Team 7.

Yuldashev Mukhammadyusuf 12194916

1.We collect and label traffic data: The first step is for us to collect traffic data and label it according to the objects present in the image. This labeled data will be used to train the YOLO model. We will be using online labeling tools to make the labeling process more efficient.

1.1. We determine the scope of our dataset: We need to decide which types of traffic signs we want our model to detect. For example, we might want to focus on speed limit signs, stop signs, yield signs, or a combination of different signs.

1.2. We collect the data: We can collect traffic data using a camera to capture images of traffic signs in the wild or by using pre-existing datasets that have been made available online. If we are collecting our own data, we need to take images from different angles and lighting conditions to ensure our model can recognize the signs under various circumstances.

1.3. We label the data: Once we have collected our traffic data, we need to label it with the objects present in each image. We can use online labeling tools, such as Labelbox or Supervise.ly, to draw bounding boxes around the traffic signs in each image and label them with the corresponding class (e.g., "stop sign", "speed limit sign", etc.).

1.4. We check the quality of the labeled data: After labeling our data, we need to check for any labeling errors or inconsistencies. We can use tools such as CVAT or RectLabel to verify that our bounding boxes are accurate and consistent across our dataset. It's important to ensure the quality of our labeled data because it will directly impact the performance of our model.

1.5. We split the data into training, validation, and testing sets: Once we have labeled and verified our data, we need to split it into training, validation, and testing sets. The training set will be used to train our YOLO model, the validation set will be used to evaluate the performance of our model during training and make adjustments as necessary, and the testing set will be used to measure the final performance of our model after training. We can use a common split ratio of 70% for training, 15% for validation, and 15% for testing, but this can be adjusted depending on the size of our dataset and the requirements of our project.

By following these steps, we can collect and label a high-quality dataset for training our YOLO model for traffic sign detection.

2.We train the YOLO model: Once we have collected and labeled our traffic data, we need to train our YOLO model to detect and classify different types of traffic objects such as cars, buses, pedestrians, and most importantly, traffic signs.

2.1. Choose a YOLO model: There are several YOLO models available, each with varying levels of accuracy and speed. For traffic sign detection, we can use the YOLOv3 or YOLOv4 models, which are both capable of detecting multiple objects in an image.

2.2. Configure the training parameters: We need to set the training parameters such as the learning rate, batch size, and number of epochs. These parameters will determine how quickly the model learns and how well it generalizes to new data.

2.3. Initialize the model with pre-trained weights: We can initialize the YOLO model with pre-trained weights on a large dataset such as ImageNet. This will help the model to learn faster and improve its accuracy.

2.4. Train the model: We can train the YOLO model using our labeled traffic dataset. During training, the model will learn to detect and classify different types of traffic objects such as cars, buses, pedestrians, and traffic signs. We can use frameworks such as TensorFlow or PyTorch to train the model on a GPU for faster processing.

2.5. Evaluate the model: After training, we need to evaluate the performance of the model on our validation set. We can use metrics such as precision, recall, and F1 score to measure how well the model performs at detecting traffic signs.

We convert the YOLO model to TensorFlow: Once our YOLO model is trained, we need to convert it to TensorFlow format. This step is important because TensorFlow is one of the most popular deep learning frameworks used for deploying models in production environments.

3.1. Installing the required dependencies: To convert our YOLO model to TensorFlow format, we need to install the required dependencies. We can use the pip package manager to install the dependencies. The dependencies include the TensorFlow 2.x package and the TensorRT package.

3.2. Downloading the conversion script: We need to download the "convert.py" script from the YOLOv7 repository. This script is used to convert the YOLO model to TensorFlow format.

3.3. Converting the YOLO model: We can now use the "convert.py" script to convert our YOLO model to TensorFlow format. We need to specify the location of our YOLO model file and the output directory where the converted TensorFlow model will be saved.

3.4. Testing the converted model: Once the conversion process is complete, we need to test the converted TensorFlow model to make sure it's working correctly. We can use the TensorFlow Lite Interpreter or TensorFlow Serving to deploy our model and test its performance.

4.We optimize the TensorFlow model for TensorFlow Lite: To use our YOLO model in ROS, we need to optimize it for TensorFlow Lite. TensorFlow Lite is a lightweight version of TensorFlow designed for mobile and embedded devices. By optimizing our model for TensorFlow Lite, we can make it more efficient and faster to run on devices with limited resources.

4.1. Installing the required dependencies: To optimize our TensorFlow model for TensorFlow Lite, we need to install the required dependencies. We can use the pip package manager to install the dependencies. The dependencies include the TensorFlow 2.x package, the TensorFlow Lite package, and the TensorBoard package.

4.2. Converting the TensorFlow model to TensorFlow Lite format: We can use the TensorFlow Lite Converter to convert our TensorFlow model to TensorFlow Lite format. We need to specify the location of our TensorFlow model file and the output directory where the converted TensorFlow Lite model will be saved.

4.3. Quantizing the TensorFlow Lite model: We can use the post-training quantization technique to quantize our TensorFlow Lite model. Quantization is a technique that reduces the precision of the weights and biases in the model to 8-bit or lower. This can significantly reduce the size of the model and make it more efficient to run on mobile and embedded devices.

4.4. Testing the optimized model: Once the optimization process is complete, we need to test the optimized TensorFlow Lite model to make sure it's working correctly. We can use the TensorFlow Lite Interpreter to deploy our model and test its performance.

5.We convert the TensorFlow model to TensorFlow Lite: Once our TensorFlow model is optimized for TensorFlow Lite, we can convert it to TensorFlow Lite format using the "tflite\_convert" command provided by TensorFlow.

5.1. Installing the required dependencies: To convert our TensorFlow model to TensorFlow Lite format, we need to install the required dependencies. We can use the pip package manager to install the dependencies. The dependencies include the TensorFlow 2.x package and the TensorFlow Lite package.

5.2. Converting the optimized TensorFlow model to TensorFlow Lite format: We can use the "tflite\_convert" command to convert our optimized TensorFlow model to TensorFlow Lite format. We need to specify the location of our optimized TensorFlow model file, the output directory where the converted TensorFlow Lite model will be saved, and the target device type for which the model will be optimized.

5.3. Testing the converted model: Once the conversion process is complete, we need to test the converted TensorFlow Lite model to make sure it's working correctly. We can use the TensorFlow Lite Interpreter to deploy our model and test its performance.

6.We measure the distance by the size of the bounding boxes: To detect the distance of the traffic signs from the camera, we can use the size of the bounding boxes of the detected signs. The size of the bounding boxes changes as the signs move closer or farther away from the camera.

6.1. Defining a variable to store the size of the bounding box: We define a variable to store the size of the bounding box of the detected traffic sign. We can use the height or width of the bounding box to measure the distance of the sign from the camera.

6.2. Mapping the size of the bounding box to the distance of the sign: We can map the size of the bounding box to the distance of the sign using a calibration curve. The calibration curve relates the size of the bounding box to the distance of the sign. We can obtain this curve through experiments.

6.3. Setting a threshold distance: We set a threshold distance beyond which the car system will receive a request to act. This threshold distance corresponds to a certain size of the bounding box. As the sign moves closer to the camera and the size of the bounding box crosses this threshold, the car system will receive a request to act.

6.4. Sending a request to the car system: Once the size of the bounding box crosses the threshold, we send a request to the car system to act accordingly. This can be achieved by integrating the ROS traffic sign detection system with the car system.