# Computer Vision: Object Detection and Classification in Warehouses using a Depth Camera on RTOS

Studienarbeit

**Chair for Embedded Systems**
**University of Siegen**

Author: Gopi Ravi Teja Gudupu
Mat.-No: 1710623
Supervisor: M.Sc. Aniebiet Micheal Ezekiel

# Abstract

This project develops a refined system for detecting and classifying objects in warehouse settings using depth cameras, which provide valuable 3D information on object dimensions and positions. Unlike traditional 2D cameras, depth cameras offer enhanced accuracy in detecting and categorizing objects amidst the complexity of varied shapes, sizes, and orientations commonly found in warehouses. By integrating this 3D data with visual information, the system improves the recognition of objects that may appear similar but differ in spatial characteristics, thus advancing automation and inventory management. The project features a multi-stage process for environment segmentation, object detection, and classification supported by machine learning models. Key algorithms explored include YOLO (You Only Look Once), known for its real-time processing capabilities. This report evaluates the effectiveness of these algorithms through metrics such as precision, recall, and mean Average Precision (mAP) while addressing challenges such as lighting conditions and computational constraints.

# List of Figures

# Contents

# 1 Introduction

This project aims to develop a system for detecting and classifying objects in a warehouse using a depth camera. Warehouses are complex environments with diverse objects of varying sizes, shapes, and orientations. Effective object detection and classification are essential for inventory management, automation, and optimizing logistics. Depth cameras offer an advantage over traditional 2D cameras by providing 3D information about objects, including their spatial positions and physical characteristics like depth, volume, and contours. This additional data dimension enhances the accuracy and precision of object detection, particularly in cluttered or dynamic warehouse settings.

By leveraging the depth camera's 3D sensing capabilities, the system can identify and classify objects based on both visual and spatial data, improving recognition of objects that might be similar in appearance but different in size or position. This technology can be integrated into warehouse automation systems, enabling real-time tracking, sorting, and organization of products. The proposed solution is designed to significantly enhance the efficiency and accuracy of warehouse operations, with a key focus on reducing human error and optimizing resource management. By implementing depth cameras for object detection and classification in warehouse environments, we can expect to see improved workflows, streamlined inventory control, and more effective automation systems, thereby revolutionizing warehouse operations.

## 1.1 Overview of warehouse automation and its significance

Image processing with different data from the cameras used from various perspectives (multi-aperture approach) is merged. Sens fusion methods such as an extended Kalman filter or particle filter are used and further developed to obtain coherent information. Additionally, developing and implementing a multi-stage process consisting of environment segmentation and the recognition and reconstruction of packets in a warehouse are undertaken. An algorithm for object representation for environment segmentation, detection, and classification of packets based on machine learning models is formulated. These methods are developed, implemented, and iteratively improved in a simulation environment to parallelise learning episodes and on the test bench. In WP5, this data is primarily used for grasp planning of the classified objects

## 1.2 Importance of carton box detection and classification in improving efficiency

Carton detection and classification involve advanced computer vision techniques that leverage machine learning and deep learning models. Traditional object detection methods have evolved into sophisticated algorithms capable of accurately identifying objects in real time. With the advent of technologies like convolutional neural networks (CNNs), warehouses can now detect boxes of different shapes, sizes, and positions with high precision. The introduction of algorithms such as YOLO (You Only Look Once) and Single Shot Detection (SSD) has enabled faster and more efficient detection, making them suitable for real-time warehouse applications.

## 1.3 Objectives of the report

This report explores the methods and processes employed in detecting and classifying carton boxes within a warehouse setting. We focus on the effectiveness of various detection algorithms, including two-stage networks and single-shot detection methods. By training a deep learning model using a stacked carton dataset, this report evaluates the model's performance through metrics such as precision, recall, and mean Average Precision (mAP).

In addition to discussing the algorithms, we will address the challenges faced during the detection and classification process, such as lighting conditions, overfitting, and computational limitations. Furthermore, the report delves into the practical integration of these models in real-world warehouse scenarios, outlining the observations and future improvements that can be made. Ultimately, this study aims to provide a roadmap for developing an efficient system for carton detection that can be scaled across various warehouse environments, reassuring the audience of the practicality of these methods.

# 2 Detection Algorithm

Accurate detection of carton boxes in warehouses depends heavily on the choice of detection algorithm. Over the years, object detection techniques have evolved into two primary categories: two-stage networks and single-shot detection (SSD). Each approach offers distinct advantages and trade-offs.

## 2.1 Two-Stage Networks

Two-stage networks, such as R-CNN (Regions with Convolutional Neural Networks) and Fast R-CNN, operate by splitting the object detection process into two distinct phases: region selection and region classification. In the first stage, potential object regions are identified, and in the second stage, these regions are refined and classified. This approach provides high accuracy but often comes at the cost of computational speed, making it less suitable for real-time applications in high-speed environments like warehouses. However, the accuracy of two-stage networks is notable, especially in cluttered environments with overlapping or occluded objects, such as stacked carton boxes. Models like Faster R-CNN have been instrumental in improving speed without sacrificing much accuracy, making them valuable in research-based environments.

## 2.2 Single Shot Detection (SSD)

Single-shot detection methods like YOLO (You Only Look Once) and RetinaNet address the limitations of two-stage networks by combining both object classification and localization into a single forward pass through the network. This significantly reduces processing time, making SSD methods more suitable for real-time applications, such as monitoring carton boxes in dynamic warehouse settings. YOLO, for instance, divides an image into an SxS grid, predicting bounding boxes and class probabilities for each cell. Its fast computation makes it ideal for detecting objects such as cartons in varying lighting conditions, angles, and arrangements.

The architecture network has 24 convolutional layers followed by 2 fully connected layers. Alternating $1\times1$ convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the imagenet classification task at half the resolution ( $224\times224$ input image) and then double the resolution for detection.

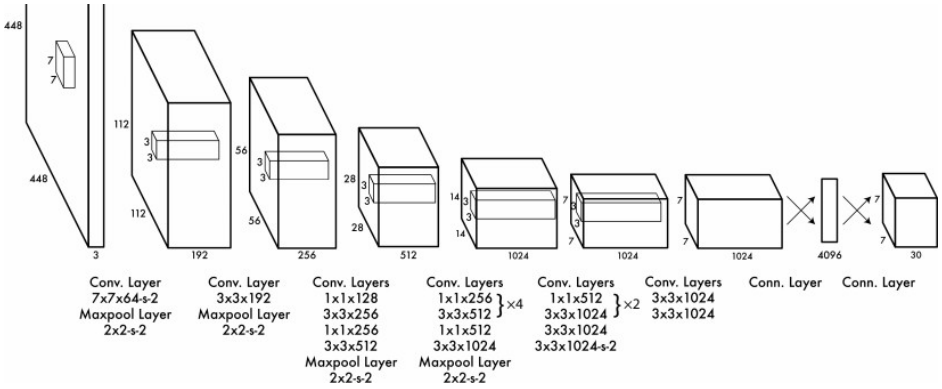


Figure 1: The architecture network of YOLO

## 2.3 YOLO Algorithm for Carton Detection

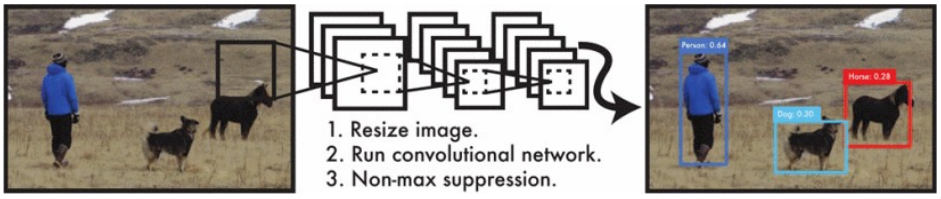


Figure 2: The YOLO Detection System

The YOLO algorithm is one of the most popular SSD models used in object detection for warehouses. It processes the entire image at once, predicting multiple bounding boxes and their associated class probabilities simultaneously. The Intersection over Union (IoU) metric, used in YOLO, measures the overlap between predicted bounding boxes and ground truth boxes. A higher IoU score indicates more accurate object detection, which is crucial when distinguishing between tightly packed or stacked carton boxes. The algorithm's real-time capability and high precision make it an ideal choice for detecting and classifying cartons in a fast-paced warehouse environment.

The model divides the image into an S×S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S×S×(B*5+C) tensor.

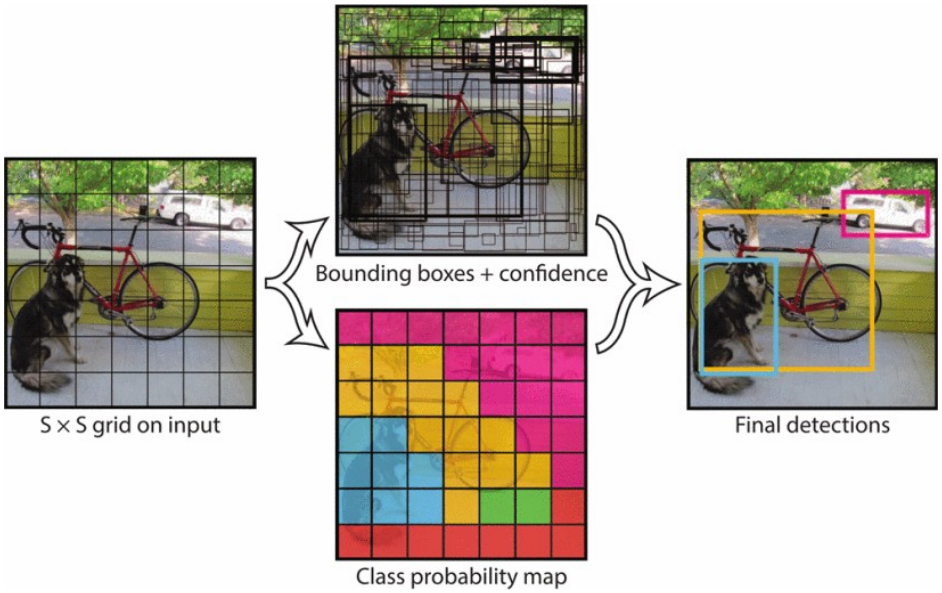


Figure 3: The working model of YOLO

# 3 Methodology

The process of detecting and classifying carton boxes using deep learning models follows a structured workflow, beginning with data preparation and ending with model evaluation. This section outlines the critical steps involved in implementing an effective detection system for carton boxes in warehouses.

## 3.1 Image Collection and Labeling

The first step in training a carton detection model is to collect a diverse dataset of images that represent the warehouse environment. These images must include various scenarios such as different lighting conditions, box sizes, orientations, and stacked configurations. Once the images are gathered, the next step is labeling them using annotation tools. Labeling involves assigning bounding boxes to each carton box in the image and categorizing them according to predefined classes. The accuracy of this manual labeling process is crucial since it directly influences the training quality of the model.


Figure 4: Stock Image


Figure 5: Labelled Image

## 3.2 Model Training

After collecting and labeling the data, the model training process begins. Typically, pre-trained weights from a model such as YOLOv8 are used as a foundation for training, allowing faster convergence and better accuracy. The training is conducted on a labeled dataset, such as the Stacked Carton Dataset (SDC), which consists of numerous images depicting different configurations of carton boxes. During training, the model learns to predict bounding boxes and classify the objects within the image. For the YOLOv8 model, training on 60 epochs provides a comprehensive understanding of the objects in the dataset, allowing for accurate detection in unseen environments.

## 3.3 Evaluation

Once the model is trained, it is evaluated on a separate validation set to measure its performance. Key evaluation metrics include precision, recall, and mean Average Precision (mAP). Precision refers to the model's accuracy in correctly identifying carton boxes, while recall measures how well it detects all relevant instances. The mAP metric is used to evaluate the performance across different IoU thresholds, with higher values indicating better model performance. In the case of the carton detection system, the model achieved an mAP of 94.9%, showing high accuracy. However, the confusion matrix

revealed some misclassifications, such as background regions being classified as carton boxes, which indicates the need for further refinement.

The results of this process provide essential insights into the model's strengths and areas for improvement, which can be addressed through fine-tuning and additional training.

## 3.4 Challenges

Detecting and classifying carton boxes in warehouses is not without its challenges. While object detection algorithms have advanced significantly, specific obstacles related to the warehouse environment can still hinder their performance. These challenges often stem from environmental conditions, computational resources, and dataset-related issues.

### 3.4.1 Detection Challenges

One of the major challenges in warehouse environments is the variability in lighting conditions. Warehouses may have inconsistent or poor lighting, which can significantly affect the ability of models like YOLO to detect objects accurately. Shadows, reflections, and glare can cause the model to misinterpret the location and edges of carton boxes. Noise from the sensor, such as camera distortions or interference from nearby equipment, can also lead to inaccuracies in detection.

Another common challenge is overfitting, particularly when training models on a small or unbalanced dataset. Overfitting occurs when the model performs well on the training data but fails to generalize to unseen environments, such as different angles or configurations of carton boxes. This issue is exacerbated in scenarios where boxes are stacked, as the model may fail to differentiate between individual boxes and recognize the entire stack as a single object.

### 3.4.2 Training Challenges

Training deep learning models for carton detection requires significant computational resources. For example, training on a CPU is incredibly time-consuming, especially when using large datasets like the SDC. In one case, training a YOLOv8 model on approximately 9,400 images took over 10 hours per epoch on a CPU, whereas the same task took only 50 minutes on a GPU. Additionally, issues related to version compatibility with dependencies such as CUDA and torch can delay training processes and complicate the deployment of models in real-time applications.

Transitioning to a cluster GPU environment presents its own set of challenges, such as managing dependencies and optimizing the model for faster performance. GPU resources are often limited in availability, meaning that delays may arise when multiple models or experiments are running simultaneously. Balancing dataset size and training time is another challenge. Large datasets improve model accuracy but significantly increase training time, while smaller datasets can lead to underfitting.


Figure 6: Performance Comparison

**10 hours on CPU vs 50 Minutes on GPU is drastic improvement in terms of time consumption for the same YOLO model, same dataset, same image size .

# 4 Experiments

## 4.1 Datasets and Training Experiments

### 4.1.1 Explored Datasets

Several datasets were considered for carton detection. The CV-Bucharest dataset from Roboflow, consisting of 437 images, was the first dataset explored. Its primary strength was its ability to detect single carton boxes with moderate accuracy, but it lacked the ability to handle stacked boxes, which is common in warehouse environments. Another dataset, Carton Detection TAD, also focused on single box detection but incorporated semantic segmentation techniques, offering slightly better accuracy.

The Online Stacked Carton Dataset (OSCD), with 8,401 images, was one of the primary datasets used for this project. It featured stacked boxes, a key challenge in warehouse environments, but was prone to overfitting, as the model often identified any rectangular shape as a carton box. Live Stacked Carton Dataset (LSCD) was another dataset with a similar configuration, containing 7,735 images. The model also faced overfitting issues with this dataset, especially in stacked box detection.

The MASON dataset, sourced from the DMK Images Archive, contained 400 images. It was primarily used for testing single box detection models. While this dataset performed well in detecting single boxes, its performance in detecting stacked boxes was less accurate, limiting its utility for warehouse environments where stacked boxes are common.

| S.No | Dataset Name | Source | Number of Images | Strength | Limitation | Reason to choose |
|---|---|---|---|---|---|---|
| 1 | CV_Bucharest | Roboflow | 437 | Single box Detection | Stacked box, Less Accurate | Test |
| 2 | Carton Detection TAD | Roboflow | 626 | Single box Detection, Accuracy | Stacked box | Semantic Segmentation |
| 3 | Online Stacked Carton Dataset (OSCD) | Research Journal | 8401 | Stacked box | Overfitting | Segmentation |
| 4 | Live Stacked Carton Dataset (LSCD) | Research Journal | 7735 | Stacked box | Overfitting | Segmentation |
| 5 | Dataset from MASON | DMK Images Archive | 400 | Single box Detection | Stacked box, Less Accurate | Test |

Figure 7: The Datasets Explored

### 4.1.2 Training Experiments

Several training experiments were conducted using different YOLO models and datasets. One experiment involved training the YOLOv8 model on the CV-Bucharest dataset for 25 epochs, resulting in a mean Average Precision

(mAP) of 71.69%. However, the detection speed was slow, and the accuracy for stacked boxes was low.

Another experiment trained the YOLOv8 model on the OSCD dataset for five epochs, achieving an mAP of 75.27%. The model faced severe overfitting issues, where it mistakenly identified any rectangular shape as a carton box, reducing its effectiveness. To address this, the dataset was combined with the CV-Bucharest dataset, and the model was trained for 60 epochs. This improved the mAP to 81.19%, but overfitting persisted.

In another attempt, the Carton Detection TAD dataset was added to the training process, and the YOLOv8 medium model was used, which provided a more robust training configuration. After 180 epochs, the model achieved an mAP of 83.95%. Despite improvements in detection speed and accuracy, the overfitting issue remained, particularly when detecting stacked boxes.

These experiments highlighted the complexities of training models in environments where boxes are stacked or arranged in various configurations. Additional steps, such as data augmentation and balanced datasets, are needed to further improve the model's ability to generalize across different scenarios.

Training Experiments Performed

| S.No | Dataset | Number of Images | YOLO Model | Epochs | mAP (50-95)* | Performance |
|---|---|---|---|---|---|---|
| 1 | CV_Bucharest | 437 | Yolo V8 Small | 25 | 71.69% | Slow Detection, Less Accuracy |
| 2 | Online Stacked Carton Dataset (OSCD) | 8401 | Yolo V8 Small | 5 | 75.27% | Slow Detection, Less Accuracy, Overfitting(Identified any rectangular shape as carton box) |
| 3 | Online Stacked Carton Dataset (OSCD) + CV_Bucharest | 8838 | Yolo V8 Small | 60 | 81.19% | Quick Detection, Very Accurate, Overfitting(Identified any rectangular shape as carton box) |
| 4 | Online Stacked Carton Dataset (OSCD) + CV_Bucharest + Carton Detection TAD | 9464 | Yolo V8 Medium | 180 | 83.95% | Quick Detection, Severe Overfitting(Identified any rectangular shape as carton box) |

Figure 8: Training Experiments Performed

## 4.2 Comparison with Grey scaled images training

Specific dataset of 257 images from Mason achieves are taken and are Greyscaled with the help of data augmentation and trained for 50 epochs to compare the performance with exactly same images without greyscaling.

### 4.2.1 Sample Greyscaling

The same exact 257 images are greyscaled. Below images shows the comparison of the same


Figure 9: Sample Image Without Greyscaling

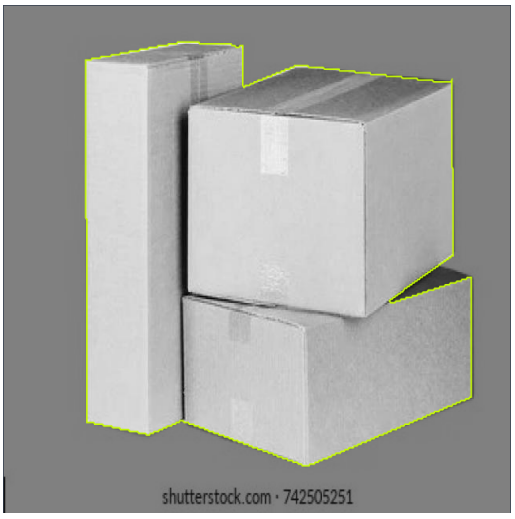


Figure 10: Sample Image With Greyscaling

### 4.2.2 Colored Images Results

The confusion matrix and the results for the images trained without greyscaling are shown below. After 50 epochs, the highest mAP50 achieved is 44.925% and mAP50-90 of 31.289% was achieved.

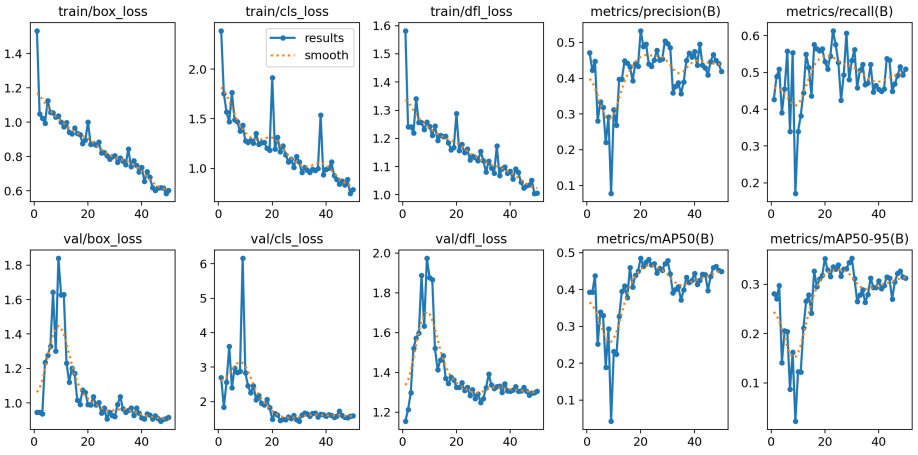


Figure 11: The Results after Training of Colored Images

| epoch | train/box_loss | train/cls_loss | train/dfl_loss | metrics/precision(B) | metrics/recall(B) | metrics/mAP50(B) | metrics/mAP50-95(B) |
|---|---|---|---|---|---|---|---|
| 50 | 0.60272 | 0.78598 | 1.0062 | 0.41909 | 0.50889 | 0.44925 | 0.31289 |

Figure 12: The Results after Training of Colored Images

### 4.2.3 Greyscaled Images Results

The confusion matrix and the results for the images trained with greyscaling are shown below. After 50 epochs, the highest mAP50 achieved is 47.795% and mAP50-90 of 33.506% was achieved.

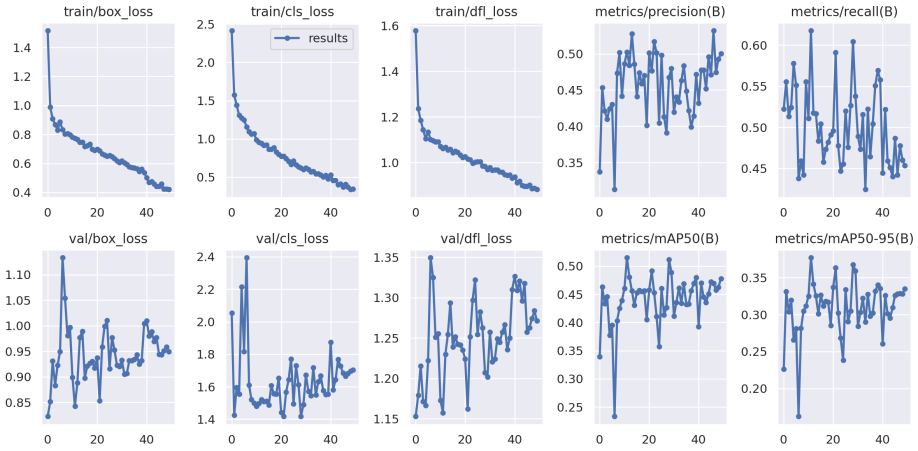


Figure 13: The Results after Training of Greyscaled Images

| epoch | train/box_loss | train/cls_loss | train/dfl_loss | metrics/precision(B) | metrics/recall(B) | metrics/mAP50(B) | metrics/mAP50-95(B) |
|---|---|---|---|---|---|---|---|
| 50 | 0.42056 | 0.3474 | 0.8818 | 0.50002 | 0.45338 | 0.47795 | 0.33506 |

Figure 14: The Results after Training of Greyscaled Images

### 4.2.4 Comparison of Results

The comparison of the confusion matrix is shown below.

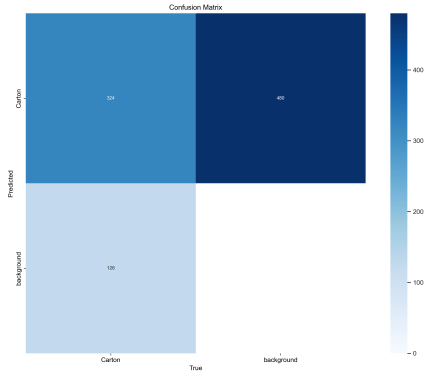


Figure 15: The Confusion Matrix Without Greyscaling

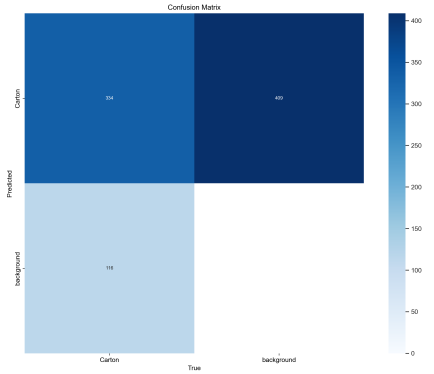


Figure 16: The Confusion Matrix With Greyscaling

The model trained without greyscaling performs well in all three cases(Single carton in the image, 2 Cartons in the image, and Stacked cartons). Also, the model trained without greyscaling seems to overfit in the image with only one carton(as the wall with a yellowish colour is also detected as a carton)

The model trained with greyscaling fails to detect cartons in images with stacked cartons and when there are two cartons in the image.


Figure 17: Detection with 1 Carton in the image (Without Greyscaling)


Figure 18: Detection with 1 Carton in the image (With Greyscaling)


Figure 19: Detection with Cartons stacked in the image (Without Greyscaling)


Figure 20: Detection with Cartons stacked in the image (With Greyscaling)


Figure 21: Detection with 2 Cartons in the image (Without Greyscaling)


Figure 22: Detection with 2 Cartons in the image (With Greyscaling)

## 4.3 Experiment on SCD: A Stacked Carton Dataset for Detection and Segmentation

A total of 8630 images with 37755 instances of cartons in the pictures are trained for 60 Epochs.
The Pictures are split in a proportion of 70% for Training, 20% for Validation, and 10% for Testing:
Training Split = 6123 images
Validation Split = 1670 images
Test Split = 837 images

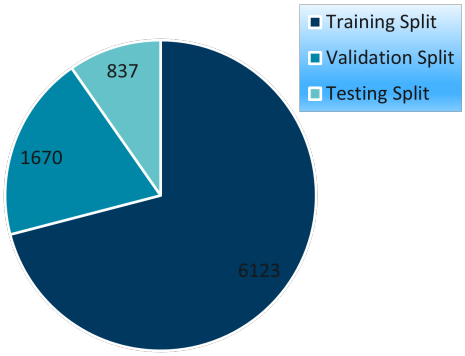


Figure 23: Image Split for Training

The confusion matrix shows that there were 6315 instances where the background was identified as Carton boxes

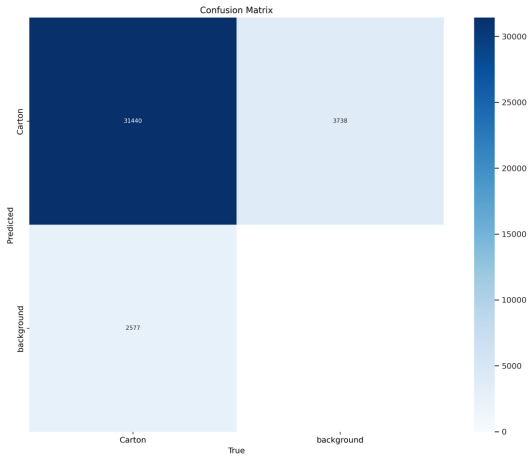


Figure 24: Confusion Matrix

The box loss and the class loss graphs indicate there is still scope for improvement as the loss graphs are still declining, and the precision graphs are still rising.
Precision - 94.3%
Recall – 88.8%
mAP50 – 94.9%
mAP50-90 – 81.2%

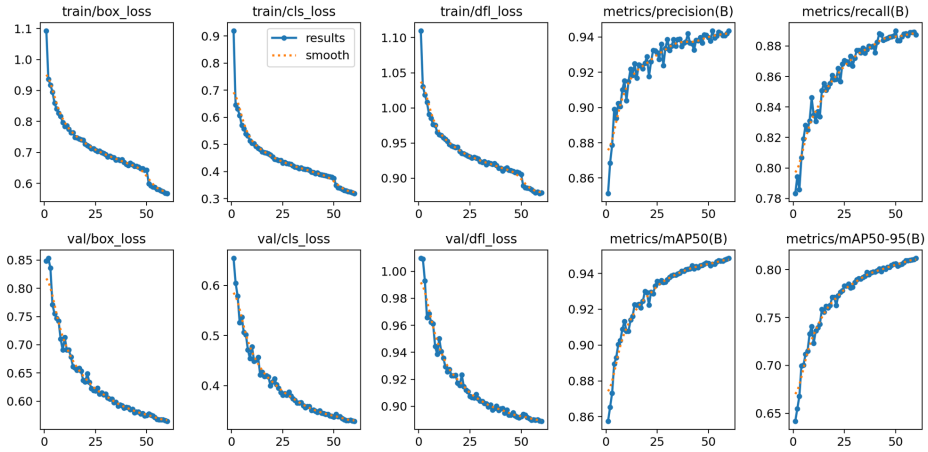


Figure 25: Results

# 5 Findings

## 5.1 Training with custom dataset gives better results

Research proves the ability of YOLO v8 architecture to detect motorbike objects. The trained model of the YOLO was trained using a custom dataset that contains 15.000 data and divided into 9 classes. The trained version of YOLO v8 model shows a promising result that is proven by its ability to detects motorbike more accurately rather than a pre-trained version of the YOLO v8 model [1]

## 5.2 YOLO v8 yields best results with a custom dataset

Another research proves Pre-trained Faster R-CNN and different versions of YOLOv5 and YOLOv8 were used and fine-tuned on our dataset, respectively, among which YOLOv8x achieved the best results on the dataset, with the mAP up to 94.2% [2]

## 5.3 Factors affecting Training time

Image size is the most critical factor affecting training time as the image is resized to the required number of pixels before the images are fed to the model. Usually, image sizes used are 224x224 and 640x640. The higher the resolution, the longer the training time and the better the accuracy.

## 5.4 Observations from the Integration meeting

1. Model is basically overfitting to only detect boxes when there is a label.
2. Performance is inconsistent due to lighting conditions and different Box sizes.
3. Camera angle is more crucial for detection than the estimated difference.
4. Different Coordinate systems are required for different positions
5. Centroid calculation is done based on the position of the carton box in the image and not the actual centroid of the Carton box.**


Figure 26: Reference Frame at the centre of the depth sensor

# 6 Results

The project demonstrates the potential of AI-driven object detection for warehouse automation. Despite challenges like overfitting and inconsistent detection in certain conditions, the model shows promise for real-world application, particularly with further refinements and testing.

# 7 References

[1] The Transfer Learning Influence on YOLO v8 Performance For Motorbike Detection
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10425731

[2] Surgical Tool Detection in Open Surgery Based on Faster R-CNN, YOLO v5 and YOLOv8Detection
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10503806

[3] SCD: A Stacked Carton Dataset for Detection and Segmentation
https://arxiv.org/pdf/2102.12808

[4] You Only Look Once: Unified, Real-Time Object Detection
https://arxiv.org/pdf/1506.02640

# A Appendix: Python Code for Training and Validation in YOLO

```
import os
from IPython import display
import ultralytics
import torch
from ultralytics import YOLO
from IPython.display import display, Image

# Ensure this code is inside the main block
if __name__ == "__main__":
    # Get the current working directory
    HOME = os.getcwd()
    print(HOME)

    # Check system environment
    ultralytics.checks()

    # Check if GPU is available
    device = 'cuda'
    print(f"Using device: {device}")

    # Load the pre-trained YOLOv8 model
    model = YOLO("yolov8s.pt")

    # Train the model with output folder specified
    model.train(
        data="M:\\Uni_Siegen\\Sem-5\\SA\\Studienarbeit\\Work
            \\Training_Offline\\Training_Colored_Images\\data.
            yaml",  # Path to your data.yaml file
        epochs=50,          # Number of training epochs
        batch=16,           # Batch size
        imgsz=640,          # Image size
        plots=True,         # Enable plot generation during
            training
        project="M:\\Uni_Siegen\\Sem-5\\SA\\Studienarbeit\\
            Work\\Training_Offline\\Training_Colored_Images\\
            runs",  # Output folder for training results
        name="exp_colored"  # Name of the specific training
            experiment (e.g., 'exp_custom')
    )

    # Validate the trained model
    model = YOLO('M:\\Uni_Siegen\\Sem-5\\SA\\Studienarbeit\\
        Work\\Training_Offline\\Training_Colored_Images\\runs
        \\exp_custom\\weights\\best.pt')  # Load the best
```

```
        weights after training

    metrics = model.val(
        data="M:\\Uni_Siegen\\Sem-5\\SA\\Studienarbeit\\Work
            \\Training_Offline\\Training_Colored_Images\\data.
            yaml",  # Path to your data.yaml file
        imgsz=640,          # Image size
        plots=True          # Enable plot generation during
            validation
    )

    # Print the validation metrics
    print(metrics)
```