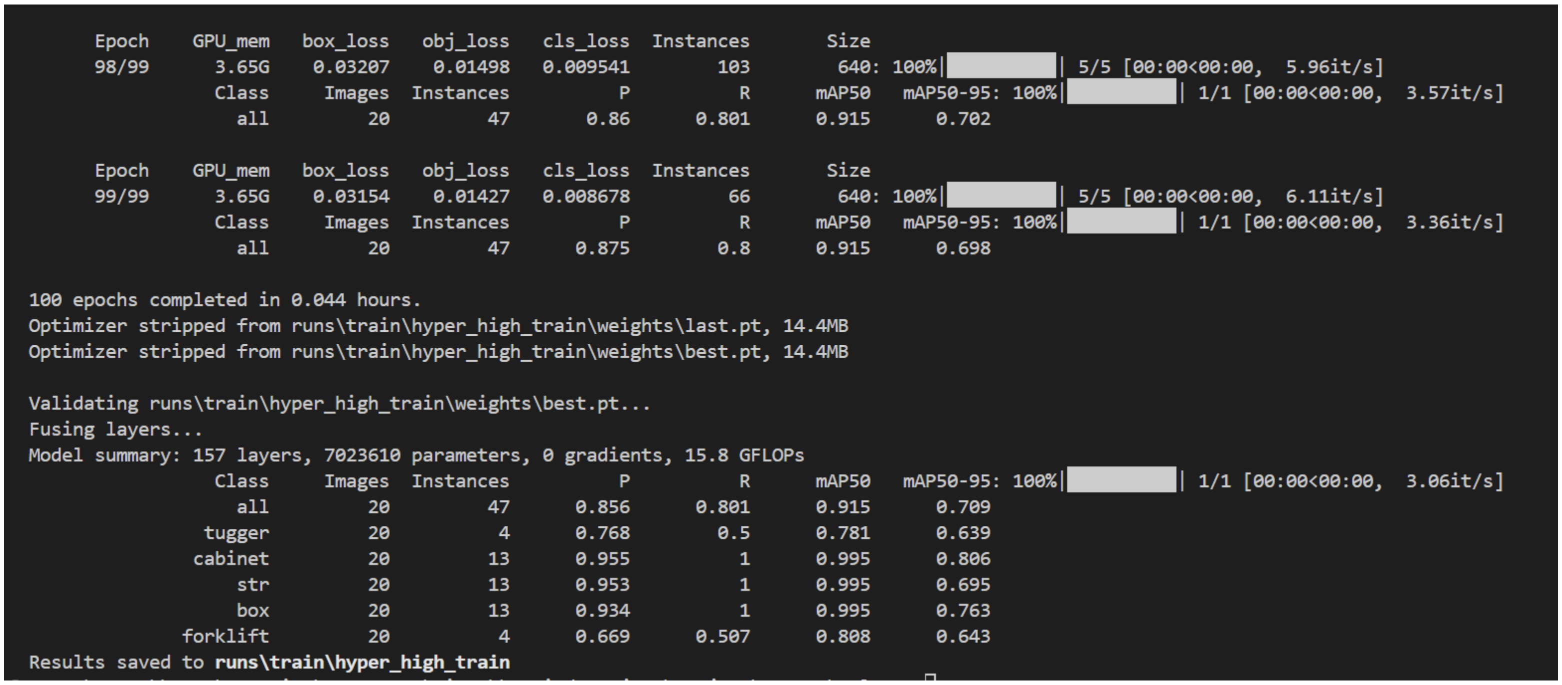
Figure 11: Evaluation results of the trained model on test dataset

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_da
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=Fals
e=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_smo
00, 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:

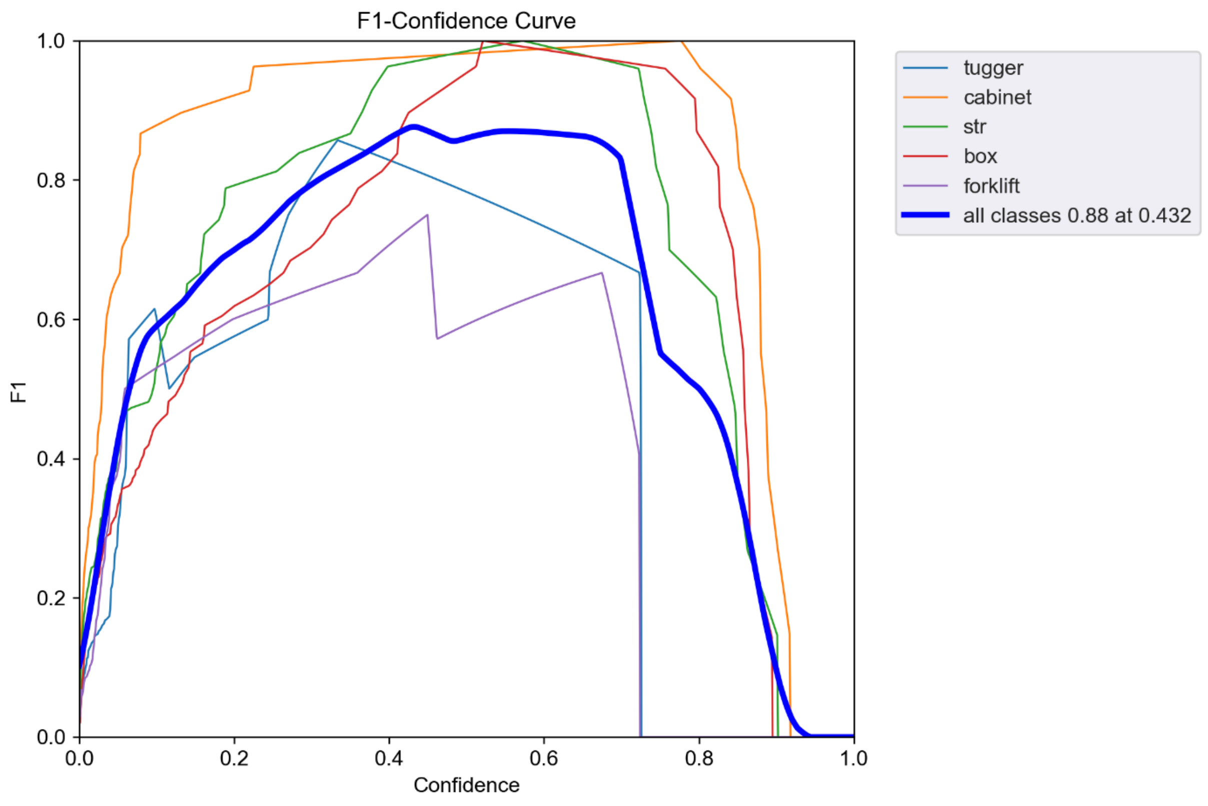


Figure 14: F1 score curve for the standard model

Hyperparameter-tuned model:

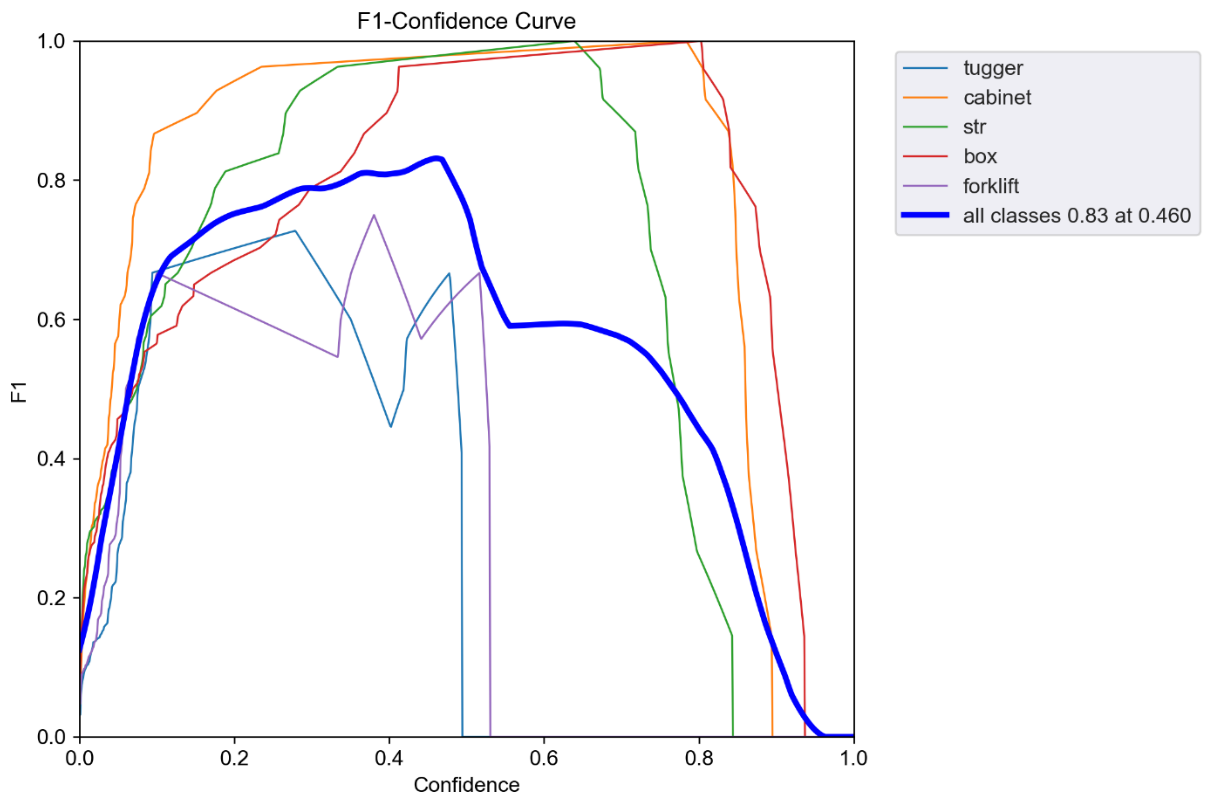
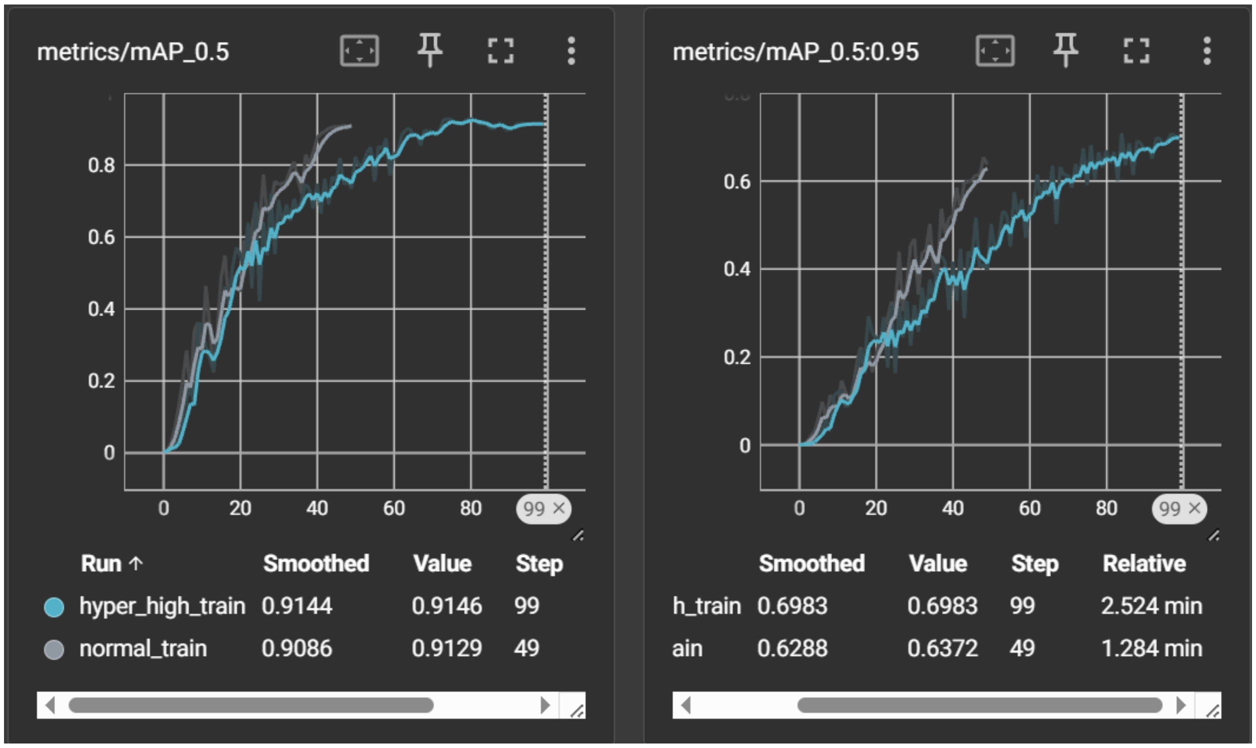


Figure 15: F1 score curve for the hyperparameter-tuned model

c. Mean Average Precision (mAP)



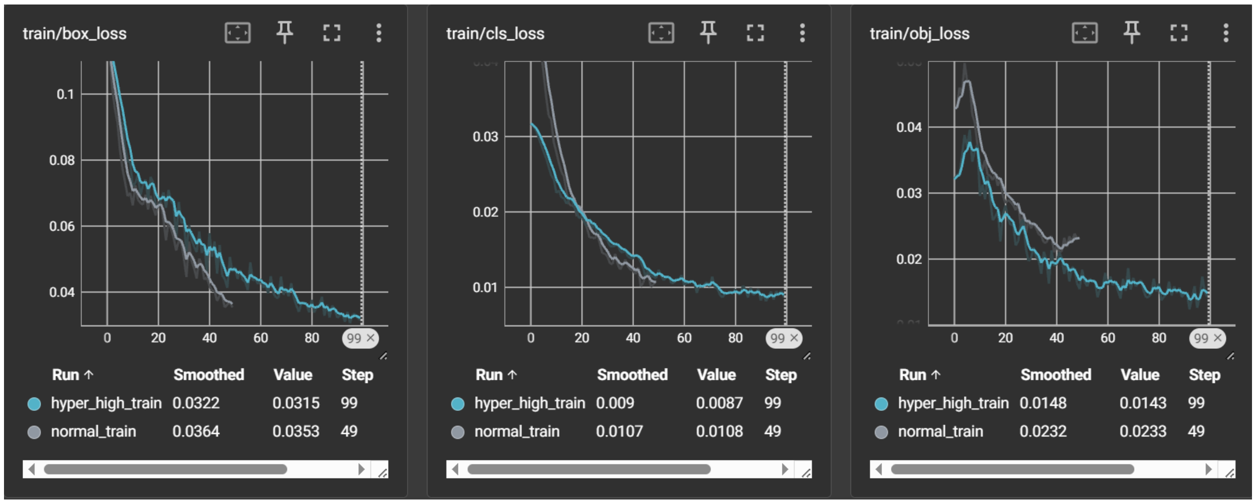
Run ↑	Smoothed	Value	Step
● hyper_high_train	0.9144	0.9146	99
● normal_train	0.9086	0.9129	49

	Smoothed	Value	Step	Relative
h_train	0.6983	0.6983	99	2.524 min
ain	0.6288	0.6372	49	1.284 min

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:



train/box_loss

Run ↑	Smoothed	Value	Step
● hyper_high_train	0.0322	0.0315	99
● normal_train	0.0364	0.0353	49

train/cls_loss

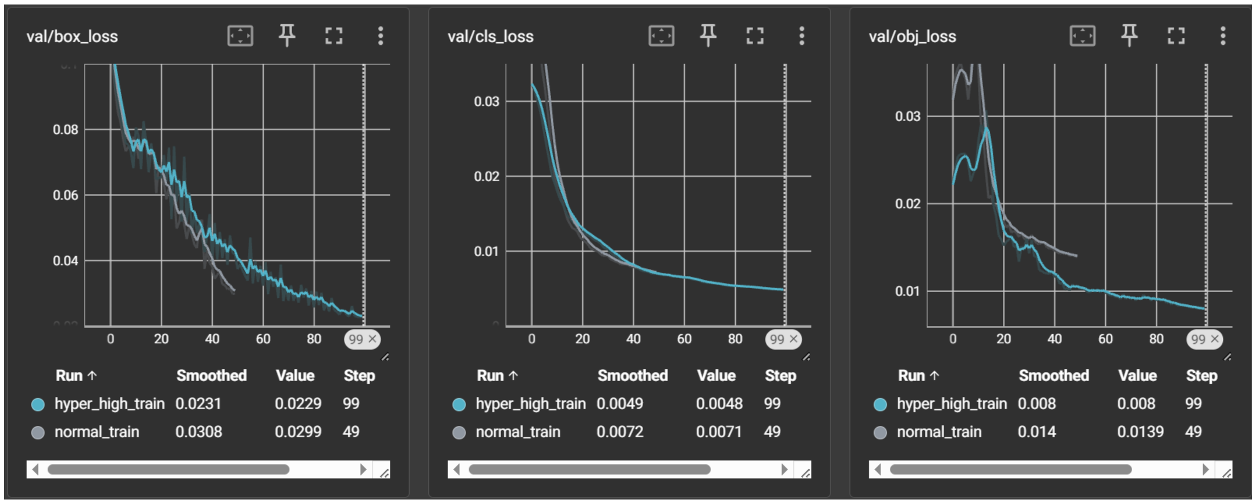
Run ↑	Smoothed	Value	Step
● hyper_high_train	0.009	0.0087	99
● normal_train	0.0107	0.0108	49

train/obj_loss

Run ↑	Smoothed	Value	Step
● hyper_high_train	0.0148	0.0143	99
● normal_train	0.0232	0.0233	49

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

Validation losses for both models:



val/box_loss

Run ↑	Smoothed	Value	Step
● hyper_high_train	0.0231	0.0229	99
● normal_train	0.0308	0.0299	49

val/cls_loss

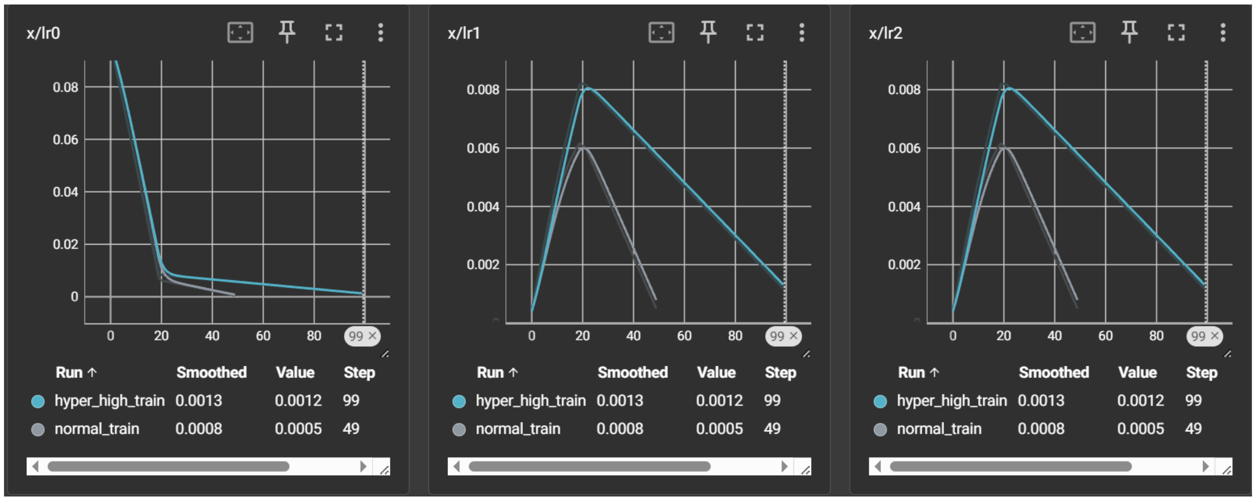
Run ↑	Smoothed	Value	Step
● hyper_high_train	0.0049	0.0048	99
● normal_train	0.0072	0.0071	49

val/obj_loss

Run ↑	Smoothed	Value	Step
● hyper_high_train	0.008	0.008	99
● normal_train	0.014	0.0139	49

Figure 18: Validation losses for both models

c. Learning Rate



x/lr0

Run ↑	Smoothed	Value	Step
● hyper_high_train	0.0013	0.0012	99
● normal_train	0.0008	0.0005	49

x/lr1

Run ↑	Smoothed	Value	Step
● hyper_high_train	0.0013	0.0012	99
● normal_train	0.0008	0.0005	49

x/lr2

Run ↑	Smoothed	Value	Step
● hyper_high_train	0.0013	0.0012	99
● normal_train	0.0008	0.0005	49

Figure 19: Learning rate schedules for both models

d. Confusion Matrices

Standard model:

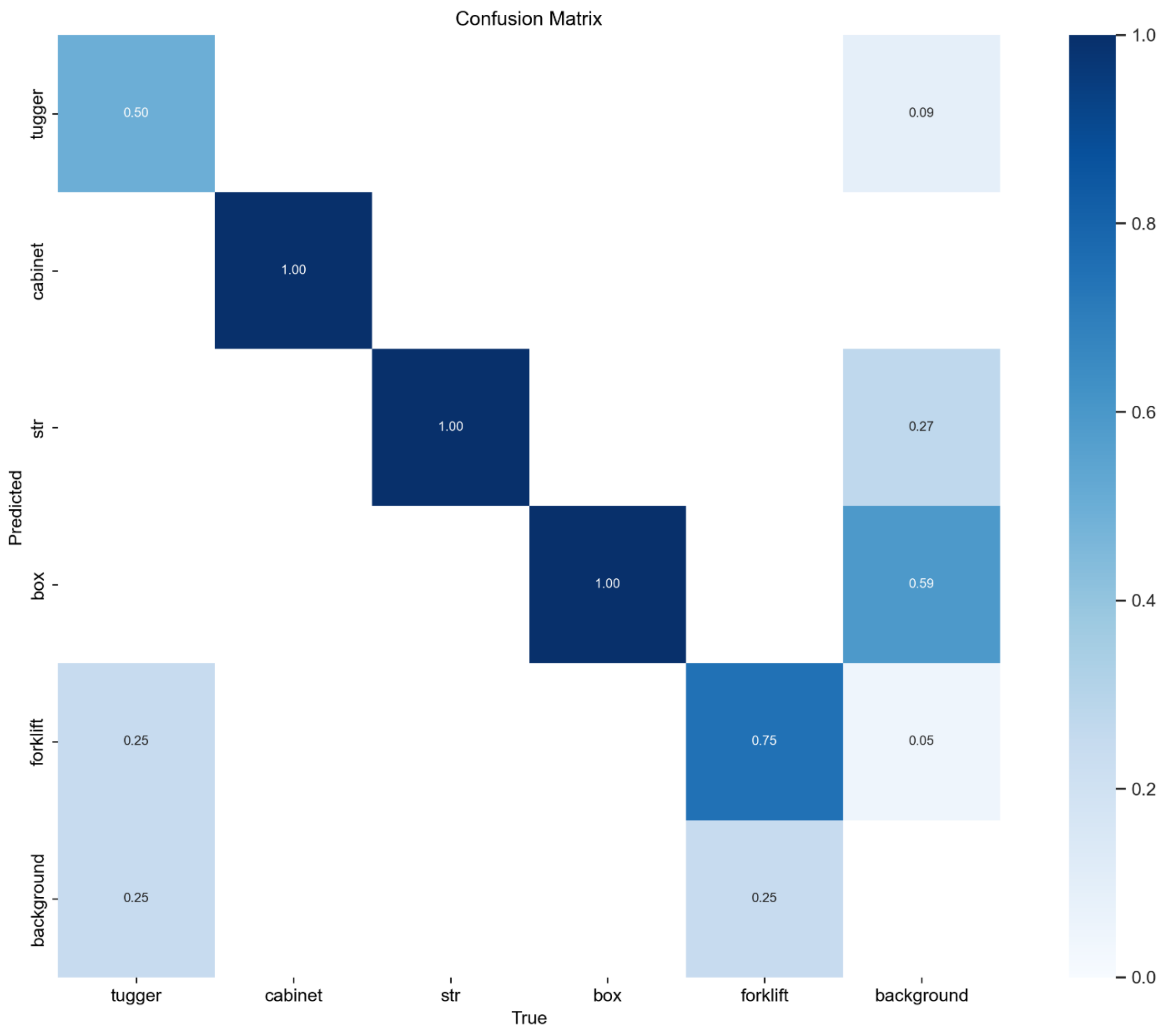


Figure 20: Confusion matrix for the standard model

Hyperparameter-tuned model:

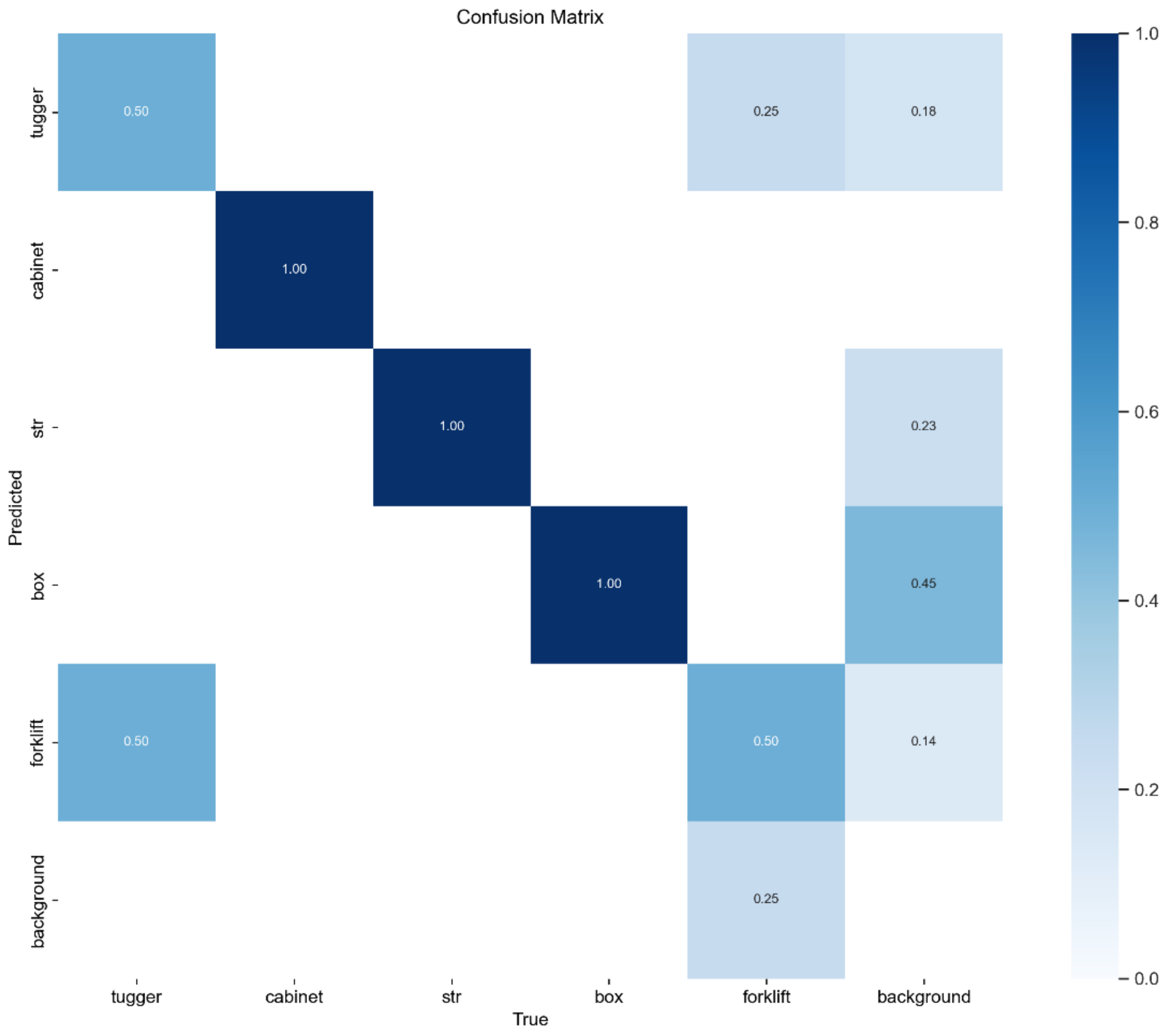


Figure 21: Confusion matrix for the hyperparameter-tuned model

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:

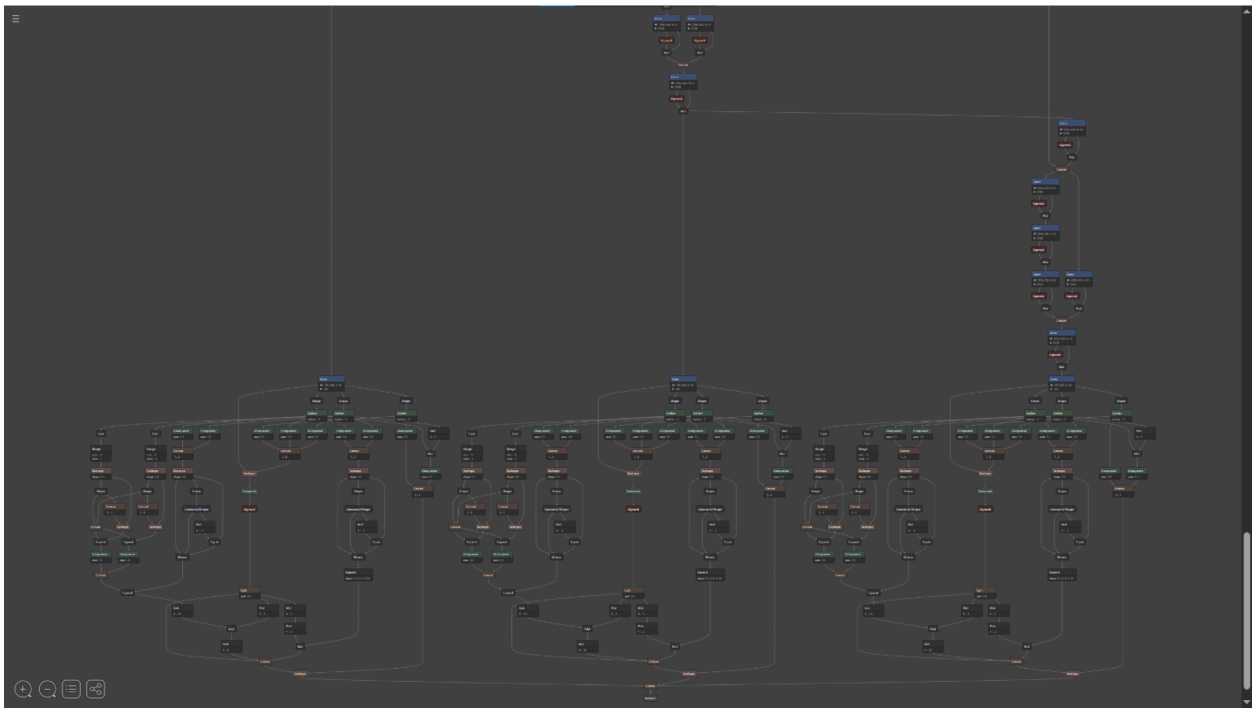


Figure 22: Netron visualization of the standard YOLOv5s model architecture

Hyperparameter-tuned model architecture:

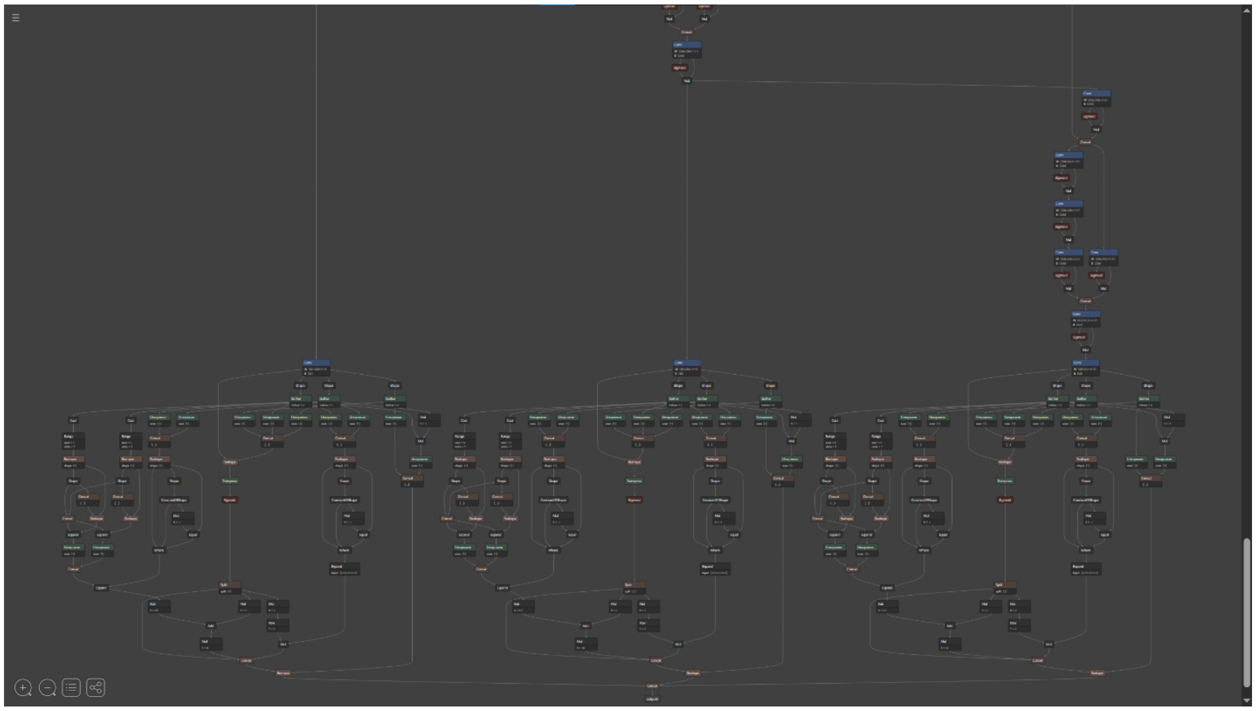


Figure 23: Netron visualization of the hyperparameter-tuned model architecture

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

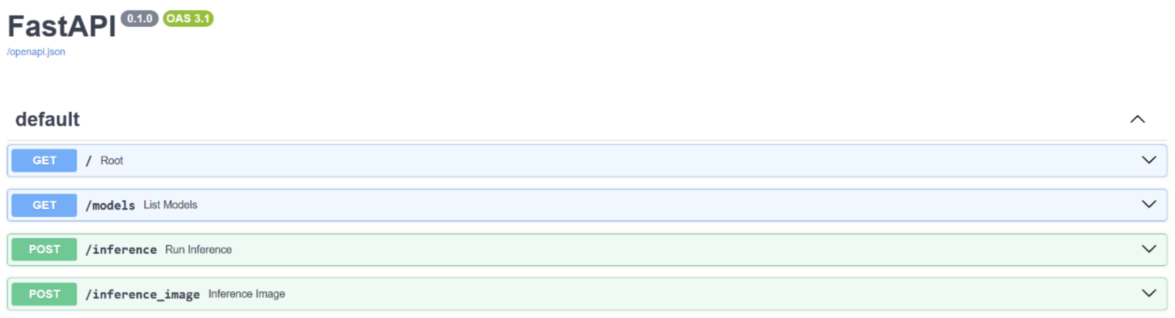
The API implementation includes:

- 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
  }
]
```

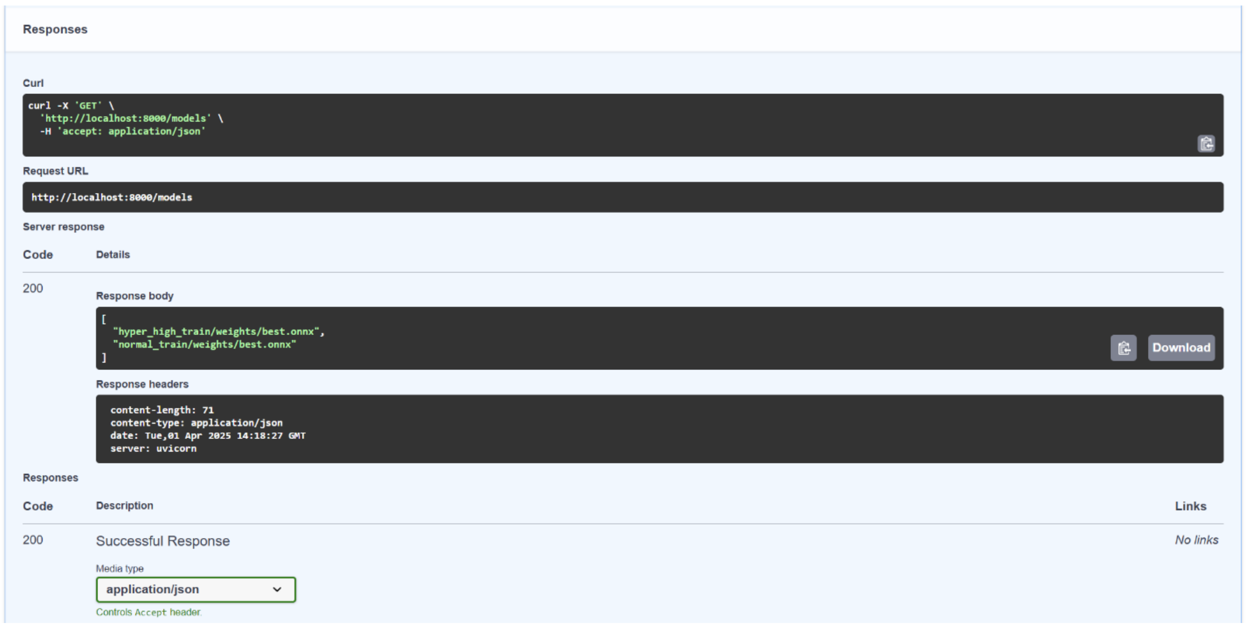
d. Testing FastAPI Endpoints



```
GET   /  Root
GET   /models  List Models
POST  /inference  Run Inference
POST  /inference_image  Inference Image
```

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

Response body

```
[
  "hyper_high_train/weights/best.onnx",
  "normal_train/weights/best.onnx"
]
```

Response headers

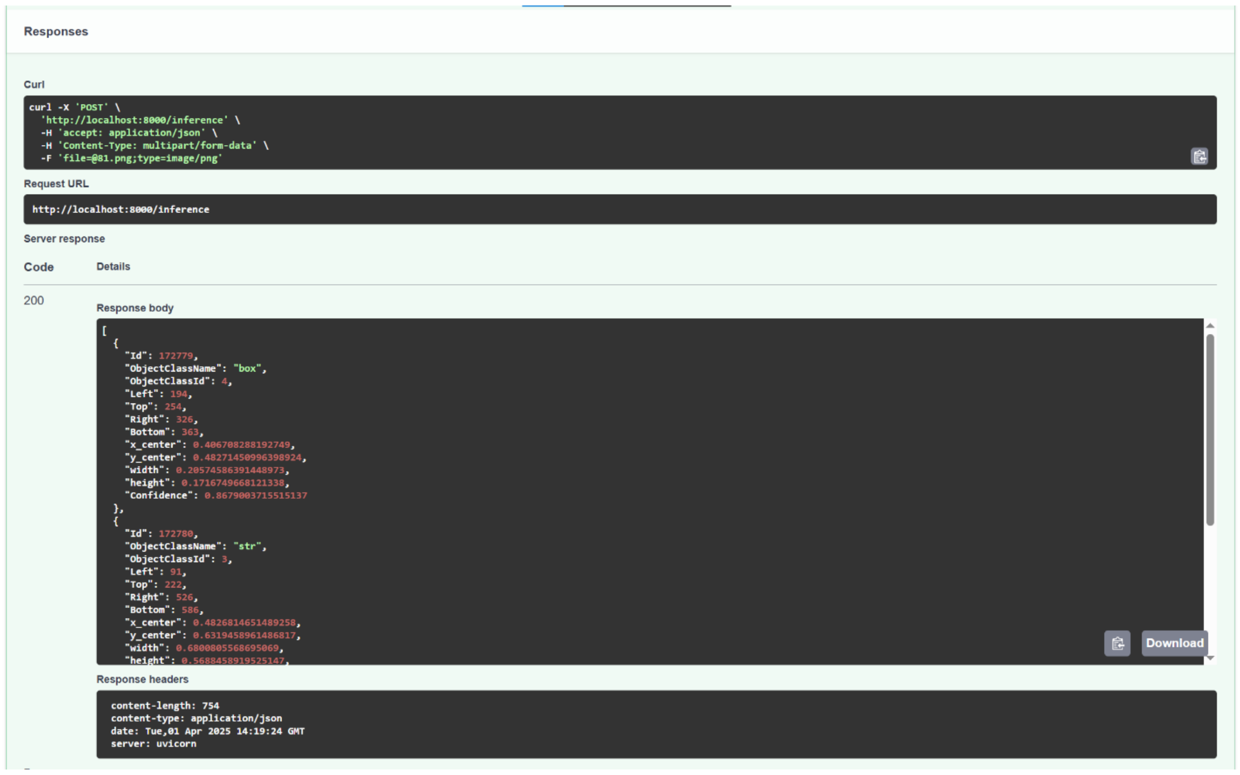
```
 content-length: 71
 content-type: application/json
 date: Tue,01 Apr 2025 14:18:27 GMT
 server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response Media type application/json Controls `Accept` header.	*No links*

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=@81.png;type=image/png'
```

Request URL

```
http://localhost:8000/inference
```

Server response

Code Details

200

Response body

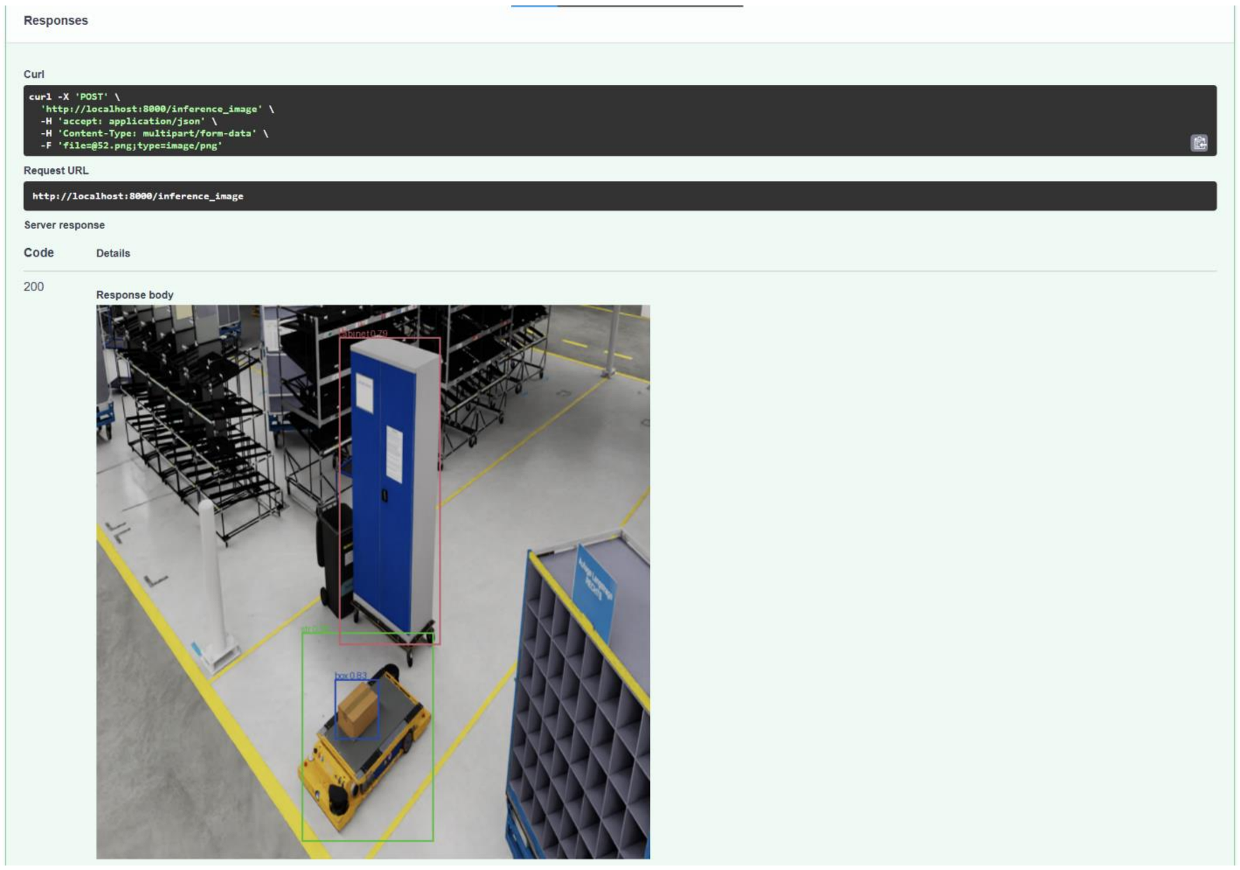
```
[
  {
    "Id": 172779,
    "ObjectClassName": "box",
    "ObjectClassId": 4,
    "Left": 194,
    "Top": 254,
    "Right": 326,
    "Bottom": 363,
    "x_center": 0.406708288192749,
    "y_center": 0.48271450996398924,
    "width": 0.20574586391448973,
    "height": 0.1716749668121338,
    "Confidence": 0.8679003715515137
  },
  {
    "Id": 172780,
    "ObjectClassName": "str",
    "ObjectClassId": 3,
    "Left": 91,
    "Top": 222,
    "Right": 526,
    "Bottom": 586,
    "x_center": 0.4826814651489258,
    "y_center": 0.6319458961486817,
    "width": 0.6800805568695069,
    "height": 0.5688458919525147,
```

Response headers

```
 content-length: 754
 content-type: application/json
 date: Tue,01 Apr 2025 14:19:24 GMT
 server: uvicorn
```

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.

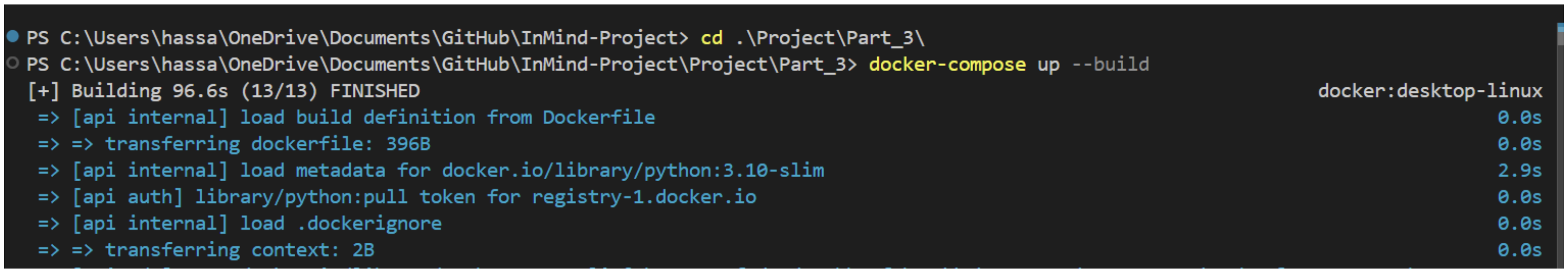
b. Docker Compose

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.

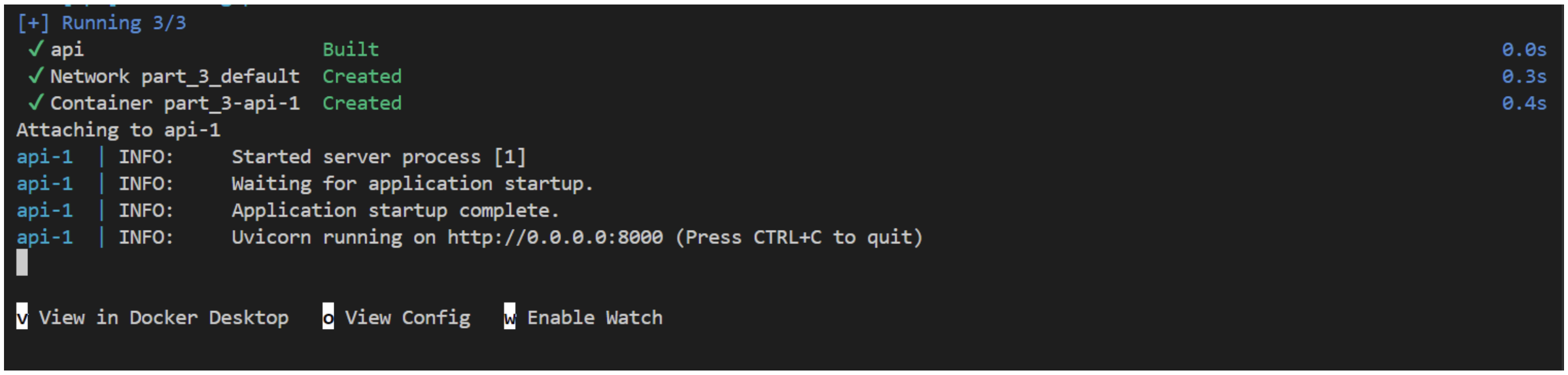
Docker Compose build process initiation:



```
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

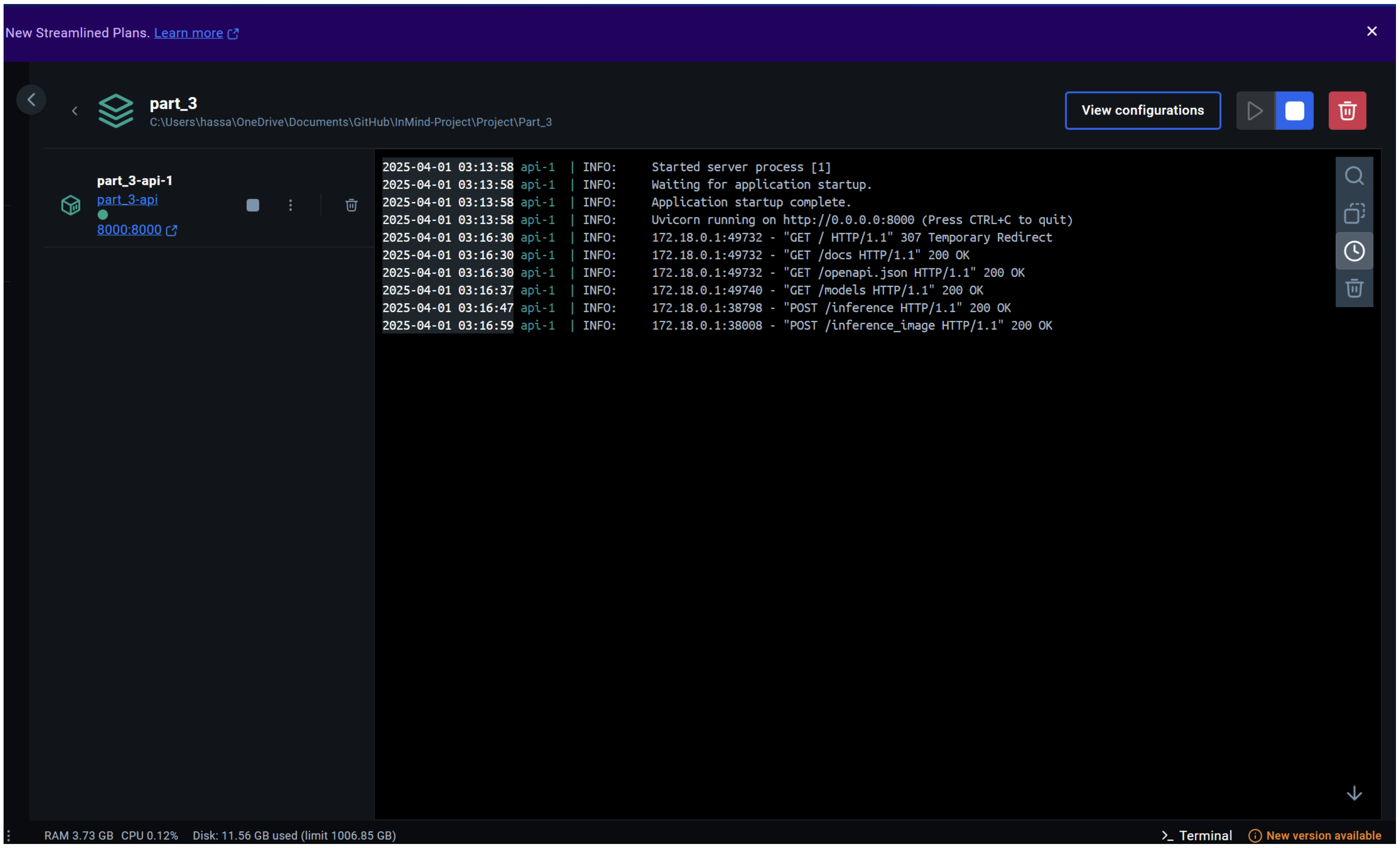
Completion of the Docker container build:



```
[+] 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)

v 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:



New Streamlined Plans. Learn more
part_3
C:\Users\hassa\OneDrive\Documents\GitHub\InMind-Project\Project\Part_3
View configurations
part_3-api-1
part_3-api
8000:8000
RAM 3.73 GB CPU 0.12% Disk: 11.56 GB used (limit 1006.85 GB)
Terminal
New version available

```
2025-04-01 03:13:58 api-1  | INFO:     Started server process [1]
2025-04-01 03:13:58 api-1  | INFO:     Waiting for application startup.
2025-04-01 03:13:58 api-1  | INFO:     Application startup complete.
2025-04-01 03:13:58 api-1  | INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
2025-04-01 03:16:30 api-1  | INFO:     172.18.0.1:49732 - "GET / HTTP/1.1" 307 Temporary Redirect
2025-04-01 03:16:30 api-1  | INFO:     172.18.0.1:49732 - "GET /docs HTTP/1.1" 200 OK
2025-04-01 03:16:30 api-1  | INFO:     172.18.0.1:49732 - "GET /openapi.json HTTP/1.1" 200 OK
2025-04-01 03:16:37 api-1  | INFO:     172.18.0.1:49740 - "GET /models HTTP/1.1" 200 OK
2025-04-01 03:16:47 api-1  | INFO:     172.18.0.1:38798 - "POST /inference HTTP/1.1" 200 OK
2025-04-01 03:16:59 api-1  | INFO:     172.18.0.1:38008 - "POST /inference_image HTTP/1.1" 200 OK
```

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.