GTU DEPARTMENT OF COMPUTER ENGINEERING

AI Based Drone Footage Foreign Object Detector For Roads

Gradution Project Report

Project Advisor

DOÇ. DR. MEHMET GÖKTÜRK

TAHA KINALI

1901042660

CONTENTS

Introduction

In this project, I developed a system capable of real-time identification of objects such as boxes, people, and dogs using a drone. This system analyzes the images captured by the drone's camera, utilizing various image processing and artificial intelligence techniques to identify specific objects. The primary goal of the project is to enhance the efficiency and accuracy of monitoring and detecting foreign objects on roads using drone technology.

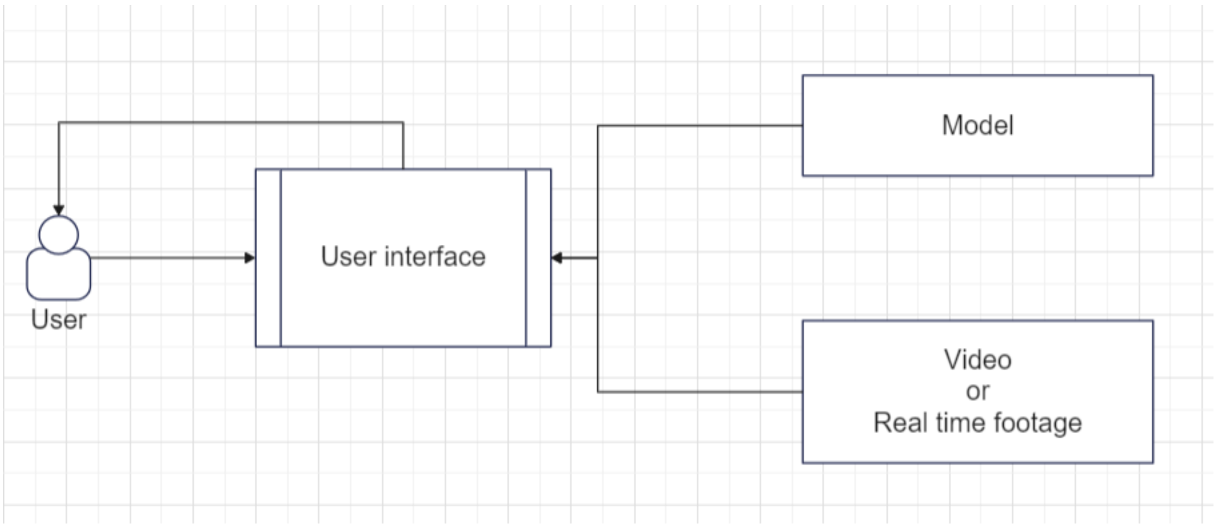
The system leverages advanced machine learning models, specifically the YOLO (You Only Look Once) model, which is renowned for its high-speed object detection capabilities. By integrating this model with drone footage, the project aims to provide a robust solution for real-time surveillance and monitoring applications. The system can be particularly useful for road maintenance, security surveillance, and traffic management, where quick and accurate detection of objects is crucial.

The development process involved multiple stages, including data collection, preprocessing, model training, and application development. The dataset used for training the model was carefully curated to include a diverse range of images with various objects. The preprocessing and augmentation techniques applied to the dataset helped in improving the generalization capability of the model, making it more robust against different environmental conditions and variations in the drone footage.

In addition to real-time object detection, the project also involved the development of a user-friendly interface using PyQt5. This interface allows users to load video files or live camera feeds and process them using the trained model. The processed video can then be reviewed, replayed, or downloaded, providing users with a comprehensive tool for object detection and analysis.

System Architecture Diagram

This report explains the system architecture developed for the AI-based drone footage foreign object detection project. The diagram below illustrates the overall structure of the system and the data flow between its components.



User Interface

The user interface (UI) allows users to upload video files or real-time footage and run the selected model. The UI handles the input data from the user and appropriately routes it to the model and video/footage components. Additionally, the UI displays the processed results from the model back to the user.

Model

The model component represents the machine learning model selected or uploaded by the user. This model processes the video or live footage data provided by the user interface and performs foreign object detection. The model then sends the processed results back to the user interface for display to the user.

Video or Real-time Footage

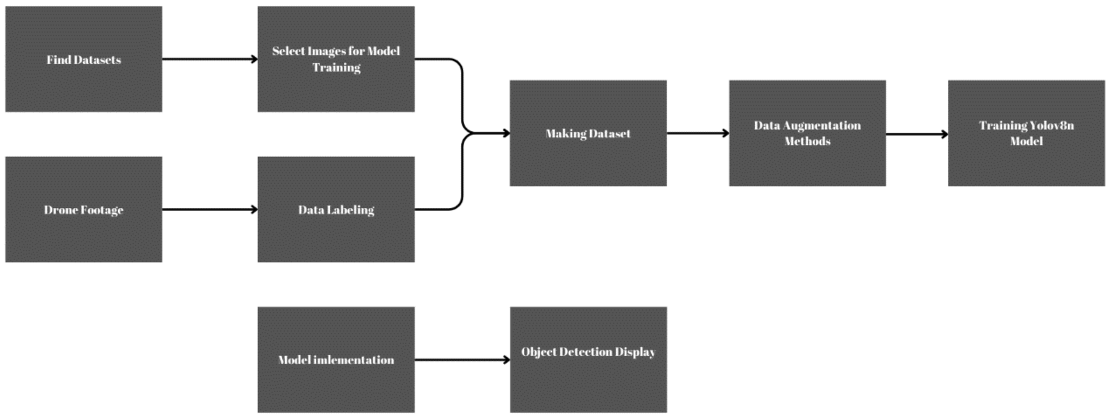
This component represents the video files or real-time camera footage uploaded or selected by the user. The data is sent from the user interface to the model component for processing.

Processing Flow

1. **User Input:** The user selects a video file or live footage source and loads the required model through the UI.
2. **Data Transfer:** The UI sends the selected video/footage and model to the respective components to initiate processing.
3. **Model Processing:** The model processes the provided video or live footage data and detects foreign objects.
4. **Result Display:** The processed results from the model are sent back to the UI and displayed to the user.

Flowchart

The system architecture for AI-based foreign object detection in drone footage involves several key steps. Initially, relevant datasets are gathered and images for model training are selected. Simultaneously, drone footage is collected and labeled to identify foreign objects. These datasets are combined and enhanced through data augmentation techniques to create a robust training set. The Yolov8n model is then trained using this dataset. Once trained, the model is implemented into the system for real-time object detection. Finally, the detected objects are displayed on the drone footage, providing a clear and intuitive interface for users.



Methods Used

This section explains the methods and technologies used in the project:

- **Image Processing:** The OpenCV library was used.
- **Artificial Intelligence:** The YOLO (You Only Look Once) model was preferred.
- **Programming Language:** Developed using Python.
- **Data Labeling:** The Roboflow platform was used.
- **User Interface:** An interface developed with PyQt5 was used.
- **Model:** The YOLOv8n model was used.
- **Model Training:** Conducted on the Google Colab platform.

Drone Footage

In my project, I collaborated with Tetra Company and used drone footage obtained from their shoots. Approximately 14 minutes of footage was obtained from the shoots conducted with Tetra. This footage was edited, with part of it used for testing to demonstrate whether the system works at the end of the project, and the remaining part allocated for model training.

My own recordings were conducted in the later stages of the project. However, due to the low quality of the drone, the clarity and quality of the footage were not as desired. About 5 minutes of footage was obtained from these recordings, and this footage was divided into two parts: one to be shown as a demo at the end of the project and the other for model training.

Dataset

During the creation of the dataset, the labeling process was done using Roboflow. Three types of labels were used: box, pedestrian, and dog. A total of 824 images from the drone footage shot with the company were labeled. Additionally, 152 images from my own drone footage were labeled. Moreover, 310 labeled images from other datasets were also used, making the total dataset size 1286 images. All the datasets I created included labels from the footage shot with the company, while some datasets also included images from other datasets and my own drone footage. Dataset set to 80% train 20% valid.

Dataset Preprocessing/Augmentation Procedures

To train different models, I used various preprocessing and augmentation methods in the datasets I created. These methods are crucial for enhancing the generalization capabilities of the models and artificially increasing the diversity of the training data. Initially, auto-orientation was applied to all images in the dataset. This step ensures the correct alignment of images based on their EXIF data, guaranteeing that all images are properly oriented. Then, the images were resized to 960x640 pixels. This standardization ensures that the dimensions of the images in the dataset are consistent and compatible with the model's input size.

In some datasets, specific horizontal and vertical regions (ranging from 10% to 90%) of the images were statically cropped. This process removes unnecessary background and highlights important features. Contrast adjustment was also performed to enhance the contrast values of the images. This process, done using the contrast stretching method, helps make features in the images more distinct. Additionally, some images were tiled into smaller pieces in a two-row and two-column format. This technique is beneficial for emphasizing local features. Furthermore, filtering was applied to ensure that the dataset contains images with at least 90% annotation.

Various augmentation methods were applied to increase the diversity of the dataset. Three outputs were generated for each training example. Images were flipped horizontally and vertically. These flips help the model learn invariant features. The images were also rotated between -10 degrees and +10 degrees. This allows the model to recognize objects from different angles. Shearing up to 10 degrees horizontally and vertically was also applied. Shearing changes the perspective of the images and provides different viewpoints. Additionally, the exposure of the images was adjusted between -10% and +10%. This adjustment simulates different lighting conditions.

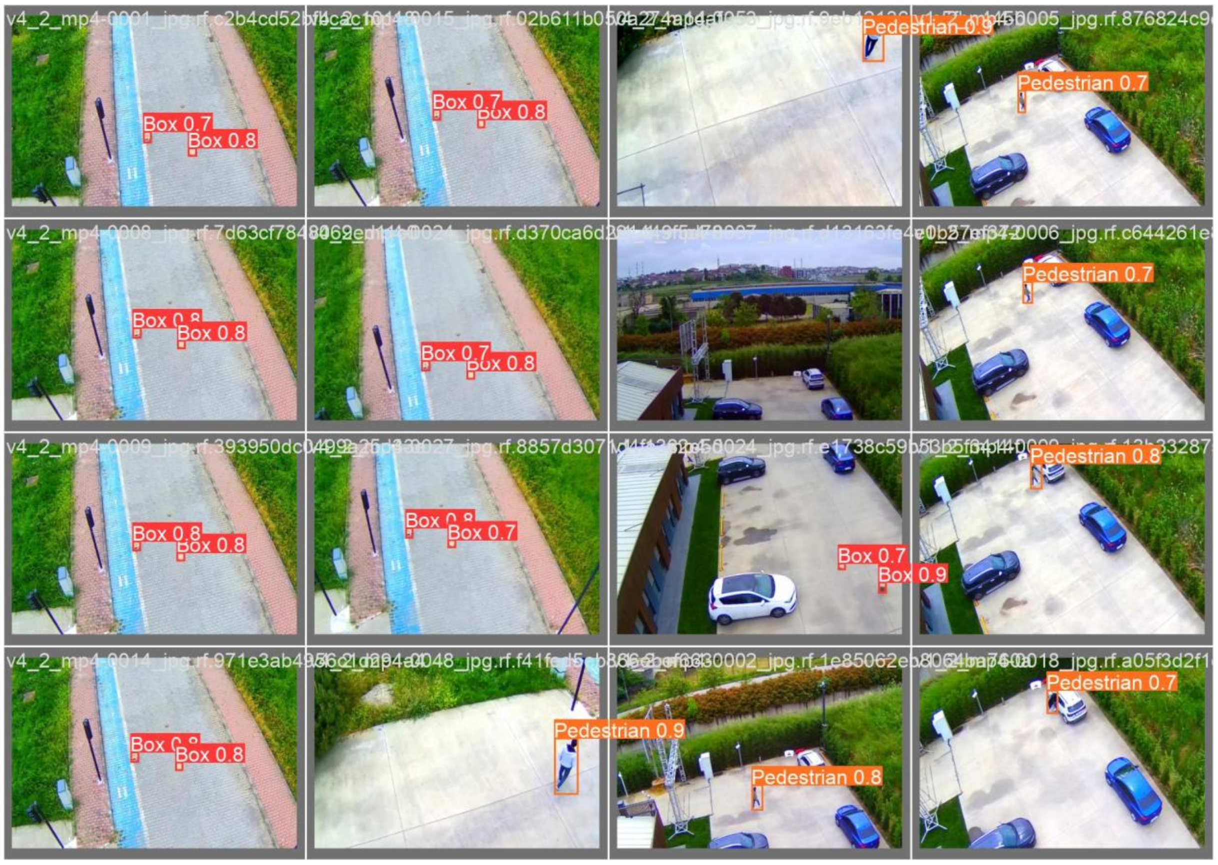
In some datasets, additional transformations such as rotation, shear, and blur were applied to specific bounding boxes. These transformations include rotating the boxes between -5 degrees and +5 degrees, shearing up to 5 degrees horizontally and vertically, and blurring up to 1 pixel. These augmentations make the model robust against minor distortions in object detection tasks. Color adjustments were also made, changing the hue between -20 degrees and +20 degrees, brightness between -20% and +20%, and adding 2% noise to the images. A mosaic effect was also applied, blending multiple images together. These color adjustments and the mosaic effect simulate different environmental conditions and increase the diversity of the dataset.

While creating the datasets, I used images provided by the company as well as images from different sources. Some images were taken from other datasets, and some were from footage I captured with my own drone. These diverse sources enrich the dataset and ensure the model's success across a wider range. By using these methods, I prepared multiple datasets for model training.

Model Training with YOLOv8n

During the model training process, I used the YOLOv8 model, which is optimized for object detection. I chose the smaller YOLOv8n model because it requires less computational power, making it more suitable for a real-time simulation system.

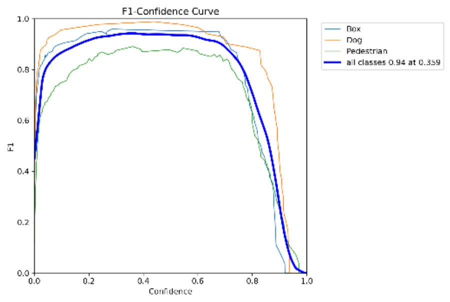
During the training process, the model repeatedly worked on the dataset to learn to recognize objects such as boxes, pedestrians, and dogs. I optimized the hyperparameters with 10 epochs, image sizes of 512 pixels, a batch size of 16, and a learning rate of 0.01. Additionally, the default hyperparameters of the YOLOv8 model include momentum, weight decay, and data augmentation techniques.



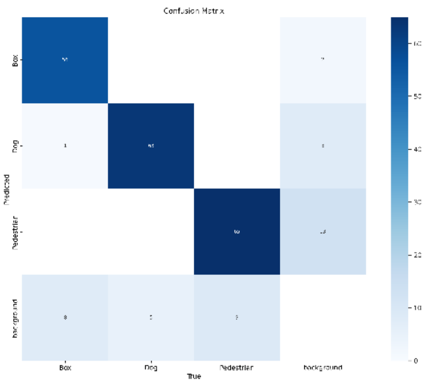
Model Evaluating

Various models were trained and evaluated, and their performance was compared using a test set. This process provided insights into which model performed better. During this work, it was observed that some models, although yielding good results, tended to misclassify square objects as boxes. Some models were more stable but failed to detect certain objects, and others did not detect dogs at all. Based on these model comparisons, one model was selected for inclusion in the report. Below are some statistics related to this selected model:

- The F1-Confidence Curve demonstrates the performance of the models in terms of the F1 score against confidence levels for different classes: Box, Dog, and Pedestrian. The overall F1 score for all classes reaches 0.94 at a confidence level of 0.359, indicating high precision and recall at this specific confidence threshold.



- When examining the overall performance of the model, as shown in the confusion matrix, high accuracy rates are observed in the box and dog classes. However, there are some confusions and misclassifications in the pedestrian and background classes. These confusions suggest that the model needs further data and improvements. While the precision and accuracy values of the model are generally promising, there are noticeable misclassifications among certain classes.



PyQt5 Application: Video and Live Camera Processing

This project is a PyQt5 application that allows users to process video files and live camera feeds using the YOLO (You Only Look Once) model. The application consists of three main pages: video and model loading page, video processing page, and live camera processing page.

General Operation of the Application

1. **Video and Model Loading Page:**
   - On this page, the user can select a video file and a YOLO model.
   - Once the video and model are selected, the user can proceed to the processing page or the live camera page.
2. **Video Processing Page:**
   - The selected video is processed frame by frame using the YOLO model.
   - The detected objects in each frame are displayed on the screen.
   - The user can replay the processed video or download it to their computer.
3. **Live Camera Processing Page:**
   - The user can process the live feed from their computer's camera using the YOLO model in real-time.
   - Detection results are displayed immediately on the screen.

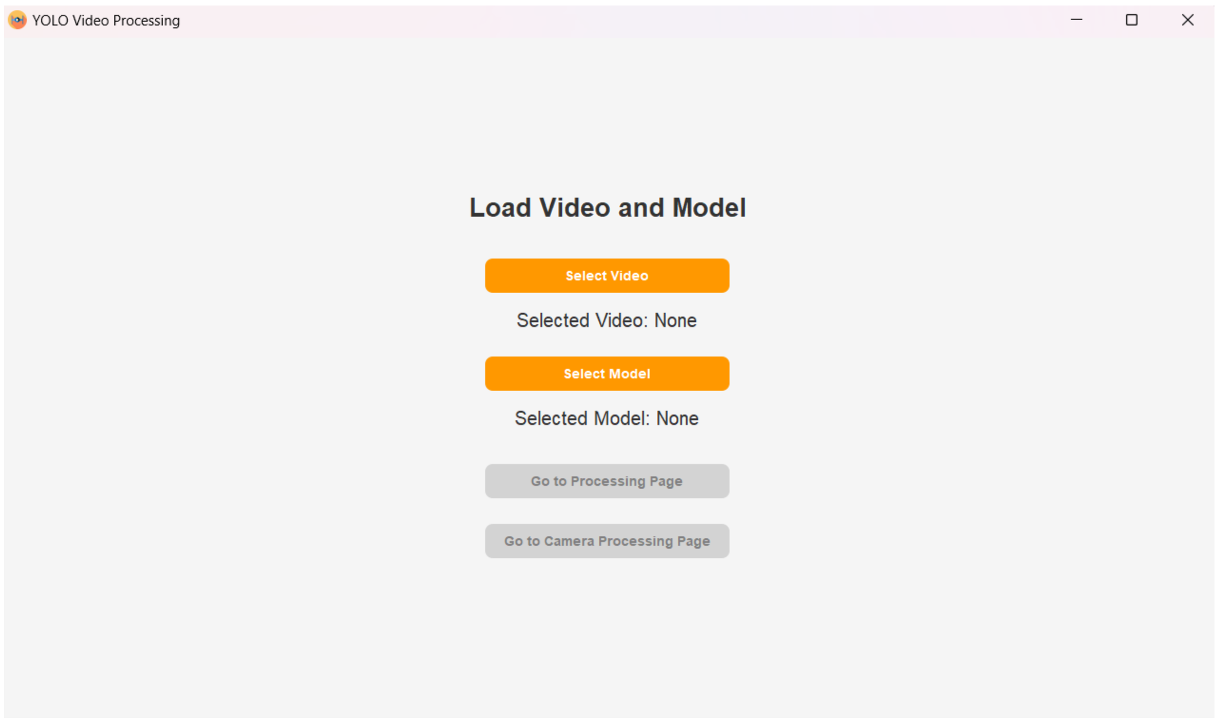
Page Details

1. Video and Model Loading Page

On this page, the user uses buttons to select the video and model files. Once the video and model are successfully loaded, the buttons to proceed to the processing page or the live camera page are enabled.

```
def init_first_page(self):
    # Creating components for the first page
    self.label = QLabel("Load Video and Model", self.first_page)
    self.video_button = QPushButton("Select Video", self.first_page)
    self.video_path_label = QLabel("Selected Video: None", self.first_page)
    self.model_button = QPushButton("Select Model", self.first_page)
    self.model_path_label = QLabel("Selected Model: None", self.first_page)
    self.proceed_button = QPushButton("Go to Processing Page",
self.first_page)
```

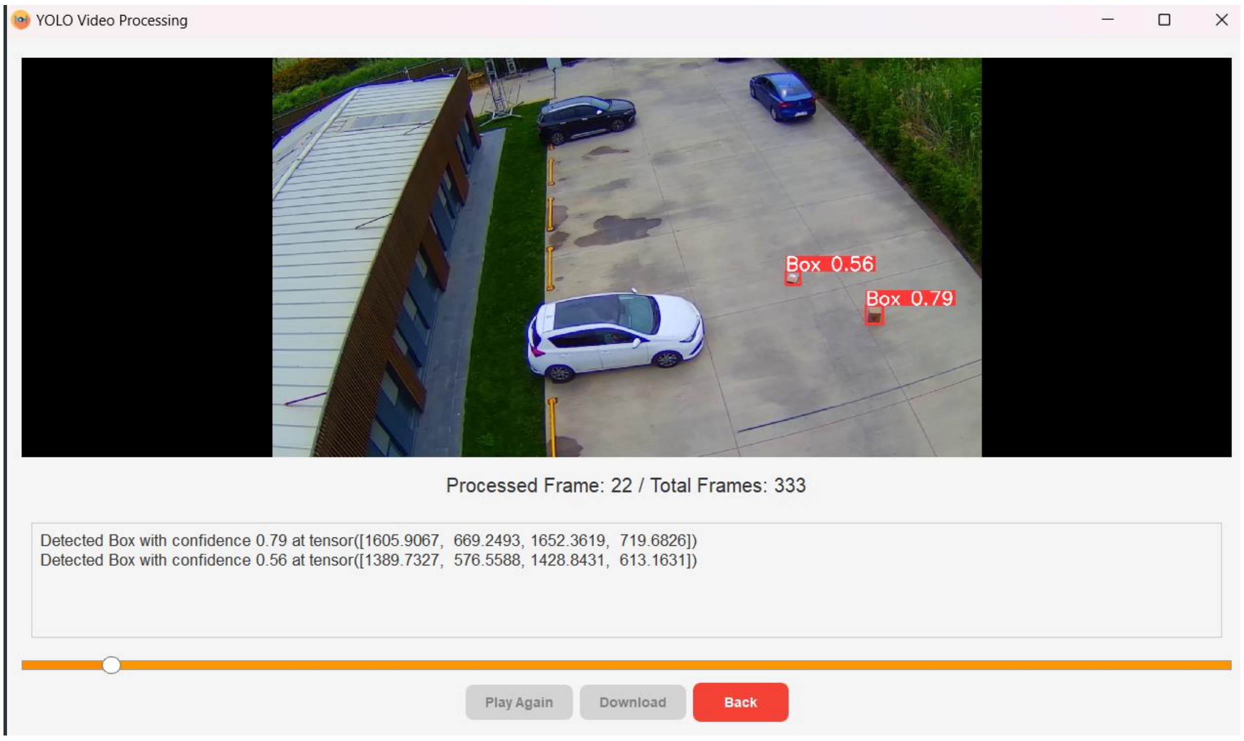
```
    self.camera_button = QPushButton("Go to Camera Processing Page",
self.first_page)

    # Connecting button click events
    self.video_button.clicked.connect(self.openFileNameDialog)
    self.model_button.clicked.connect(self.openModelDialog)
    self.proceed_button.clicked.connect(self.go_to_processing_page)
    self.camera_button.clicked.connect(self.go_to_camera_page)

    # Creating the layout for the page
    button_layout = QVBoxLayout()
    button_layout.addWidget(self.video_button)
    button_layout.addWidget(self.video_path_label)
    button_layout.addWidget(self.model_button)
    button_layout.addWidget(self.model_path_label)
    button_layout.addWidget(self.proceed_button)
    button_layout.addWidget(self.camera_button)

    main_layout = QVBoxLayout(self.first_page)
    main_layout.addWidget(self.label)
    main_layout.addLayout(button_layout)
    main_layout.setContentsMargins(20, 20, 20, 20)
```



2. Video Processing Page

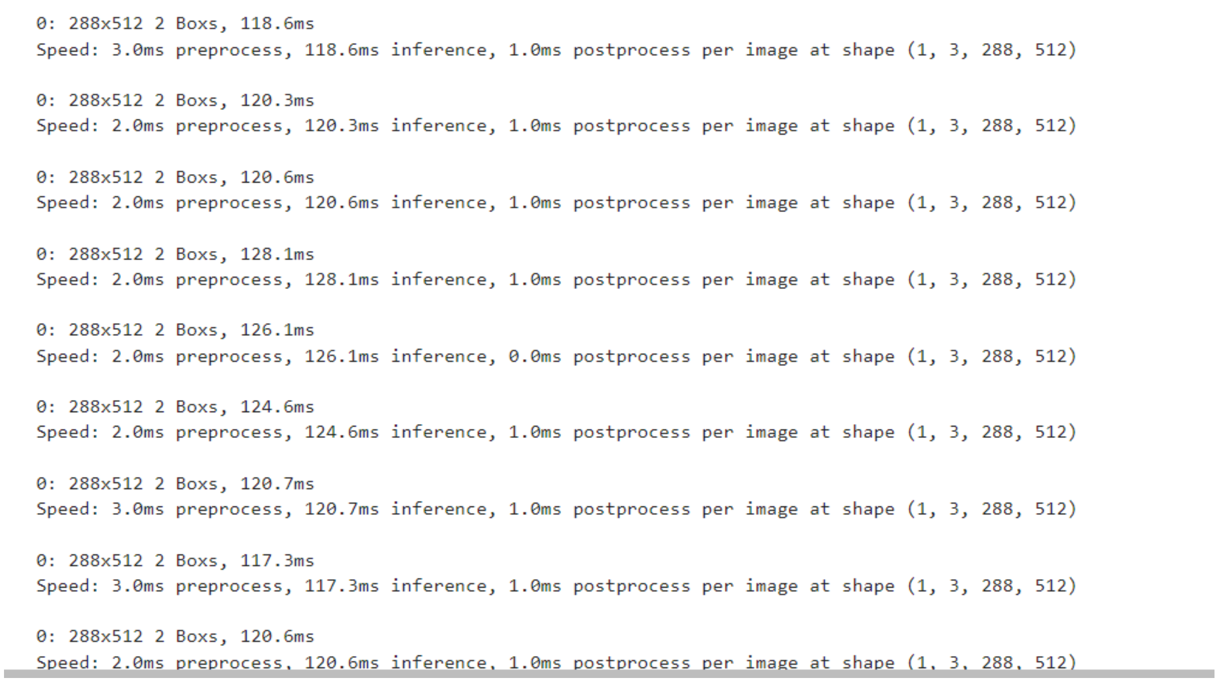
On this page, the selected video is processed using the YOLO model. The objects detected in each frame are displayed on the screen. The user can replay the processed video or download it.

```
def init_second_page(self):
    # Creating components for the video processing page
    self.video_label = ClickableLabel(self.second_page)
    self.frame_info_label = QLabel(self.second_page)
    self.detection_results_text = QLabel(self.second_page)
    self.seek_bar = QSlider(Qt.Horizontal, self.second_page)
    self.play_again_button = QPushButton("Play Again", self.second_page)
    self.download_button = QPushButton("Download", self.second_page)
    self.back_button = QPushButton("Back", self.second_page)

    # Connecting button click events
    self.play_again_button.clicked.connect(self.play_processed_video)
    self.download_button.clicked.connect(self.download_processed_video)
    self.back_button.clicked.connect(self.back_to_loading)

    # Creating the layout for the page
    video_layout = QVBoxLayout()
    video_layout.addWidget(self.video_label)
    video_layout.addWidget(self.frame_info_label)
    video_layout.addWidget(self.detection_results_text)
    video_layout.addWidget(self.seek_bar)

    button_layout = QHBoxLayout()
    button_layout.addWidget(self.play_again_button)
    button_layout.addWidget(self.download_button)
    button_layout.addWidget(self.back_button)

    main_layout = QVBoxLayout(self.second_page)
    main_layout.addLayout(video_layout)
    main_layout.addLayout(button_layout)
    main_layout.setContensMargins(20, 20, 20, 20)
```



Processed Frame: 22 / Total Frames: 333

Detected Box with confidence 0.79 at tensor([1605.9067, 669.2493, 1652.3619, 719.6826])
Detected Box with confidence 0.56 at tensor([1389.7327, 576.5588, 1428.8431, 613.1631])

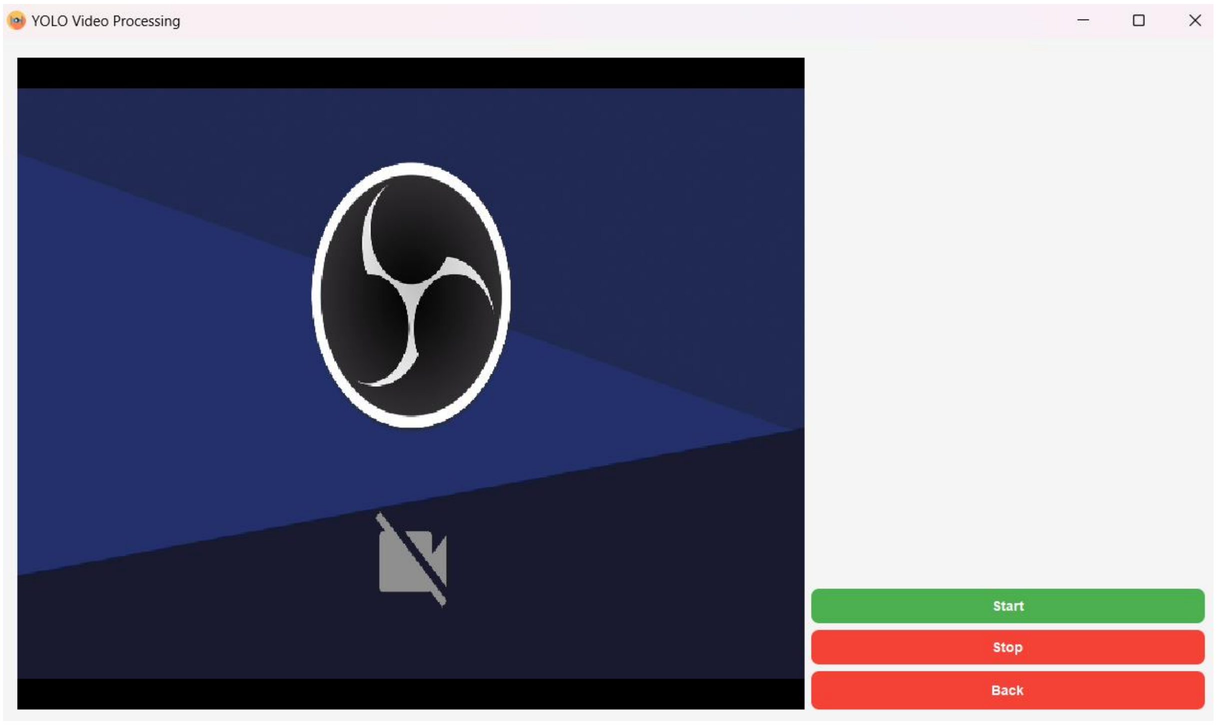


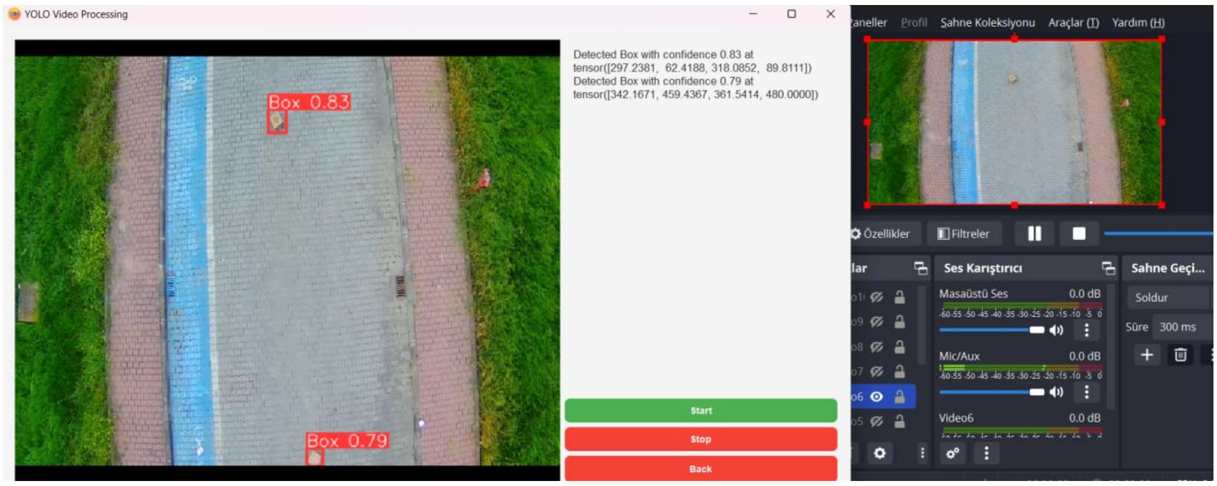
```
0: 288x512 2 Boxs, 118.6ms
Speed: 3.0ms preprocess, 118.6ms inference, 1.0ms postprocess per image at shape (1, 3, 288, 512)

0: 288x512 2 Boxs, 120.3ms
Speed: 2.0ms preprocess, 120.3ms inference, 1.0ms postprocess per image at shape (1, 3, 288, 512)

0: 288x512 2 Boxs, 120.6ms
Speed: 2.0ms preprocess, 120.6ms inference, 1.0ms postprocess per image at shape (1, 3, 288, 512)

0: 288x512 2 Boxs, 128.1ms
Speed: 2.0ms preprocess, 128.1ms inference, 1.0ms postprocess per image at shape (1, 3, 288, 512)

0: 288x512 2 Boxs, 126.1ms
Speed: 2.0ms preprocess, 126.1ms inference, 0.0ms postprocess per image at shape (1, 3, 288, 512)

0: 288x512 2 Boxs, 124.6ms
Speed: 2.0ms preprocess, 124.6ms inference, 1.0ms postprocess per image at shape (1, 3, 288, 512)

0: 288x512 2 Boxs, 120.7ms
Speed: 3.0ms preprocess, 120.7ms inference, 1.0ms postprocess per image at shape (1, 3, 288, 512)

0: 288x512 2 Boxs, 117.3ms
Speed: 3.0ms preprocess, 117.3ms inference, 1.0ms postprocess per image at shape (1, 3, 288, 512)

0: 288x512 2 Boxs, 120.6ms
Speed: 2.0ms preprocess, 120.6ms inference, 1.0ms postprocess per image at shape (1, 3, 288, 512)
```

3. Live Camera Processing Page

On this page, the live feed from the user's camera is processed using the YOLO model. Detection results are displayed in real-time on the screen.

```
def init_third_page(self):
    # Creating components for the live camera processing page
    self.camera_view = QLabel(self.third_page)
    self.detection_results_camera = QLabel(self.third_page)
    self.camera_start_button = QPushButton("Start", self.third_page)
    self.camera_stop_button = QPushButton("Stop", self.third_page)
    self.camera_back_button = QPushButton("Back", self.third_page)

    # Connecting button click events
    self.camera_start_button.clicked.connect(self.start_camera)
    self.camera_stop_button.clicked.connect(self.stop_camera)
    self.camera_back_button.clicked.connect(self.back_to_loading_from_camera)

    # Creating the layout for the page
    video_layout = QVBoxLayout()
    video_layout.addWidget(self.camera_view)

    detection_layout = QVBoxLayout()
    detection_layout.addWidget(self.detection_results_camera)

    button_layout = QVBoxLayout()
    button_layout.addWidget(self.camera_start_button)
    button_layout.addWidget(self.camera_stop_button)
    button_layout.addWidget(self.camera_back_button)

    main_layout = QHBoxLayout(self.third_page)
    main_layout.addLayout(video_layout, 2)
    main_layout.addLayout(detection_layout, 1)
    main_layout.addLayout(button_layout, 1)
    main_layout.setContentsMargins(20, 20, 20, 20)
```





```
Detected Box with confidence 0.83 at
tensor([297.2381,  62.4188, 318.0852,  89.8111])
Detected Box with confidence 0.79 at
tensor([342.1671, 459.4367, 361.5414, 480.0000])
```

These pages are switched using `QStackedWidget`. This widget allows easy transition between different pages. Each page has its components and layouts defined separately, and the pages can be switched based on user interactions.

Core Functions

- **openFileNameDialog**: Opens a file dialog to select the video file.
- **openModelDialog**: Opens a file dialog to select the YOLO model file.
- **load_model**: Loads the selected model file.
- **load_video**: Loads the selected video file.
- **check_ready_to_proceed**: Enables the buttons to proceed to the processing or camera page once both the video and model are loaded.
- **go_to_processing_page**: Switches to the video processing page.
- **go_to_camera_page**: Switches to the live camera processing page.
- **start_processing**: Starts processing the video.
- **update_frame**: Processes each frame of the video and displays the result.
- **play_processed_video**: Replays the processed video.
- **download_processed_video**: Downloads the processed video.
- **back_to_loading**: Returns to the video and model loading page.
- **start_camera**: Starts processing the live camera feed.
- **update_camera_frame**: Processes each frame of the live camera feed and displays the result.
- **stop_camera**: Stops processing the live camera feed.
- **print_detection_results**: Displays the detection results for the video.
- **print_camera_detection_results**: Displays the detection results for the live camera feed.

Discussion

In this project, the developed system achieves a frame rate of 7-8 FPS (Frames Per Second) for foreign object detection in drone footage. However, these results were obtained using a CPU (Central Processing Unit). It is anticipated that using a GPU (Graphics Processing Unit) would significantly increase the processing speed, resulting in higher FPS values. The use of a GPU, with its parallel processing capabilities, could greatly accelerate object detection and provide more suitable performance for real-time applications.

Other areas for potential improvement in the project include the use of different models and the expansion of the dataset. Currently, the YOLOv8n model is used, but other deep learning models such as Faster R-CNN and SSD (Single Shot MultiBox Detector) could also be explored. Comparing the performance and accuracy of these models would help in selecting the most suitable one.

During the dataset creation phase, additional images could have been captured. However, due to the unavailability of a drone in the later stages of the project, further footage could not be obtained. This limitation may affect the diversity of the dataset and the overall performance of the model. In future work, more footage can be collected under various weather conditions and environmental settings to expand the dataset and improve the model's generalization capabilities.

Moreover, more classes could be included during the labeling process to enhance the model's ability to recognize a wider range of objects. Currently, the dataset includes labels for boxes, pedestrians, and dogs. Adding classes for vehicles, bicycles, and traffic signs could improve the model's capacity to identify various objects on roads.

Lastly, based on feedback and user experiences obtained during the project, some improvements could be made to the PyQt5 interface. Making the interface more user-friendly and providing more customization options could positively impact the user experience.

In summary, there are numerous opportunities to enhance the project both technically and practically. Utilizing more powerful hardware, experimenting with different models, and expanding the dataset with diverse and high-quality images can significantly improve the overall performance and accuracy of the system.

CONCLUSION

In this project, a system capable of real-time object detection on drone footage has been developed. Various image processing and artificial intelligence techniques were utilized to process images captured by drone cameras and identify specific objects. Particularly, the YOLO (You Only Look Once) model was employed to identify objects such as boxes, humans, and dogs.

During the development of the project, data labeling was carried out using the Roboflow platform, and a user-friendly interface was created with PyQt5. This interface allows users to process both video files and live camera feeds. Model training was performed on the Google Colab platform, using an optimized version of the YOLOv8n model.

The dataset used for model training was obtained from drone footage collected in collaboration with Tetra Company and various other sources. The diversity and quality of the dataset were meticulously curated to enhance the overall success rate of the model. Data processing and augmentation methods were applied to improve the model's generalization ability, ensuring successful operation under different environmental conditions.

During the model training and evaluation process, the performances of different models were compared, and the most successful model was included in the report. The high accuracy rates and good F1 scores of this model demonstrate the overall success of the project. However, the observation of some misclassifications in pedestrian and background classes indicates that the model requires more data and improvements.

In conclusion, the developed system is capable of real-time object detection in drone footage, successfully identifying objects such as boxes, humans, and dogs. This project has made a significant contribution to the fields of drone technology and artificial intelligence, and with future improvements, it holds the potential to serve a wider range of applications.

REFERENCE

https://encord.com/blog/yolo-object-detection-guide/

https://dipankarmedh1.medium.com/real-time-object-detection-with-yolo-and-webcam-enhancing-your-computer-vision-skills-861b97c78993

https://universe.roboflow.com/oas-xolv8/oas-yebo7/browse