Construction & Robotics

RESEARCH DRIVEN PROJECT VOL. 7

Research Paper / SS 2024

Sigrid Brell-Cokcan
Thomas Adams

AUTHORS

Xiangcheng Li
Yufei Hu
Zeynep Sertkaya
Jiani Zhu Xingying
Li Zonghai Yang
Alperen Balcin
Merve Yanik
Damla Yigit
Nalan Yilmazarslan

construction robotics
www.cr.rwth-aachen.de

international M.Sc.programme

www.cr.rwth-aachen.de
cr@ip.rwth-aachen.de

Editors:
Univ.-Prof. Dr.-techn. Sigrid Brell-Cokcan
Dr.-Ing. Thomas Adams

RWTH Aachen University
Chair of Individualized Production
Campus-Boulevard 30
52074 Aachen
Germany
www.ip.rwth-aachen.de

Layout:
David Lukert B.Sc.

Original cover:
Lukas Kirner M.Sc.

Funded by Production Engineering - Grant of RWTH Aachen University

The content of the book was created for teaching purposes as part of the Research Driven Projects course.

RWTH Aachen University
Templergraben 55
52066 Aachen

First published 2024

Available via the institutional repository of RWTH Aachen University:
DOI: 10.18154/RWTH-2024-09253

The authors are responsible for the content of the papers.

XINGYING LI, ZONGHAI YANG

SmartRebar: Automated Reinforcement Monitoring System

ABSTRACT

The application of AI in construction sites has gained significant attention since the early 21st century, primarily focusing on non-productive functions such as helmet detection and material counting. This research explores the potential role of AI in the production process, specifically through the utilization of the YOLO (You Only Look Once) real-time detection model to identify and analyze construction reinforcements such as stirrups, longitudinal bars, and other elements tied to columns and beams.

The study investigates three primary research questions: (1) Can YOLO accurately identify various reinforcements in real-time? (2) Can the addition of segmentation and tracking layers enable YOLO to analyze area and count information effectively? (3) Can YOLO recognize the relative positions of stirrups or longitudinal bars in 3D space?

To address these questions, a dataset comprising 200 images was collected, 100 from actual construction sites and 100 from a laboratory model. The images were annotated using the "Segment Anything" format, specifically tagging different types of reinforcements, which were then converted into a YOLOv8-segment training format.

Through extensive training and fine-tuning, our research achieved several key outcomes. YOLO successfully identified all selected reinforcements and created bounding boxes encompassing the center points of all detected reinforcements in the images. The model also demonstrated the capability to recognize 3D information, determining whether reinforcements were positioned closer to the observer or camera. Additionally, YOLO could detect the absolute pixel differences in the y-coordinates between the center points of selected reinforcements and, using the width of the bounding box, identify the relative distances between them, such as the ratio of the diameters of adjacent reinforcements to the distance between them. Finally, the model was able to classify and count the different types of detected reinforcements.

Keywords: YOLOv8; Computer vision; 3D Recognition; Real-time Monitoring;

DOI: 10.18154/RWTH-2024-09257

Image created with Adobe Firefly

1. Introduction

The digitization process on construction sites is currently at a lower level compared to other industries. The review of construction components primarily involves quality inspections of rebar cages in beams and columns through random sampling on-site, with technicians manually measuring and compiling reports for construction operations.

- This approach lacks digital evidence storage, making the supervision quality of rebar cages highly dependent on the professional ethics of the technicians.
- The manual inspection process is time-consuming and labor-intensive, making it difficult to conduct comprehensive quality inspections across the entire site within limited timeframes.
- The absence of visualized rebar cage information means that future tasks, such as quality inspections, renovations, and demolitions of concrete structures, face additional testing challenges.

Simplifying and digitizing the manual supervision process in construction is crucial. Approaches like the use of point clouds and Building Information Modeling (BIM) are being explored, but both face similar challenges in terms of computational power and processing speed. In recent years, convolutional neural networks (CNN) have emerged as powerful tools for object recognition and information extraction from images. These models have demonstrated efficacy in automating tasks traditionally reliant on manual effort [1].

In this study, the performance of the recently popular real-time classification and detection model YOLOv8 is analyzed on a custom-built dataset to explore the potential of real-time detection AI models in the construction process. YOLOv8 is one of the computer vision modules in the artificial intelligence family. Computer vision enables computers to derive numeric information from digital images, videos, depth images, and 3D point clouds, process the information, and automate actions. Various aspects of applying computer vision to construction sites are relevant to this study, particularly object detection, recognition, labeling, learning processes, and 3D scene modeling. Although YOLOv8 has advantages in speed, it is slightly lacking in accuracy compared to other models [1] .

To explore the potential applications of YOLOv8 in the construction process, this study investigates three primary research questions:

1. Can YOLO accurately identify various reinforcements in real-time?
2. Can the addition of segmentation and tracking layers enable YOLO to analyze area and count information effectively?
3. Can YOLO recognize the relative positions of stirrups or longitudinal bars in 3D space?

2. Literature review

current famous usecase of computer vision in construction projects including using collected point clouds, images, and other sensor data to generate visualized models of construction site progress and then monitoring the construction progress. It shows great capabilities in visualization and abstruct information from construction site, automation, etc., but faces challenges such as high technical thresholds and complex data processing [2].

Figure 2 A result from Mask R-CNN.

2.1 Computer vision in point clouds

There are also studies in computer vision-based construction progress monitoring that using machine learning models to learn 2D image patches sampled per element and classify them into different material types. This enables the inference of different material types from point clouds, and progress tracking at the operational level based on reasoning about the observed frequencies of different material types [1].

2.2 AI applications for 2D images

The other example, Mask R-CNN models, used for image segmentation and information extraction, have found applications in detecting defects and segmenting surfaces [3].

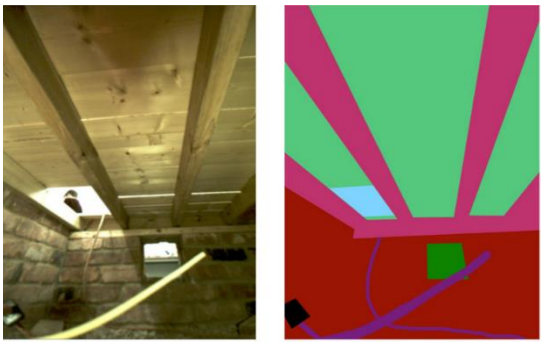


Figure 1: Encoding floorboards (light green), joists (magenta), walls (red), vents (dark green), cables (purple), and openings (cyan). Areas in black remained unclassified [3].

In the realm of construction sites, the utilization of the Mask R-CNN model is still in its nascent stages. However, there have been studies indicating its reliability in extracting various types of information from construction sites. Examples including automatically detect and segment defects on structure surfaces [4], identify regions such as floorboards, joists, air vents, pipework [3] and so on.

Here shows a result in the research conducted by Gary A. Atkinson and Wenhao Zhang in 2020, the Mask R-CNN model was found to reliably extract information from construction sites using only a small number of samples. They utilized the model to detect the underfloor scene, with results depicted in

2.3 AI applications for 3D images

There is also a lot of research in 3D imagery, such as ALPMS, which works by inputting images of a construction site and a four-dimensional Building Information Model (BIM) and outputting the percentage of completion of activities needed to update the project schedule. It creates as-built point clouds from the images, compares them with the planned BIM, generates orthographic views of the under-construction elements by projection transformation or Neural Radiation Field (NeRF), applies Deep Learning (AI)-based semantic segmentation for schedule inference, and estimates the percentage completion. Finally, the rich semantic information at the activity level is transferred to the as-built point cloud and BIM for 3D visualisation of the progress status. When applying the framework to two building construction projects, on average, ALPMS was able to report the activity percentage of completion with an average absolute error of less than 6%. Future research will focus on automating the mapping of project schedule activities and activity-level progress details for timely schedule updates and project completion dates [5]. Figure 2 shows the workflow.

To realise the application of Yolo model in the construction and supervision process, the movable camera position and ranging function are indispensable, however, most of the current research on Yolo is used for classification and counting, and the few ranging studies are based on the use of a specific space with a fixed camera, and the parameters are determined through on-site debugging. In this paper, the hoop spacing is dynamically obtained from the positioning frame of the hoops, which enables the use of a dynamic camera and the estimation of the relative position of the hoops.

3. Research method

The overall research two main experiments research process:

- Training model process: Analyze and find suitable training methods and hyperparameters to train and obtain suitable models.
- Output analysis process: Analyze the output of the model to get possible function based on the trained model.

The training model process mainly covers data collection, creation of the training set, and model tuning and training. This includes video collection, frame extraction and annotation, training set expansion, model training, and hyperparameter tuning, with the goal of finding a solution for enabling the model to accurately determine rebar types and their spatial locations.

Training model process:

Figure 3 Training model process

The authors are responsible for the content of the papers.



Figure 2: ALPMS workflow. Source: Pal et al. (2024) [5].

3.1 Data collection

The collected raw dataset is divided into two groups: 100 images collected from the construction site and 100 model images collected in the laboratory. The limited variety and small number of collected images could lead to a severe overfitting problem.

By altering the brightness, color temperature, and modifying the background, the original data set is expanded from 200 images to 1700 images and results in a better analysis output [6].

3.2 Datasets building

The creation of the Dataset is crucial. By labeling the same type of rebar located in the background as "back-ground," the model is enabled to recognize the 3D position and identify the rebar in the foreground.

3.3 Adjusting training methods

The creation of the Dataset is crucial. By labeling the same type of rebar located in the background as "back-ground," the model is enabled to recognize the 3D position and identify the rebar in the foreground. It has been shown that too few images and too many labelled columns and beams can lead to overfitting of the generated model. When a given training set is well labelled and there is enough features available for the model to learn, the model can accurately locate and classify the target by the pre-designed features, but when overfitting occurs, the model misses the pre-designed features and extracts other features to locate and classify the target. Overfitting results in a significant decrease in the reliability of the model's performance.

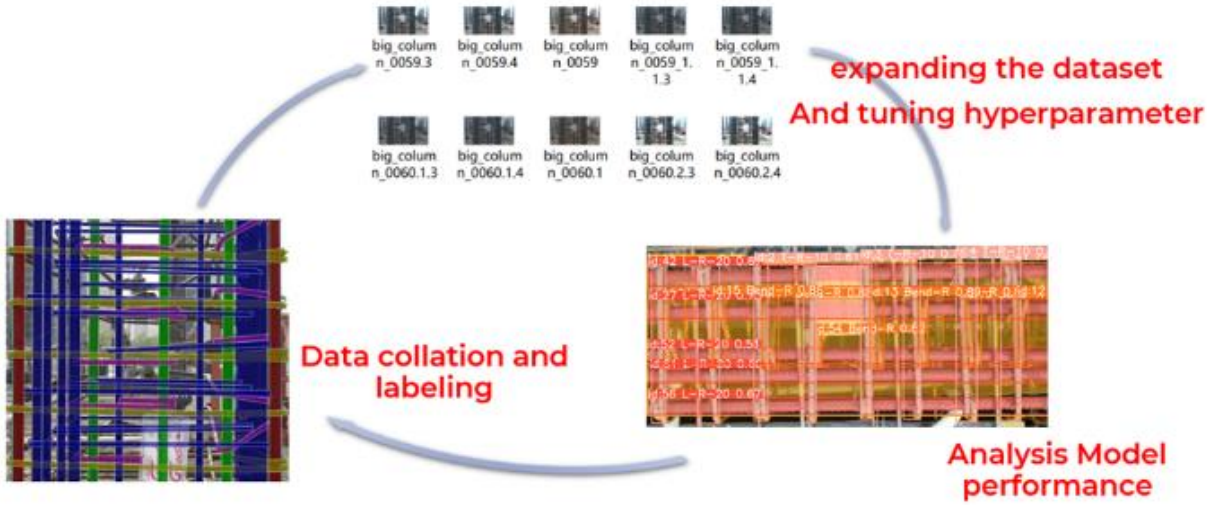


Figure 3: Training model process

3.4 Model performance analysis

For this model, the most difficult feature to obtain is the 3D position information of the longitudinal bars, because the longitudinal bars in the produced model are too distinguishable from each other, which leads to the difficulty of capturing the 3D information during the model training process, in addition to adjusting the correct hyperparameters, it is also necessary to periodically investigate the trained model to ensure that it does not lose the 3D information.

4. Prototype

4.1 Concept development

4.1.1 Concept of the Prototype

The project is designed to automate the monitoring of rebar placement in construction environments using sophisticated computer vision technologies. The fundamental concept revolves around leveraging the YOLOv8 model, a cutting-edge object detection algorithm, for real-time identification and classification of various rebar types, such as stirrups and longitudinal bars, within a three-dimensional spatial context. This system extends its capabilities by incorporating image segmentation, object tracking, and video processing technologies, which collectively enhance the detection accuracy and spatial awareness of the monitored construction elements. By integrating these advanced technologies, the project provides an intelligent and automated solution that not only improves construction quality and safety but also reduces the manual effort required to ensure precise rebar placement [7].

4.1.2 Tools and Platforms Used

The development of this prototype is based on several established open-source tools and platforms that are commonly used in computer vision and machine learning applications:

YOLOv8: YOLOv8 is the cornerstone of our system, providing real-time object detection capabilities that are essential for identifying rebar types and their positions within video feeds. The script my_detect.py demonstrates how YOLOv8 is utilized for continuous object tracking across video frames. The model is initialized with pre-trained weights optimized for detecting construction elements, and it processes each video frame to identify rebar, drawing bounding boxes around detected objects and classifying them by type. This setup allows for the precise localization of rebar in a 3D space, crucial for ensuring correct placement and spacing [8].

OpenCV: OpenCV is extensively used for handling video and image processing tasks within the prototype. In count_base.py, OpenCV manages the real-time input from

The authors are responsible for the content of the papers.

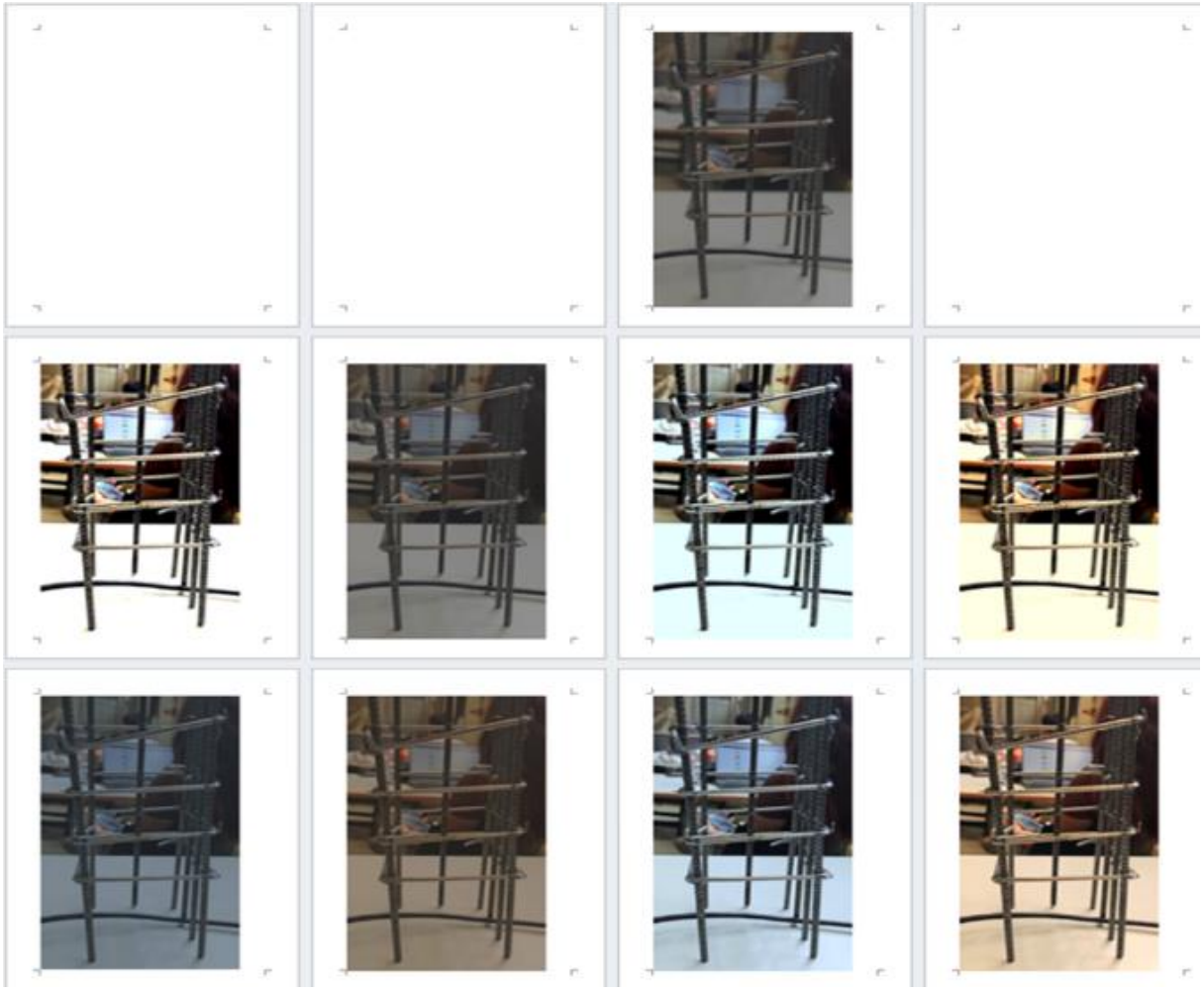


Figure 4: The process of data expansion is demonstrated, where the original image is expanded by adjusting brightness, color, and temperature to avoid model overfitting.

video feeds, captures frames, and converts them into formats suitable for processing by YOLOv8. OpenCV is also responsible for the visualization of detection results, drawing annotations and bounding boxes on frames, and displaying them in real time. This integration is vital for providing immediate feedback on rebar placement, helping to quickly identify any deviations from the desired configuration [9].

Segment Anything: This tool is employed to enhance the prototype's detection accuracy by performing image segmentation. By distinguishing rebar from the background, the system can more accurately identify and classify each element. The segmentation process is integrated into the detection pipeline to refine the regions of interest, improving the overall reliability of the rebar detection model. This step is critical for environments where the background might be complex or contain elements similar to rebar, thus preventing false positives [10].

BoT-SORT: Object tracking is another key component of the prototype, facilitated by BoT-SORT. This algorithm ensures that the detected rebar is continuously tracked across multiple frames, maintaining an accurate representation of their positions over time. The script my_detect.py incorporates BoT-SORT to link detected objects between frames, which is crucial for monitoring rebar that moves or changes appearance slightly due to camera angles or environmental conditions. This capability is essential for a comprehensive monitoring system that needs to ensure the correct placement and integrity of rebar throughout a construction project [11].

4.1.3 Research Process

The prototype design is visually represented through several diagrams and flowcharts that illustrate the system architecture and data processing pipeline. For instance, one

diagram may depict the flow of video data from the camera to the microcomputer, where it is processed by OpenCV, analyzed by YOLOv8, and then passed through segmentation and tracking modules. Another visualization could highlight the integration of different software components, showing how each tool (e.g., YOLOv8, OpenCV, BoT-SORT) interacts to deliver real-time detection and monitoring. This comprehensive visualization not only helps in understanding the system's operational flow but also aids in identifying potential areas for optimization or troubleshooting.

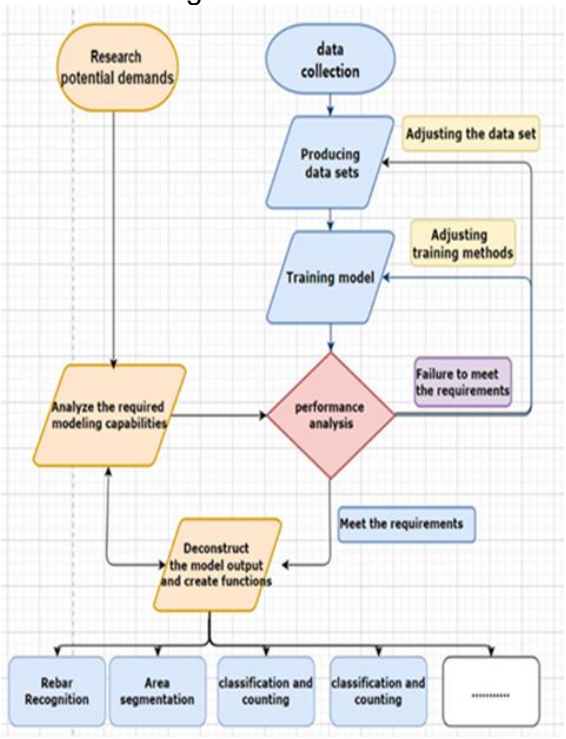


Figure 5: Overall research process

4.2 Prototype development

4.2.1 Realization of the Demonstration Sample

The prototype of "SmartRebar-Automated Reinforcement" is built to work with both simple and advanced tech. On the hardware side, it uses cameras and a microcomputer, like a Raspberry Pi or Arduino, to capture real-time video from construction sites. This setup is super flexible—you can use high-res cameras for a detailed view or just regular PC webcams, whether they're built-in or external. This means the system can easily fit into different setups, making it versatile for various construction needs.

For software, the system relies on YOLOv8, a powerful tool for detecting objects, which helps analyze video data instantly. Scripts like count_base.py and my_detect.py show how this model can be used to spot and track rebar in videos. With OpenCV, the system can process video from any camera, whether it's a high-end one on a construction site or a basic webcam plugged into a PC. This makes it adaptable and user-friendly, no matter what kind of equipment you have [12].

4.2.2 Hardware Configuration and Installation

For deploying YOLOv8 and OpenCV, a basic hardware setup is adequate. It is recommended to use high-definition cameras with at least 1080p resolution and 30 FPS frame rate for capturing detailed images. A microcomputer or workstation with a GPU such as the NVIDIA GTX 1050 or AMD Radeon RX 560, 15GB of RAM, and a mid-range processor like the Intel Core i5-10400 or AMD Ryzen 5 3600 is suitable for running YOLOv8 and OpenCV efficiently. For storage, an SSD or HDD with around 500GB is sufficient to manage video data and datasets. A standard internet connection with a bandwidth of at least 20 Mbps is enough for data transfer and remote monitoring. This setup provides a balanced and cost-effective solution for real-time object detection, ensuring smooth operation without the need for high-end components.

To set up the microcomputer for deploying YOLOv8 and OpenCV, start by installing the operating system, such as Ubuntu or Windows. If using an NVIDIA GPU, install the necessary GPU drivers and CUDA toolkit to ensure optimal performance. Next, create a Python virtual environment and install the required packages by running the following commands in the terminal.

Enter the command line in the prompt:

- conda activate (envspace_name)
- pip install ultralytics opencv-python
- sudo apt install ffmpeg

The authors are responsible for the content of the papers.

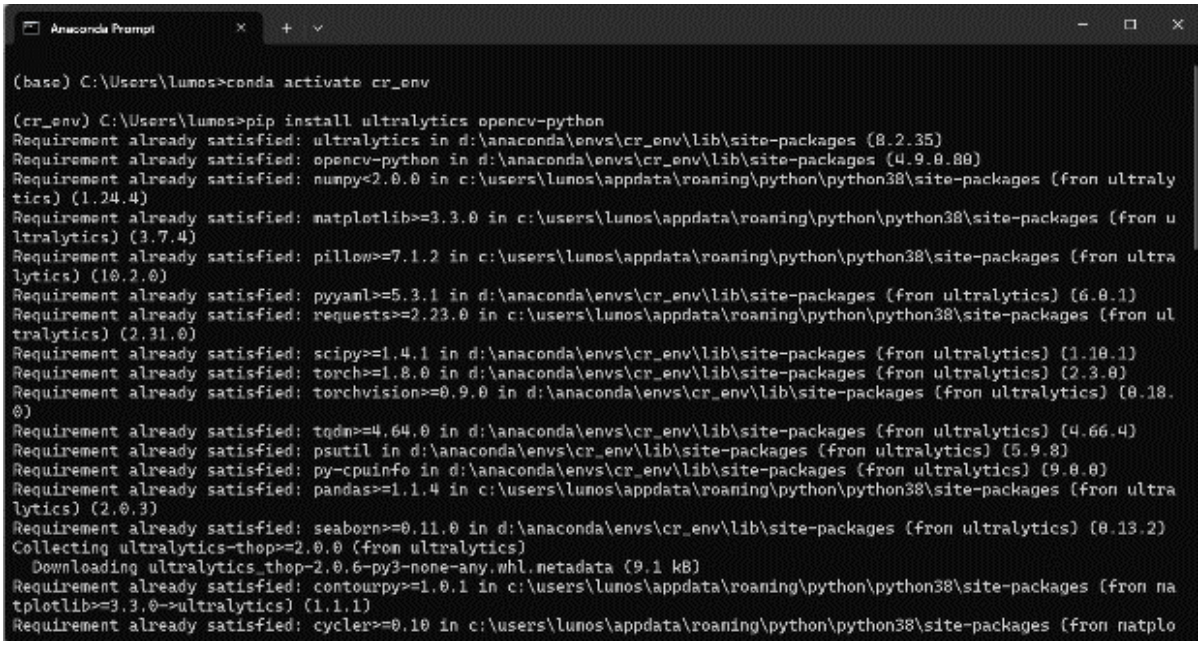


Figure 6: Enter the command in the prompt

```python
def adjust_darker(input_folder, output_suffix=".1", factor=0.8):
    # Get all file names in a folder
    for filename in os.listdir(input_folder):
        # Check if the file is a .jpg image and does not contain '.'
        if filename.lower().endswith('.jpg') and '.' not in filename[:-4]:          # Exclude the extension part for dot check
            # Opening an image file
            img_path = os.path.join(input_folder, filename)
            img = Image.open(img_path)
            random_number = round(random.uniform(-0.1, 0.1), 2)
            adjusted_factor = factor + random_number                                # introduce 20% randomness

            # Adjusting Brightness
            brightness_enhancer = ImageEnhance.Brightness(img)
            img_brightness_enhanced = brightness_enhancer.enhance(adjusted_factor)  # Reduce brightness by factor

            # Adjusting Contrast
            contrast_enhancer = ImageEnhance.Contrast(img_brightness_enhanced)
            img_enhanced = contrast_enhancer.enhance(adjusted_factor)               # Reduce contrast by factor

            # Constructing a new filename
            new_filename = f"{os.path.splitext(filename)[0]}{output_suffix}{os.path.splitext(filename)[1]}"
            new_img_path = os.path.join(input_folder, new_filename)
            print(f'adjust_darker: {filename} -> {new_filename}')

            # Saving the adjusted image
            img_enhanced.save(new_img_path)
```

Figure 8: Introducing Randomness

4.2.3 Model training

(1) Data Preparation Step

High-definition cameras are used to collect video data on construction sites. Then, the video frames are extracted using the FFmpeg tool in Anaconda Prompt. FFmpeg is an open-source multimedia framework that can handle audio and video data, including recording, conversion, and streaming operations.

The command used is as follows:

ffmpeg -i D:\0.8m_bar\25s_40s_beam.MP4 -ss 00:00:20 -t 00:00:20 -vf fps=4 output_dir/data-beam_%04d.jpg

- -i specifies the input file. D:\0.8m_bar\25s_40s_beam.MP4 is the path to the input video file.
- -ss specifies the starting point in the video. Here, 00:00:20 indicates that processing starts at the 20-second mark of the video.
- -t specifies the duration. Here, 00:00:20 means that processing will cover the next 20 seconds of video content starting from the specified start time.
- -vf indicates the use of a video filter.
- fps=4 sets the video frame rate to 4 frames per second. This means 4 frames will be extracted per second from the video.

```python
def adjust_warmer(input_folder, output_suffix=".4", factor=0.9):
    # Get all file names in a folder
    for filename in os.listdir(input_folder):
        # Check if the file is a .jpg image and does not contain '.'
        if filename.lower().endswith('.jpg') and '.3' not in filename[:-4]:          # Exclude the extension part for dot check
            # Opening an image file
            img_path = os.path.join(input_folder, filename)
            img = Image.open(img_path)
            random_number = round(random.uniform(-0.05, 0.05), 3)
            adjusted_factor = factor + random_number                                 # introduce 10% randomness
            # Adjusting Saturation
            color_enhancer = ImageEnhance.Color(img)
            img_color_enhanced = color_enhancer.enhance(factor)                      # Increase saturation by 20%

            # Adjusting Warmth by adjusting the image colors
            r, g, b = img_color_enhanced.split()
            r = r.point(lambda p: p * factor if p * factor <= 255 else 255)          # Increase red channel
            b = b.point(lambda p: p * (2 - factor) if p * (2 - factor) >= 0 else 0)  # Decrease blue channel
            img_warmer = Image.merge('RGB', (r, g, b))

            # Constructing a new filename
            new_filename = f"{os.path.splitext(filename)[0]}{output_suffix}{os.path.splitext(filename)[1]}"
            new_img_path = os.path.join(input_folder, new_filename)
            print(f'adjust_warmer: {filename} -> {new_filename}')

            # Saving the adjusted image
            img_warmer.save(new_img_path)
```

Figure 9 adjusting the brightness

• output_dir/ specifies the output directory.

• data-beam_%04d.jpg is the naming format for the output images, where %04d is a placeholder that indicates a 4-digit number sequence, resulting in filenames like data-beam_0001.jpg, data-beam_0002.jpg, and so on.

Subsequently, in Anaconda Prompt, Labelme is used to annotate the collected images in the "segment anything" format. After annotation, JSON files containing the annotation information. are generated, such as data-beam_0001.json, data-beam_0002.json, and so on.

Next, some functions are defined to augment the dataset by adjusting the brightness and warmth of the images. For example, starting with the image data-beam_0001.jpg, the process involves the following steps:

(2) **Brightness Adjustment**: The brightness of the image is modified to create variations. This results in:
   - data-beam_0001.1.jpg (the original image with increased brightness)
   - data-beam_0001.2.jpg (the original image with decreased brightness)
   - data-beam_0001.1.2.jpg (the brightness-increased image further adjusted to decrease brightness)

- **Introducing Randomness**: During the transformation process, random values are introduced, which means that each of the augmented images will have unique brightness distributions [13].

Next, the warmth and coolness of the images are adjusted. This process is similar to the brightness adjustments, but to enhance the randomness of the dataset, some images marked with certain labels are randomly selected to skip this transformation. For example, during the process of cooling down the images, those with names containing '.3' are excluded from the transformation.

After adjusting the brightness, the number of images was increased to four times the original amount. Following the warmth and coolness adjustments, the number of images was expanded to approximately fifteen times. the original quantity. However, it is important to note that while the number of images has been increased, the JSON files have not bee n proportionally expanded.

The transformed images only changed in terms of brightness and color, and did not alter the content or position of the images (such as flipping or rotating). Therefore, all the augmented images essentially share the same coordinate information, meaning their JSON files are identical. As a result, all the

```python
base = r"D:\Desktop\data_column"
json_dir = os.path.join(base, "json")
labels_dir = os.path.join(base, "labels")

print('Looking for JSON files...')
if not os.path.exists(json_dir):
    print('Error: JSON files directory not found:', json_dir)
else:
    print('JSON files directory found.')

    if not os.path.exists(labels_dir):
        print('Creating labels directory...')
        os.makedirs(labels_dir)

    for name in findAllFile(json_dir):
        print('Creating TXT file for', name)
        with open(os.path.join(base, "json", f"{name}.json"), 'r') as jf:
            data = json.load(jf)
        height = data["imageHeight"]
        width = data["imageWidth"]

        s = ""
        for item in data["shapes"]:
            label = item["label"]
            if label not in label_dict.keys():
                label_dict[label] = label_num
                label_num += 1

            s += str(label_dict[label]) + " "

            points = item["points"]
            for point in points:
                s += str(point[0] / width) + " "
                s += str(point[1] / height) + " "
            s = s[:-1] + "\n"

        with open(os.path.join(labels_dir, f"{name}.txt"), 'w') as tf:
            tf.write(s)


    yaml_path = os.path.join(base, "dataset.yaml")
    with open(yaml_path, "w") as f:
        print('Creating YAML file...')
        f.write(f"path: {base}\n")
        f.write("train: images\n")
        f.write("val: images\n")
        f.write("test: \n\n")
        f.write("names:\n")
        for key, num in label_dict.items():
            f.write(f"  {num}: {key}\n")

    print("Process completed.")
```

Figure 10 main modifications of YAML file

JSON files were duplicated and pasted until the number matched the expanded image set [14]. Then, the names of the JSON files were batch-renamed to correspond one-to-one with the expanded image set, completing the dataset expansion process.

An important step that follows is converting the "segment anything" format JSON annotation files into the TXT format files used for training YOLO. After a series of investigations and attempts, a Python script was written to achieve this. The script performs the following tasks:

- **Read and Parse JSON Content**: The script first reads and parses the JSON file to obtain the image's width and height.
- **Process Annotations**: For each annotation (in shapes), it retrieves the label and checks if it is already mapped in label_dict. If not, a new label ID is assigned.
- **Write TXT Files**: The script then constructs the YOLO format label content, which includes the label ID and coordinates of the annotation points relative to the image dimensions. This content is written into a corresponding TXT file, which has the same name as the input JSON file but with a .txt extension.
- **Create YAML File**: Finally, the script creates a YAML file to describe the basic information of the dataset.

Subsequently, according to YOLO's required format, the YAML file used for training the model is modified based on the corresponding content in the newly generated YAML file. The main modifications include:

```python
from ultralytics import YOLO
import multiprocessing

def train_model(epochs, imgsz, workers, batch_size, pretrained, device=0,
                mosaic=False, box=10.5, cls=5.5,  patience=50, verbose= True,
                lr0=0.01, lrf=0.01, optimizer='auto', overlap_mask=True,
                val=True, plots=True,  degrees=5, flipud=0.2, scale=0.1):#
    # Load a model
    model = YOLO(r"D:\Desktop\ultralytics\ultralytics\runs\segment\train4\weights\best.pt")
    # Train the model
    model.train(data=r"D:\Desktop\ultralytics\ultralytics\cfg\datasets\coco128-seg.yaml",
                epochs=epochs,
                imgsz=imgsz,
                workers=workers,
                batch=batch_size,
                pretrained=pretrained,
                patience=patience,
                device=device,
                mosaic=False,
                lr0=lr0,
                lrf=lrf,
                optimizer=optimizer,
                overlap_mask=overlap_mask,
                val=val,
                degrees=degrees,
                flipud=flipud,
                scale=scale,

                # Add mosaic parameter to augmentations
                )

if __name__ == "__main__":
    multiprocessing.freeze_support()
    # Define hyperparameters grid
    imgsz_list = [760]
    pretrained_list = [True]
    optimizer_list = ['auto']  # Add optimizer options here

    # Grid search
    for imgsz in imgsz_list:
        for pretrained in pretrained_list:
            for optimizer in optimizer_list:
                train_model(epochs=300, imgsz=imgsz, workers=1, batch_size=2,
                            pretrained=pretrained, mosaic=False, device=0,
                            patience=25, optimizer=optimizer, overlap_mask=True,
                            val=True)
                print(f"Training model with imgsz={imgsz}, pretrained={pretrained}, optimizer={optimizer}")
```

Figure 11 labels output and file paths

- **Mapping Output Labels to Actual Names**: This involves defining the correspondence between model output labels such as '0', '1', '2', '3', etc., and their actual names, for example, '20MM Longitudinal Rebar' as 'L-R-20', '10MM Stirrups' as 'T-R-10', and background as 'B-R', etc.
- **File Paths for Training**: The paths to the training files are specified.

Note that there is a space after the colon in the mapping between the output number and the real name of the model under names, and that not adding a space will result in a program error.

(3) Model parameters select process

Next comes the process of selecting hyperparameters for training the model.

In our training setup, all required hyperparameters were first identified and then tested multiple times. The hyperparameters that were most challenging to evaluate were divided into three groups:

- **imgsz**: 460, 760, 1080
- **Pretrained**: True, False
- **Optimizer**: Auto, SGD, Adam, AdamW, etc.

After training on a small subset of the data, the most suitable combination was identified by comparing box loss and test results: **imgsz: 760**, **Pretrained: True**, **Optimizer: Auto**.

It is important to note that the **Optimizer: Auto** option overrides the initial settings for the learning rates **lr0** and **lrf**. Through multiple trials, it was found that **Optimizer: Auto** was consistently the best option.

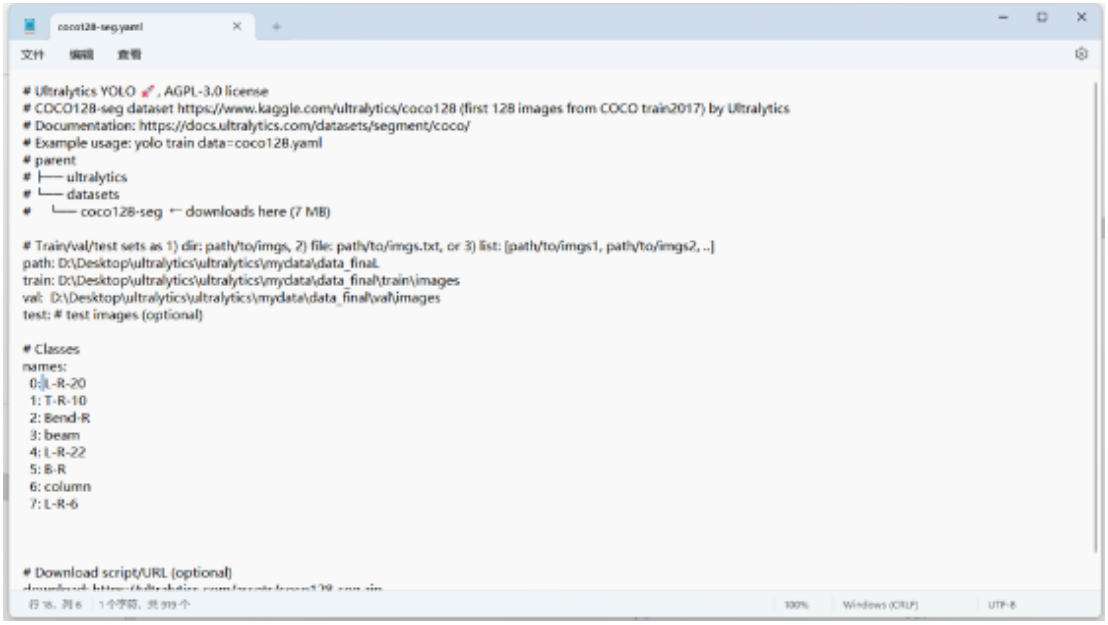


Figure 12: model's parameter

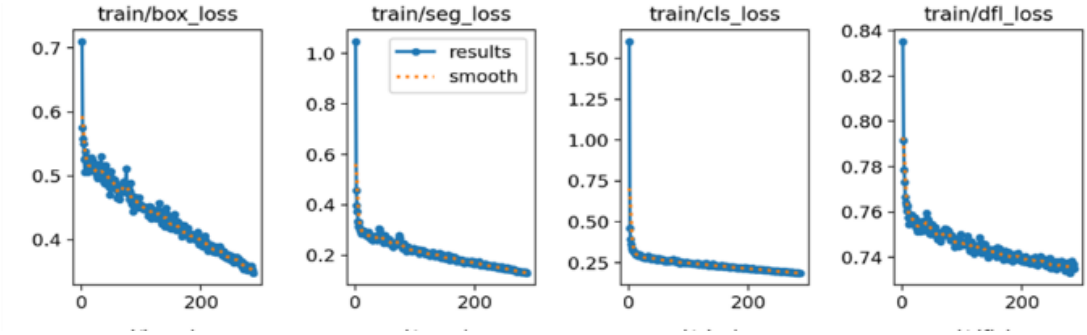


Figure 13: model performance

(4) Training supervisory process

During training, to mitigate the risk of overfitting, it is crucial to periodically check whether the trained model retains its ability to recognize 3D features [15]. If the model loses its ability to identify 3D features, training must be stopped promptly. In such cases, training should be resumed from the most recent suitable checkpoint or epoch where the model still demonstrated good performance in recognizing 3D features [16].

The code is available on Github: https://github.com/lumosconcerto3o/SmartRebarDetection.git

4.3 Prototype demonstration

4.3.1 How it works

- **Video Capture**: The system can use any camera—whether it's a high-quality site camera or just a basic webcam plugged into your PC. It captures live video feeds from the construction site to monitor the placement of rebar.
- **Detection and Tracking**: Using the YOLOv8 model, it analyzes the video frame-by-frame. It looks for different types of rebar, identifies them, and places a colored box around each detected piece [17]. This makes it easy to see exactly where each piece of rebar is on the screen. The ***my_detect.py*** script handles this process, running the YOLOv8 model to detect objects and track them across frames.
- **Real-Time Feedback**: The results are displayed instantly. You can watch the video feed with all the rebar labeled and tracked in real time, which helps in immediately identifying if something is misplaced or missing. The system even counts the number of rebar pieces and shows this data on the video output. For

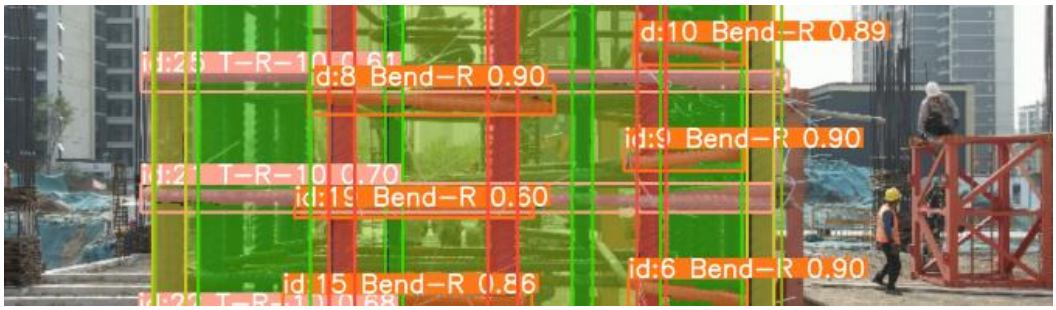


Figure 14: YOLOv8 model training for rebar detection, labeling positions and classifications

example, a small counter might pop up on the screen, showing the total number of detected rebar pieces and their types.

4.3.2 Outputs

Annotated Video: The output video will have boxes around detected rebars with labels showing what type they are. If you're using the system in a construction setting, you'll see these boxes moving around as workers handle the rebar, giving you a real-time update on placement accuracy [18].

Count and Classification: The system doesn't just show where the rebars are; it also counts them. You'll get a running tally of different types of rebar detected on the screen, making it easier to keep track of inventory and placement without manual counting.

Distance Measurement: In addition to detection and counting, the system can measure the distance between rebars, which is crucial for ensuring they are placed correctly according to construction guidelines. This data is overlaid on the video, so you can see exactly how far apart each piece is [19].

Alerts and Reports: If something is out of place—say, two pieces of rebar are too close together or one is missing—the system can send an alert. This helps the construction manager quickly address any issues, ensuring the site remains compliant with safety standards.

5. Scientific results and discussion

This study used the YOLOv8 model, a real-time object detection algorithm, to identify and classify different types of rebar used in construction. We focused on three main goals: (1) determining how accurately YOLO can detect reinforcements like stirrups and longitudinal bars in real time, (2) testing if adding segmentation and tracking features helps YOLO analyze the area and count of these reinforcements more effectively, and (3) assessing if YOLO can recognize the positions of these elements in 3D space [20].

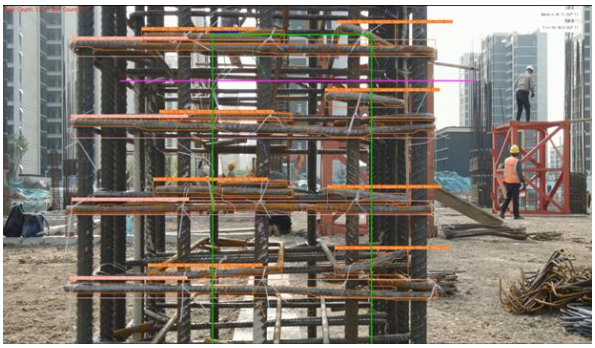


Figure 15 Correct and incorrect identification and distance

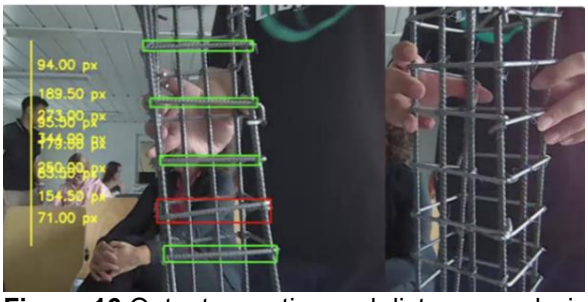


Figure 16 Output: counting and distance analysis



Figure 17 Output: counting and distance analysis

The authors are responsible for the content of the papers.

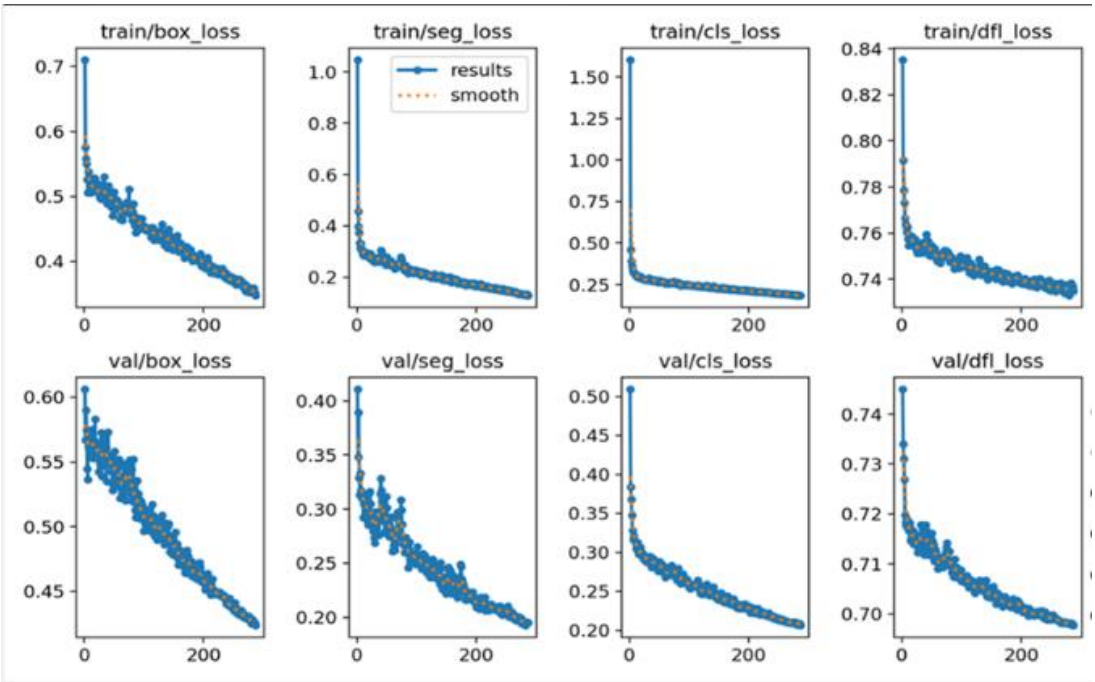


Figure 14: Model performance and comparison

Adapted, output that meets two goals, and a model that meets expectations is presented:

- **Accuracy in Detection and Classification:**

The results showed that the YOLOv8 model was effective in detecting and classifying all selected reinforcements, including stirrups and longitudinal bars. We initially used a dataset of 200 images but expanded it to 1700 images through data augmentation, such as changing brightness and color temperature and altering backgrounds [6]. This helped the model to achieve high precision in detecting these elements on both actual construction sites and in controlled lab settings. The model accurately placed bounding boxes around the center points of all detected reinforcements, showing its ability to correctly identify rebar types.

- **Analyzing Area and Count:**

With the addition of segmentation and tracking layers, the YOLOv8 model could not only identify reinforcements but also provide detailed information about their area and count. Segmentation improved the model's ability to distinguish rebar from the background, especially in complex environments where other objects might be present. The tracking feature ensured that the model could continuously detect rebar across multiple video frames, which is crucial for keeping an accurate count in dynamic construction scenarios [21].

- **Recognizing 3D Spatial Information:**

The model demonstrated the ability to understand 3D spatial information. It could determine whether reinforcements were closer to or further from the camera. This was done by calculating the pixel differences in the y-coordinates between the center points of detected reinforcements [22]. By using the width of the bounding boxes, the model could also estimate the distances between different reinforcements, like comparing the sizes of adjacent bars to the space between them.

6. Conclusion

- **Implications for Construction Automation:**

These findings suggest that using YOLOv8 in construction can greatly improve the automation and digitalization of inspection processes. The model's ability to detect, classify, and analyze rebar in real time can reduce the need for manual inspections, which are often slow and prone to errors. Adding segmentation and tracking features makes the model more reliable, especially in dynamic environments where objects are moving or partially hidden.

- **Challenges and Limitations:**

The model's performance depends on the quality of the input data and the precision of the annotations. In settings with poor lighting or a lot of visual clutter, the accuracy of detection might decrease. Moreover, running this model in real time requires substantial computing power, which may not be available on all construction sites.

- **Future Research Directions:**

Future research could focus on optimizing the YOLOv8 model to work on lower-end hardware, making it more accessible for a wider range of construction sites. Expanding the dataset to include more diverse construction scenarios and rebar types could improve the model's versatility. Additionally, integrating other AI models, like Mask R-CNN.

7. References

[1] Redmon, J. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[2] Cheng, T., & Teizer, J. (2013). Real-time resource location data collection and visualization technology for construction safety and activity monitoring applications. *Automation in construction*, 34, 3-15.

[3] Xu, Y., Li, D., Xie, Q., Wu, Q., & Wang, J. (2021). Automatic defect detection and segmentation of tunnel surface using modified Mask R-CNN. *Measuremen*t, 178, 109316.

[4] Tabernik, D., Šela, S., Skvarč, J., & Skočaj, D. (2020). Segmentation-based deep-learning approach for surface-defect detection. *Journal of Intelligent Manufacturing*, *31*(3), 759-776.

[5] Pal, A., Lin, J. J., Hsieh, S. H., & Golparvar-Fard, M. (2024). Activity-level construction progress monitoring through semantic segmentation of 3D-informed orthographic images. *Automation in Construction*, *157*, 105157.

[6] Gong, J., & Caldas, C. H. (2010). Computer vision-based video interpretation model for automated productivity analysis of construction operations. Journal of Computing in Civil Engineering, 24(3), 252-263.

[7] Li, Y., Lu, Y., & Chen, J. (2021). A deep learning approach for real-time rebar counting on the construction site based on YOLOv3 detector. *Automation in Construction*, *124*, 103602.

[8] Zhichao, H., Yi, W., Junping, W., Wanli, X., & Bilian, L. (2023). Improved lightweight rebar detection network based on YOLOv8s algorithm. *Advances in Computer, Signals and Systems*, *7*(10), 107-117.

[9] Frangez, V., Lloret-Fritschi, E., Taha, N., Gramazio, F., Kohler, M., & Wieser, A. (2021). Depth-camera-based rebar detection and digital reconstruction for robotic concrete spraying. *Construction Robotics*, 1-12.

[10] Ahmadi, M., Lonbar, A. G., Sharifi, A., Beris, A. T., Nouri, M., & Javidi, A. S. (2023). Application of segment anything model for civil infrastructure defect assessment. *arXiv preprint arXiv:2304.12600*.

[11] Aharon, N., Orfaig, R., & Bobrovsky, B. Z. (2022). BoT-SORT: Robust associations multi-pedestrian tracking. *arXiv preprint arXiv:2206.14651*.

[12] Zhang, J., Kang, X., Ni, H., & Ren, F. (2021). Surface defect detection of steel strips based on classification priority YOLOv3-dense network. *Ironmaking & Steelmaking*, *48*(5), 547-558.

[13] Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., & Kalinin, A. A. (2020). Albumentations: fast and flexible image augmentations. *Information*, *11*(2), 125.

[14] Jin, Y., Mishkin, D., Mishchuk, A., Matas, J., Fua, P., Yi, K. M., & Trulls, E. (2021). Image matching across wide baselines: From paper to practice. *International Journal of Computer Vision*, *129*(2), 517-547.

[15] Su, H., Qi, C. R., Li, Y., & Guibas, L. J. (2015). Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *Proceedings of the IEEE international conference on computer vision* (pp. 2686-2694).

[16] Hou, Z., Qin, M., Sun, F., Ma, X., Yuan, K., Xu, Y., ... & Kung, S. Y. (2022). Chex: Channel exploration for cnn model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 12287-12298).

[17] Khan, N., Zaidi, S. F. A., Yang, J., Park, C., & Lee, D. (2023). Construction work-stage-based rule compliance monitoring framework using computer vision (CV) technology. *Buildings*, *13*(8), 2093.

[18] Wang, S., Kim, M., Hae, H., Cao, M., & Kim, J. (2023). The development of a rebar-counting model for reinforced concrete columns: Using an unmanned aerial vehicle and deep-learning approach. *Journal of Construction Engineering and Management*, *149*(11), 04023111.

[19] Khow, Z. J., Tan, Y. F., Karim, H. A., & Rashid, H. A. A. (2024). Improved YOLOv8 Model for a comprehensive approach to object detection and distance estimation. *IEEE Access*.

[20] Jana, A. P., & Biswas, A. (2018, May). YOLO based Detection and Classification of Objects in video records. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)* (pp. 2448-2452). IEEE.

[21] Jiang, H., Sun, D., Jampani, V., Yang, M. H., Learned-Miller, E., & Kautz, J. (2018). Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 9000-9008).

[22] Caicedo, J. C., & Lazebnik, S. (2015). Active object localization with deep reinforcement learning. In *Proceedings of the IEEE international conference on computer vision* (pp. 2488-2496).

The authors are responsible for the content of the papers.