Traffic Sign Detection and Recognition using Deep Learning (YOLOv5)

Rahul Bhakad[1] Shantanu Bawankule[2] Kunal Gaikwad[3] Prof. P.R.Dongare[4]
[1,2,3,4]Department of Computer Engineering
[1,2,3,4]Sinhgad Academy of Engineering, Savitribai Phule Pune University, Pune, Maharashtra, India

Abstract— Traffic signs are important to ensure smooth traffic flow without mishaps. Traffic symbols are the pictorial representations having different necessary information required to be understood by the driver. Traffic signs in front of the vehicle are ignored by the drivers and this can lead to catastrophic accidents. To tackle this problem, we implemented a deep learning model based on YOLO's (You Only Look Once) latest version YOLOv5 in this project. YOLO is one of the fastest object detection algorithms for real-time detection. The goal is to find and classify traffic signs in real-world street settings. This paper explains the various steps to implement the YOLOv5 for detecting and recognizing different types of traffic signs and help the driver for safe navigation.

Keywords: YOLO, Convolutional Neural Network, CNN, Google Colab, Roboflow, Deep Learning, Machine Learning, Object Detection, Text-To-Speech Conversion, Pyttsx3

I. INTRODUCTION

Every day, over 400 traffic accidents occur in India, according to official statistics. Road signs aid in the prevention of traffic accidents, assuring the safety of both drivers and pedestrians. Furthermore, traffic signals ensure that road users follow specified regulations, reducing the chances of traffic offenses. The usage of traffic signals also helps with route guidance. All road users, including automobiles and pedestrians, should prioritize road signals. For a variety of causes, such as focus issues, tiredness, and sleep deprivation, we overlook traffic signs. Poor vision, the impact of the outside world, and environmental circumstances are all factors that contribute to ignoring the signs.

Using state of the art technology, YOLOv5 ('You only look once') we can automate the task of detecting and recognizing traffic sign images and help drivers for easy navigation. It is an object detection algorithm that divides images into a grid system. Each cell in the grid is responsible for detecting objects within itself. YOLOv5 is one of the most famous object detection algorithms due to its speed and accuracy. YOLO is extraordinarily rapid, with a mean average precision (mAP) that is more than double that of conventional real-time systems This technology will examine images acquired by a car's front-facing camera in real time and assist the driver by raising concerns to him or her through audio output or vehicle navigation display. The approach "looks once" at the image, in the sense that it produces predictions after only one forward propagation through the neural network.

As a result, we can conclude that one-stage approaches are effective for obtaining faster results with good precision. Because traffic sign detection must be done in real time, the YOLOv5 would be a better choice.

II. IMPLEMENTATION STEPS

A. Collecting Dataset

Our dataset contains a total of 8946 images of traffic signs, our dataset is divided into two parts, one is for training and another one is for testing. There are a total of 7800 images for training and 1146 images for testing. Images are annotated for identifying location and the class of traffic signs in the image. The dataset contains natural traffic signs pictures, shot on different kinds of streets (roadway, rustic, metropolitan) during the daytime, around sunset and different weather patterns which makes traffic signs suffer from difference in orientation, light conditions or occlusions. There are a total 36 classes of traffic signs e.g., Stop, No-Parking, U-turn, 50km/h etc. Hence our model can identify different types of traffic signs on the road and thus making our model very reliable.

B. Annotating Our Training Images

Before training and testing our object detection model we will need a training and testing dataset. Training and testing dataset contains annotated images which is required to train and test the object detection model before using it for prediction. We will use Roboflow which is an online tool to label our traffic sign dataset and then export the dataset in YOLOv5 format. Roboflow Annotate is a simple web-based tool for managing and labeling your images and exporting them in YOLOv5's annotation format. Following are the steps to create and import the dataset in YOLOv5 format.

1) We need to sign up at https://roboflow.com/.
2) On the Roboflow Free Tier, all data is added via the Web UI. After logging in to account Select "Create New Project". This will trigger a model to pop up with an option to upload data.
3) Then select "Upload Your Own Data".
4) Upload your dataset, label the unannotated images, then generate and export a version of your dataset in YOLOv5 Pytorch format.
5) It will generate a code snippet, then copy the snippet into your training script or notebook to download your dataset.

C. Training Model

We used Google's Colab Notebook to train our model. Colab is a browser-based tool that allows users to write and run python code. It's great for machine learning, data analysis, and education. More precisely, Colab is a hosted Jupyter notebook service that requires no installation and provides free access to computational resources and GPUs. Colab has a GPU with a capacity of 12GB. The types of GPUs accessible for customers to use in Colab change over time. This is frequently required for Colab to be able to provide free

access to certain resources. Nvidia K80, T4 and P100 GPUs are available from Colab.

1) First step is to create a colab account by signing up from a google account.
2) We have to create a new notebook.
3) Then we have to set up an environment for our notebook to python 3.8 and use gpu to make the training process faster.
4) Then we first clone the YOLOv5 repository and install dependencies. This will set up our programming environment to be ready to run the traffic sign detection training and inference commands. Below are the following commands to clone the YOLOv5 repository.
   !git clone https://github.com/ultralytics/yolov5
   %cd yolov5
   %pip install -qr requirements.txt
   %pip install -r roboflow
5) Then to export the dataset in a colab notebook we will copy and run the code snippet which was generated while importing the dataset in roboflow. The export creates a YOLOv5 .yaml file called data.yaml specifying the location of a YOLOv5 images folder, a YOLOv5 labels folder and information on our custom classes.
   from roboflow import Roboflow
   rf=Roboflow(api_key="YOUR API KEY HERE")
   project=rf.workspace().project("YOUR PROJECT")
   dataset=project.version("VERSION").download("yolov5")
6) To start training we use training command with the following options:
   – img: define input image size.
   – batch: determine batch size.
   – epochs: define the number of training epochs.
   – data: set the path to our yaml file.
7) We define the location of train, val and test, the number of classes (nc) and the names of the classes. We use a batch size of 32, image size of 640 and train for 1000 epochs. Below is the command to start the training process.
   !python train.py --img 640 --batch 32 --epochs 1000 --data {dataset.location}/data.yaml --weights yolov5s.pt --cache

III. EVALUATE YOLOV5 PERFORMANCE

Model was trained for 1000 epochs and achieved mAP@0.5 (Mean Average Precision) equal to 91.75%. After completing the training results.png is generated which includes parameters like Precision, Recall, Objectness, Box Classification. All are shown in below figures.

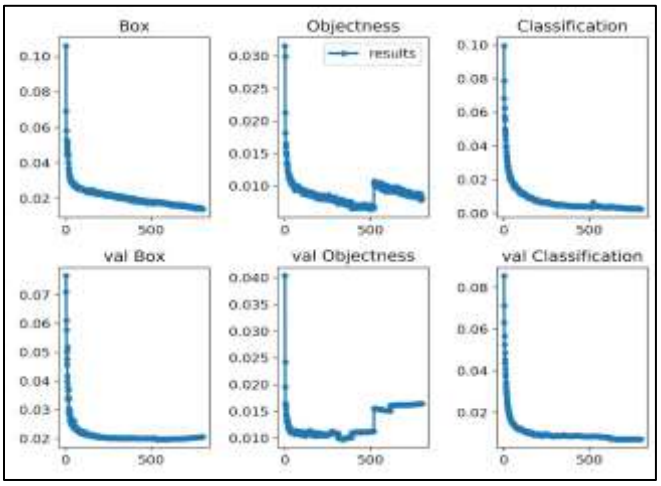

Fig 1: Box, Objectness, Classification.

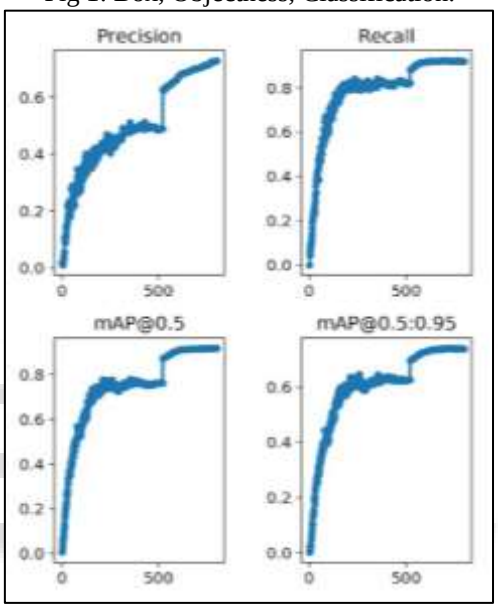

Fig. 2: Precision, Recall.

Loss functions:
For the training set:

- Box: loss due to a box prediction not perfectly covering an object.
- Objectness: loss due to an incorrect box-object IoU prediction.
- Classification: loss due to mispredictions '1' for the correct classes and '0' for all the other classes for the object in that box.

For the valid set (the same loss functions as for the training data):

- val Box
- val Objectness
- val Classification

Precision & Recall:

- Precision: determines how accurate the predictions are. It is the percentage of your correct predictions.
- Recall: measures how well it locates all the positives.

Accuracy functions:

- mAP (mean Average Precision) calculates a score by comparing the ground-truth bounding box to the

detected box. The higher the score, the better the model's detection accuracy.

- mAP@ 0.5: when IoU is set to 0.5, the AP of all images of each category is calculated, and then the average of all categories is determined.
- mAP@ 0.5.0.95: represents the average mAP at various IoU thresholds (from 0.5 to 0.95 in steps of 0.05).
- IoU (Intersection over Union) measures the overlap between two boundaries. It is used to measure how much the predicted boundary overlaps with the ground truth.
- AP (Average Precision) is a metric to measure the accuracy of object detectors. It calculates the average precision value for recall values ranging from 0 to 1.

IV. INFERENCE

After the training is done, weights are generated which will be used for detection of traffic signs. "best.pt" contains the best performing weights saved during training. But before using the best.pt weight for detection of traffic signs through webcam we need to set up the environment and install dependencies. The source flag defines the source of our detector, for webcam we use 0. The weights flag defines the path of the model which we want to run our detector with. Then the conf flag is the thresholding objectness confidence.

- $ git clone https://github.com/ultralytics/yolov5
- $ cd yolov5
- $ pip install -r requirements.txt
- $ python detect.py --weights runs/train/exp/weights/best.pt --img 416 --conf 0.1 --source 0

V. RESULTS:

Fig. 3: No Stopping Sign Detected.

Fig. 4: Pedestrian Crossing Sign Detected.

Fig. 5: Speed Limit 40 km/h Sign Detected.

Fig. 6: Caution Sign Detected.

VI. WORKING ARCHITECTURE

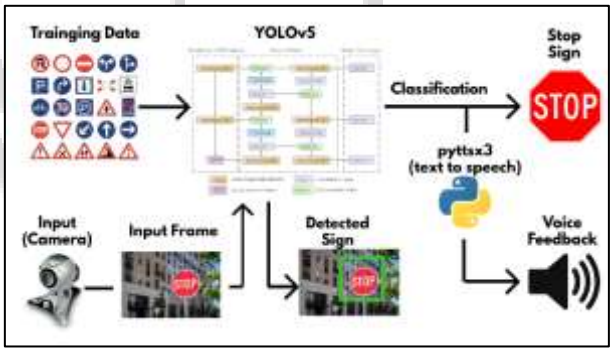

Fig. 7: YOLOv5 Trained Model Detecting Traffic Signs.

1) Training: Model is first trained with the traffic signs dataset.
2) Input data: After training we will be using a camera as an input device to feed video frames to the trained model.
3) Prediction: The trained TSDR model can then process the input frames to detect and recognize the traffic signs and create bounding boxes around the traffic signs with the label of the detected traffic sign.
4) Voice Feedback: The class prediction of the traffic signs detected in every frame will be a string e.g., "stop". By using pyttsx3 we can then convert these strings to speech and through the speaker we can produce the voice feedback of the detected traffic sign.

VII. CONCLUSION

We discussed the steps to implement yolov5 for detecting and recognizing a large number of traffic-sign categories in this paper with the goal of automating traffic-sign recognition and informing the driver via text or audio output. Implementation

steps included acquiring images of traffic signs then annotating those acquired images to create training dataset then training the model on the dataset using the trained model for real life detection. We proposed using a technique called the YOLOv5 algorithm and how it is employed in traffic sign detection. When compared to other object detection approaches such as Fast R-CNN and Faster R-CNN, this technique delivers faster detection results. The system uses a deep network to learn a huge number of categories while also detecting them efficiently and quickly. During the picture acquisition step, the photos will be captured with a camera and the detection will be done using the YOLOv5. When a traffic sign is detected, the system provides a voice alert. This model is best suited when requiring precise and fast traffic sign detection and recognition.

REFERENCES

[1] Github link to Ultralytics YOLOv5 repository- https://github.com/ultralytics/yolov5
[2] Roboflow link for labeling images- https://roboflow.com/annotate
[3] Google Colab link for training model- https://colab.research.google.com
[4] YOLO paper published in 2016 link- https://arxiv.org/pdf/1506.02640