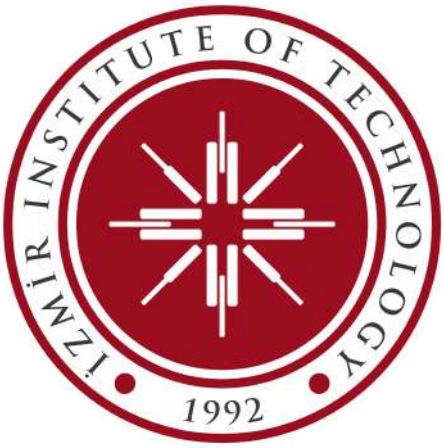
CENG483 Behavioural Robotics
Fall 2023

Project Title: Real-time Traffic Sign and Light Recognition for Advanced Driving Assistance System (ADAS)

Mid-phase Progress Report

December 17, 2023



Group Members

- Boğaçhan Ali Düşgül & 270206051
- Cem Tolga Münyas & 270206042
- Emre Sengir & 270206023
- Harun Durmuş & 270206025
- Ozan Gülbaş & 270206031

Abstract

In this mid-progress report, the structure of the system, sources researched and used, the methodology, problems encountered, solutions founded and developments carried out and the latest observations with interpretations are mentioned. What we did up until this point was well stated. Advancing from its initial state, this project targets creating a robust traffic sign and light detection system for older vehicles lacking Advanced Driver-Assistance Systems (ADAS). We proposed to configure an existed deep learning model for traffic conditions in Turkey, which is combination of several models on the web sources, for identifying and categorizing traffic signs and lights through process. This method confronts challenges in accurate detection and complex classification, yet has shown promising results. The model's significant accuracy in detecting and classifying traffic elements signifies a major step towards applying this technology in real-world settings, enhancing safety features for older vehicles.

1 Introduction

This project has advanced from its initial proposal, with the goal of developing a comprehensive traffic sign and light recognition (TSR) system for older vehicles lacking integrated ADAS. We have embarked on creating an innovated cascaded deep learning model capable of detecting and classifying both traffic signs and lights. Our approach diverges from conventional methods, which typically focus on one class, by utilizing a two-stage cascade system. The first stage detects the presence of traffic signs or lights, while the second classifies traffic signs into various categories and traffic lights into red, yellow, and green. YOLO, CNNs, and labelImg are a few examples of the tools that have been crucially effect this progress.

In advancing our project, we encountered some challenges in both detection and classification stages of TSR system. In contrast to typical single-class concentration, our new deep learning model incorporates a two-stage cascade method. The initial stage faced hurdles in accurately detecting the presence of traffic signs or lights. Following that, classifying these into various traffic sign categories and traffic lights as red, yellow, or green presented further complexities.

Despite these challenges, we have achieved an acceptable confidence score which demonstrates the model's near-accurate ability to detect and classify traffic signs and lights. This achievement marks a significant step toward our aim of implementing this system in real-world scenarios, utilizing real-time cameras and user interfaces, thereby extending the safety features of modern TSR systems to older vehicles.

2 Literature Review

In a general perspective, studies about TS-TL detection and recognition focuses on basic and complex approaches. Where basic approaches mostly depends on fundamental machine learning methods like random decision trees, k-NN and support vector machines. They have low complexity levels and easy to applicate and less susceptible for overfitting which possibly makes them useful for stationary enviromental applications. In our case for our general desire to develop a model to make it applicable for older cars that are not having ADAS system, we chose a complex deep learning model and similar studies to refer.

In the study we refer in our proposal we made more observations during our own implementations which we would like to share. In the referred study (Jayasinghe et al. (2022)) researchers implemented a similar model like we inspired from them. Main distinctions are that they tried to perform already configured CNN models on detection and classification stages like Residual Networks and SSD-MobileNet etc., by determining superclasses for detection stage and underclasses for classification. They also shared a important notice on their own testing which was increase in probability of error due to the unbalanced number of data per class relation. They observered that classes having less number of data then the mean data per classes on general dataset, are more prone to produce lesser precisions at the output, in order to solve it they apply augmentation on these classes to balance portions.

Another real-time traffic sign recognition system for autonomous vehicles is described in the article (Kardkovács et al. (2011)), which addresses issues such as fluctuating ambient lighting, occlusions, sign rotations, and varied regional sign designs. Color segmentation, region of interest identification, and traffic sign detection are all features of this system, which employs feature extraction and data mining classifiers. Its architecture includes the filtration of regions based on color and shape, alongside traffic sign recognition employing Haar-like features and neural network-based voting. Tested under various conditions, the system demonstrates promising results for real-time applications.

In relation to the project we are carrying out, the article's focus on traffic sign and light recognition is parallel to our work. However, our project adopts a two-stage cascading deep learning model that integrates YOLO and CNNs and intends to extend this feature to older automobiles. While both programs address the complexity of real-time recognition in a variety of situations, our project stands out with its cascading model and emphasis on fitting for older vehicles.

3 Methodology

3.1 Architecture and internal structure of the model

During the first meetings on the project we mostly argued about the scheme that will be used to detect and classify the real time images that will be obtained from a camera-type input device. There were several options in consid-

eration while we were arguing but the main one that effected our work was (Jayasinghe et al. (2022)) where the researchers used a cascaded approach to detection and classification with two different types of CNN's.

In the first stage they defined super classes that will detect objects under a scene image from driver's general perspective and bound them under boxes which will be the output of the first stage and input to the classifier. Similar to this instead of developing or researching a CNN model we used the advantage of YOLO model which is also a specialized CNN model for object detection.

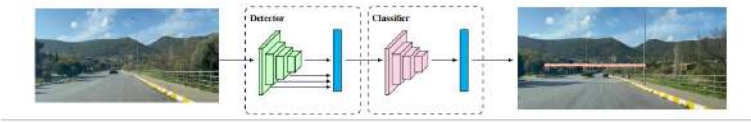


Figure 1: Architecture of learning model (Jayasinghe et al. (2022))

Reason we worked with YOLO was it's simplicity to train, observe and manipulate it's source code in order to achieve our desired results, some of them will be explained in later parts. In order to train detector with YOLO we need 4 fundamental parameters,

o Configuration file
o Image reshape size
o Device type
o Weight file

Where the most important one is the configuration file which includes paths to the necessary files for supervised learning. As we said afore YOLO is an another CNN model with it's own internal structure with certain hyperparameters we are going to share under the subsection below.

Detector output designed to be sent to the next CNN model purposed for classification of detected traffic sign or light. We solved the light classification with a simple approach under the detection source code of YOLO with the help of attributes of OpenCV library where if the range of previously given lower and upper 1x3 array type threshold values intersects with the hsv transformation of detected box of traffic light YOLO returns the color of detected traffic light without the need of an external need of CNN.

With the solution of light classification problem we only left with a proper CNN model which will perform the sign classification task. During the research of a dataset with proper cropped images we encountered, one that will be explained in below subsection, and with various code implementations of this dataset. We selected two CNN models that works one with PyTorch and other Tensorflow and used tensorflow for our midphase report.

Tensorflow model is a supervised model where it categorizes the assigned number of classes from 0 to 42 as the output layer of the neural network. As an input it reads the content of the folder (images) and performs resize and normalization operation transforming image a 48x48 matrix with values between 0 and 1.

After this transformation we are ready to input our CNN. CNN's working style can be summarized with three main operations, which are convolving, pooling, local contrast normalization (task special) and dropout. These classes includes many different parameters for specific uses but mainly they simplify and prevent the information within the transferred data and develop a mathematical upgrade technique while performing it. In general we use convolution and pooling operations for simplifying the data and local contrast normalization and dropout to prevent the consistency of information.

Before the training researchers defined a class to compute an initial learning rate with a small warmup training, given Cosine Decay class performs the necessary operations in order to determine the initial learning rate. With the initial learning rate value rest of the training operation updated with the categorical cross entropy loss function and stochastic gradient descent optimizer, result graphs will be shared in experimental results part.

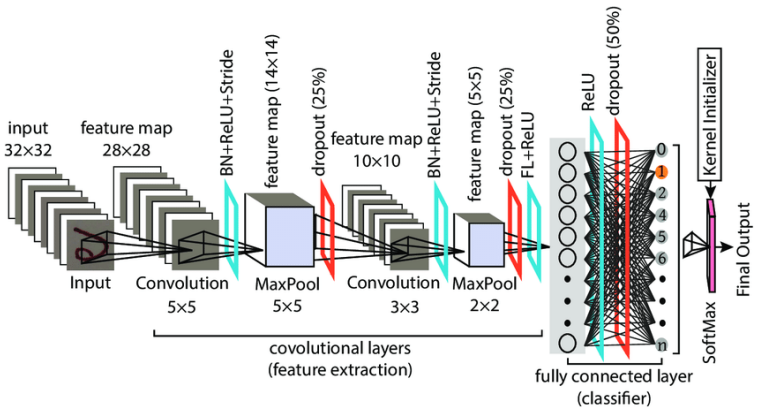


Figure 2: Example of an overall CNN model (Rahaman et al. (2019))

3.2 Dataset characteristics

During our researches we encountered GTSRB (mykola (2018)) in Kaggle platform, which is "German Traffic Sign Recognition Benchmark" Dataset. Dataset has 43 sign classes with more than 40.000 cropped images in total. But the main problem that wasted most of our time was dataset's compatibility with Turkiye.

In order to move quickly in steps of project and demonstrate a plot project we decided to keep the test area of final product limited by our campus area. So we took scene pictures from the different locations of our campus and added their YOLO out (savecrop) to our CNN dataset classified as classes 44-55, in the mean time we also removed unnecessary classes from the dataset to decrease possibility of error that might be produced by the distribution of weights.

Returning to the initial stage (YOLO), we gathered various scene images from driver's perspective (ituracingdriveless (ituracingdriveless)) and also included the afore mentioned pictures from our campus in order to annotate them with the tool called LabelImg. LabelImg is a basic tools that helps to user to draw boxes in specified format and returns them to the given locations as .txt files including ROI (region of interest) points which are, normalized center points of x and y and horizontal and vertical lengths of the box. .

3.3 Train, validation and test

For training we installed yolov5, reached it's files from windows terminal and installed necessary packages to operate. Thanks to one of our teammates, we decided to make advantage of his computer's GPU, so we followed a guideline to help to perform training quick in time. Main tools to perform the training with a GPU device was PyTorch and NVIDIA CUDA. Due to our misunderstanding we installed incompatible versions of torch and cuda at first and tried to train it with CPU first but after some corrections we performed YOLO training under 1 hour with the GPU unit.

During the training of YOLO some of the important hyperparameters that were manipulated are, batch size, number of epochs, image size, confidence threshold and non-maximum suppression threshold.

Due to some noisy outputs as the result of several test videos we updated YOLO dataset with the third 'none' class to prevent these noisy objects from passing the confidence threshold.

As we referred for light classification, we also solved the sign classification stage of the model inside the YOLO files by simply importing necessary tensorflow modules and uploading our trained CNN model file. In result we printed the confidence scores of both YOLO and CNN sign classifier on the test videos, detailed results will be shared in experimental results part.

3.4 User perspective and hardware implementation

For this stage of the project we are not yet implemented any embedded system to test on field in real time with camera tools etc.. For midphase presentations we developed a web UI to demonstrate what we propose in the end of the project.

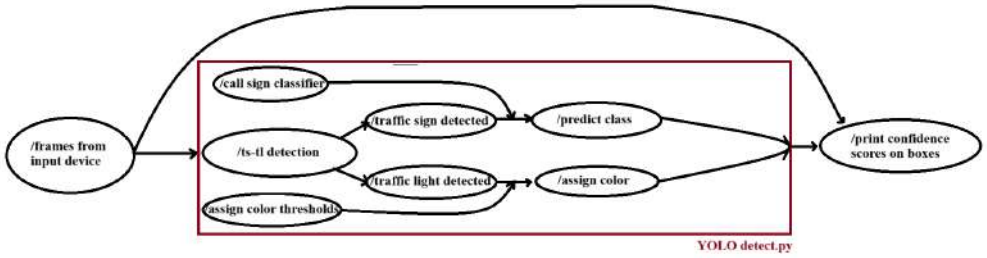


Figure 3: rqt graph modelling of overall flow

4 Experimental Results

In this section, we will discuss the training outcomes of our models through informational graphs which reveal the models' various properties that we use for understanding what will be the models' expected behavior with the given datasets and models' learning nuances that we can use for optimizing the models' behavior. Furthermore, we will also discuss the experiments we have done with the combination of these two models. Before continuing on the results, there are some terms we need to explain to what they refer to in this paper's context.

- ***Precision:*** This refers to the ratio of the model's predicted true positives to all positives (true positives + false positives).
- ***Recall:*** This term refers to the ratio of the model's predicted true positives to all actual positives (true positives + false negatives).
- ***Confidence:*** This refers to the probability of a detected object belonging to a specific class in real. This means the model is more confident that the object it detected belongs to the class it predicted when its confidence level is higher.
- ***Intersection over Union (IoU):*** This is used for evaluating the model's performance by comparing the predicted bounding box to the provided (ground truth) bounding box.

4.1 Detection

First, we trained YOLO for 300 epochs using the yolov5s model weights as a base. Here are the results:

- **F1-Confidence Curve (Figure 4)**
  The F1 score is a measure of the model's accuracy that combines the model's precision and recall by taking their harmonic mean. The main benefit of this graph is to find the optimal confidence threshold value which we can use for the predictions. When we look at the bold blue curve which considers the traffic light and traffic sign classes together, it increases up to a certain level and decreases rapidly afterward. This decrease tells us after a certain confidence score, which is 0.637 in this graph, the model starts to miss the actual positive predictions which lower the recall and overall F1 score. Thus, if we set the threshold confidence value in detection to accept a confidence score of 0.637 or higher, the model will have an optimized precision and recall which is shown as 89% in this graph.
- **Precision-Confidence, Recall-Confidence and Precision-Recall Curves (Figure 5)**
  The graph of figure 12c shows the precision of the model at various confidence thresholds for the classes. A higher precision value at a specific confidence level tells us that the model is very accurate when it finds something. In the legend of the graph the label "all classes 1.00 at 0.974" means when the model detects an object with a confidence value of 97.4% or higher, the model is perfectly accurate of what it detected.

  The graph of figure 12d shows the recall of the model against various confidence thresholds. A high recall means that the model can find most of the actual positives, but it may also be making more false positive errors. The recall converges to zero near 100% confidence score which means the model can find all actual positives and make no false positive errors, which is consistent with the figure 12c graph. The label "all classes 0.94 at 0.000" in the legend of the graph means the recall is at its highest at the starting point which makes sense when considering the previous conclusion we made so far.

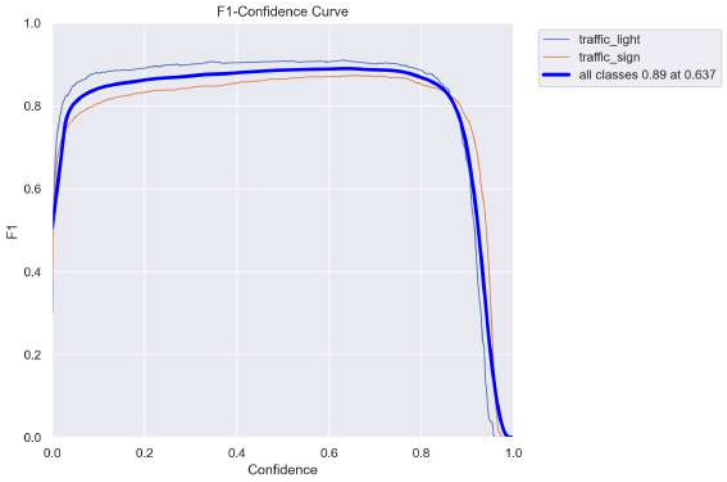
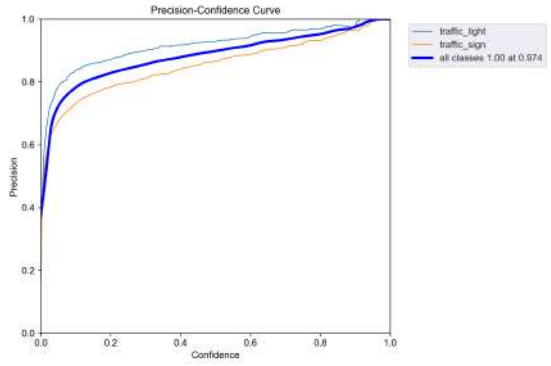
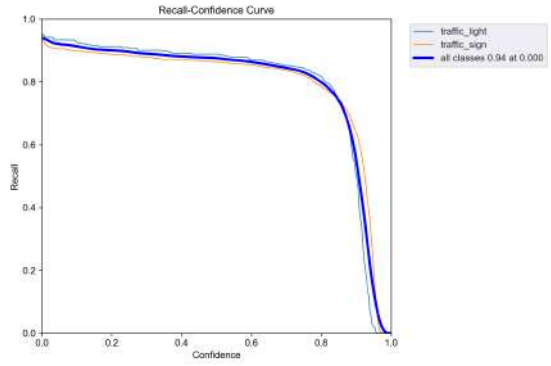


Figure 4: F1-Confidence Curve

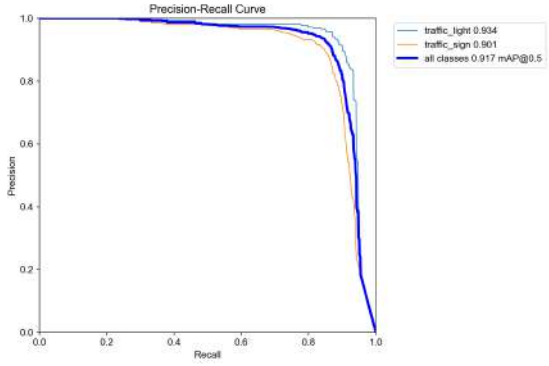
The graph of the figure 5c shows the trade-off between the model's precision and recall in the predictions. The label "all classes 0.917 mAP@0.5" in the legend means the maximum average precision (mAP), which is 0.917, is achieved at the IoU threshold of 0.5. We will use this hyperparameter for detection in our coalition experiments.



(a) Precision-Confidence Curve



(b) Recall-Confidence Curve



(c) Precision-Recall Curve

Figure 5: Precision-Confidence Recall-Confidence and Precision-Recall Curves

- **Loss and Metrics Comparison by Epochs (Figure 6)**
  These graphs show different aspects of the model's performance throughout the training of the model. Box loss

represents the error in the bounding box predictions, object loss represents the loss related to the objectness score, indicates the error in class predictions, and metrics show the variation of the related parameters during the training and validation. A decrease over epochs in box loss means that the model is getting better at predicting the bounding boxes, a decrease over epochs indicates that the model is becoming more confident and accurate in predicting the presence of objects within the bounding boxes, and a decrease over epochs in class loss meaning the model is learning to classify the objects it detects which results in improving accuracy.

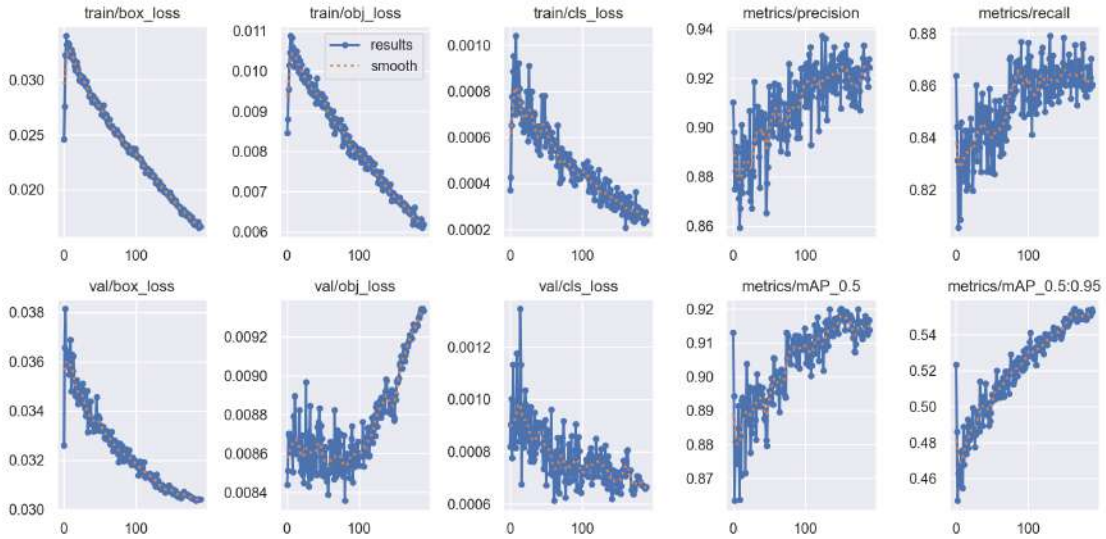


Figure 6: Loss and Metrics Comparison by Epochs

4.2 Classification

Second, we trained a pre-constructed and configured CNN model for traffic sign classification. The training ran for 30 epochs and resulted in x.xxx accuracy with x.xxx loss. In figure 7, we see the change of training and validation accuracy over 30 epochs on the left, and on the right, we see the change of training and validation loss over 30 epochs. On the training and validation accuracy graph, we see both training and validation accuracy rise significantly at the beginning and then flatten. This is an expected output and shows that the model is converging to a solution. As the epochs past, the training accuracy increases and the gap between the validation accuracy is closing. At the end of the training validation accuracy is decreasing ever so slightly, which indicates that model starts to overfit merely. Thus, with the dataset we have, 30 epoch is a good end point for the training.
On the training and validation loss graph, we see the same conclusions that we made with the accuracy graph, including a slight increase in the validation loss towards the and, which is a sign of overfitting. It makes sense that these graphs are consistent since the loss is a measure of the error and accuracy is a measure of the correct predictions.

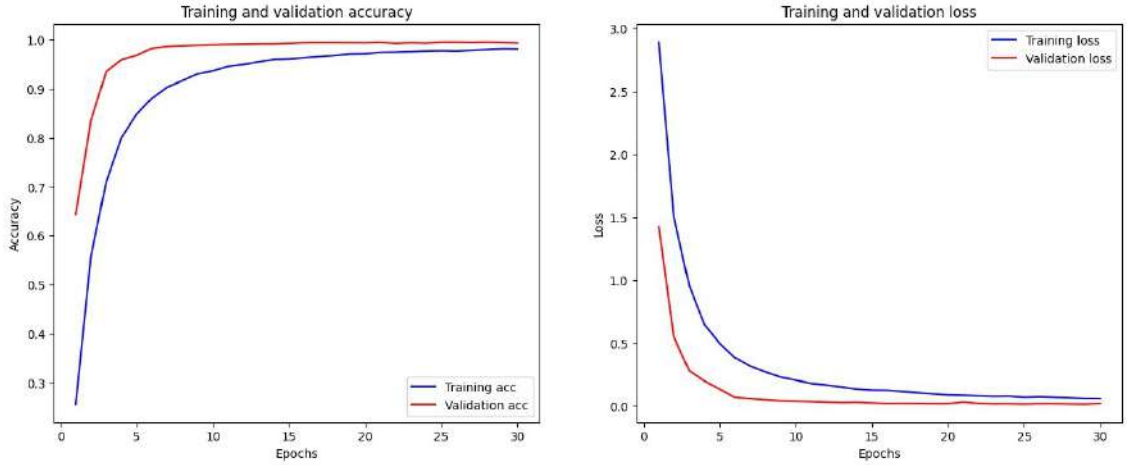


Figure 7: Training and Validation Accuracy and Loss Plots

4.3 Coalition

Third, we combined our trained YOLO and CNN models and ran some experiments. Those experiments covers how the combined model behavies under different scenes such as, day and night; rainy, clouded and clear weather; inner city and highways. Our trained YOLO model is behaving well with the threshold value we found in the F1-Confidence Curve (Figure 4). However, our trained CNN model was not behaving as expected and needed some tuning.
First, to eliminate miss predictions of the CNN model we optimized our CNN prediction threshold to only show predictions with 85% confidence or higher. With this, the CNN model wont show the prediction that is not fully sure about its guess. Thus, we prohibited miss predicitions when non-optimal cases occurs such as, damaged signs in inner city (Figures 8 & 9).

Next, we have seen that some common traffic signs are not present in our training dataset, thus even if the YOLO model detects that there is a traffic sing in the scene, our CNN model couldn't make a accurate prediction. Therefore, we compleated the missing traffic sign by adding new images to our training dataset. At the current state of the CNN model, there is no missing class for the traffic signs that exist in our campus (Figure 10).

In addition, we observed that our trained YOLO model also detects traffic signs that are facing backwards. To eliminate this issue, we came up with a plan to implement a psuedo class named "none", which we can use to label those inverted traffic signs in the scenes as "none" and train the model accordingly (Figure 11).

And lastly we included some experiment outputs from different types of scenes (Figure 12).

(a) Before adjusting the threshold

(b) After adjusting the threshold

Figure 8: Before and After Adjusting the Threshold Value in a Inner City Scene

(a) Before adjusting the threshold

(b) After adjusting the threshold

Figure 9: Before and After Adjusting the Threshold Value in a Rainy Wheather

(a) Before completing the dataset

(b) After completing the dataset

(c) Before completing the dataset

(d) After completing the dataset

Figure 10: Before and After Completeing the Dataset - Scenes From Campus

Figure 11: A Scene that Shows YOLO detecting a Traffic Sign Faced Backwards

(a) Inner City

(b) Inner City

(c) Night

(d) Freeway

Figure 12: Various Experiment Outcomes

5 Conclusions and Future Works

In order to summarize what we have achieved so far; we have completed the processes of labelling images from different datasets and divided them into 2 super-classes which are ”traffic sign” and ”traffic light” which we gathered together. In the initial phases of the project, obtaining the appropriate dataset was the first challenge that we faced. Following the conclusion of this procedure, we used the computer vision platform ”Roboflow” to train our dataset in the YOLO v5 (You Only Look Once) format.

As we discussed earlier, we have used a cascaded deep learning model that starts with YOLO and followed by Convolutional Neural Network (CNN) in which the cascaded system detects the traffic light or traffic sign in the first stage and then classifies the YOLO's output as different categories for traffic signs and red, green, and yellow for the traffic lights. The cascaded system's components also feed one another in the same script so we have gained from the computational power.

In our CNN model, we used TensorFlow-Keras for building and training neural networks. We built our dataset for the CNN model to classify traffic signs and marks on the German-Traffic-Sign-Recognition-Benchmark, and we also added traffic signs used in Turkey, which are not included in the German dataset, as separate classes here. We have investigated the cropped images from different datasets in Turkey and integrated them in our model.

We also made use of the advantages of CUDA, processing power of GPUs (Graphics Processing Units) for computationally intensive tasks as we are training our model. The computational platform we use to train our model is NVIDIA RTX-4060.

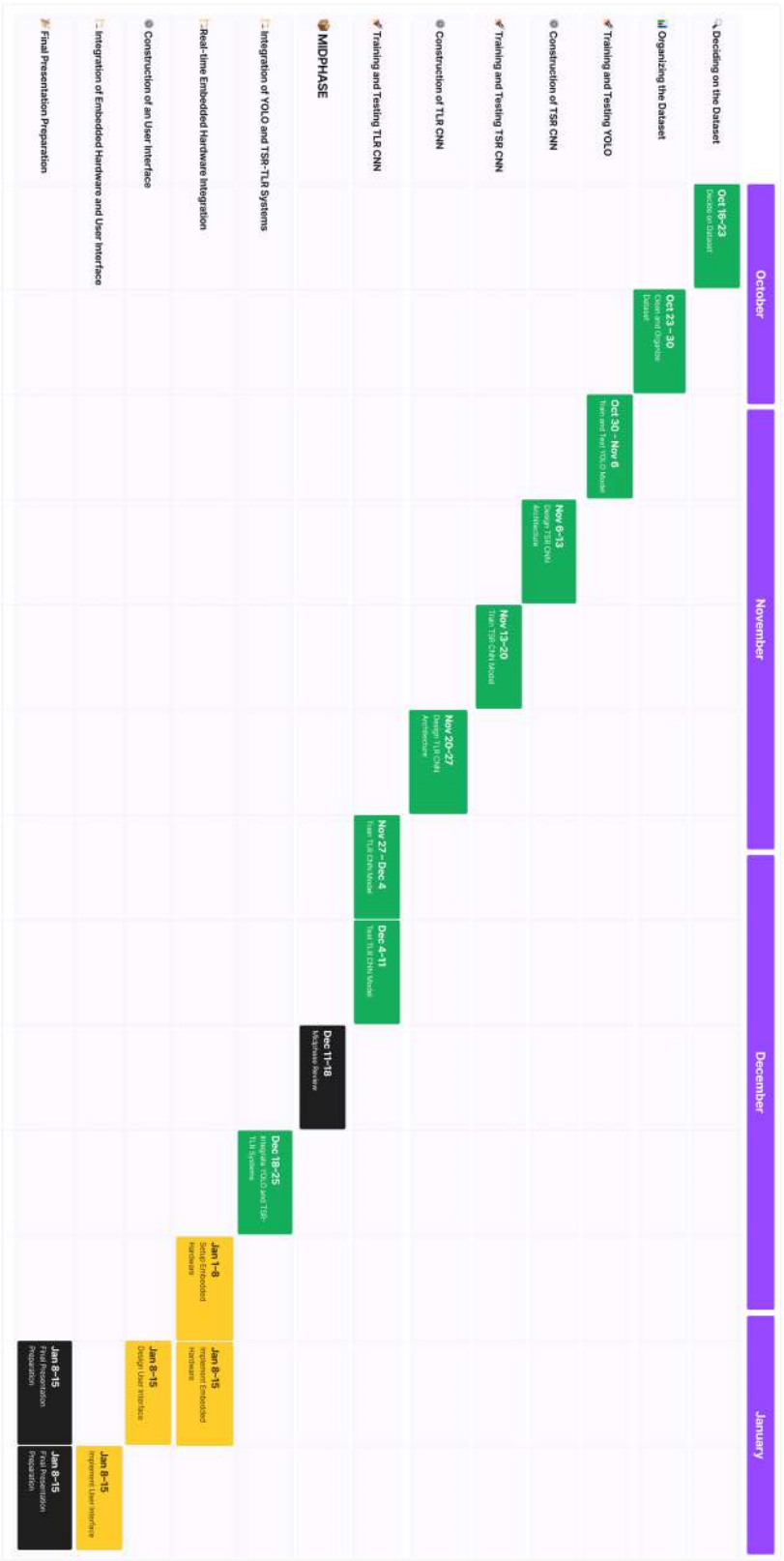
For the final process for so far, we have determined the experimental results and compared them with the number of training epochs which was 30 in our case. We also analyzed the loss function compared with the number of epochs in order to stabilize the model. The outcome was pretty satisfactory, our model is now able to recognize the traffic signs and traffic lights correctly.

Additionally, we have designed a Head-Up-Display (HUD) model which projects information to the windshield seemingly hovering just above the road ahead from the point of view of the driver. This HUD will also project the traffic signs and lights which was recognized in real-time and sends instructions or warnings to the driver based on these traffic signs and lights. We have designed a model for these features which you can clearly observe in the demonstration. Beyond this point, the dataset for CNN will be enhanced if there is a significant traffic sign which we did not add to the CNN as cropped images.

In the final stage of the project, we will embed our system in a Raspberry Pi to process the real-time images which data flows through the camera located at the level of the car's rear-view mirror. It will continuously monitor the road ahead, when it detects a traffic sign or traffic light, the data will flow through the CAN (Controller Area Network) of the automobile and a high-quality image will appear in the HUD for the corresponding sign or light. Thanks to this system, the driver will be able to look at the Head-Up-Display and receive feedback about the condition of the road without being distracted while driving.

We believe that this project will fill an important gap in the market. Since this project aims to eliminate the problem of traffic sign and traffic light classification in older vehicles that do not have autonomous driving, customers may be interested in such (TSR-TLR) system because they can have this safe driving assistance at lower prices.

6 Weekly Schedule/Project Plan



References

ituracingdriveless. Github - ituracingdriverless/TTVS: Türkiye Trafik işaretleri Veriseti - Turkish Traffic Sign Dataset. https://github.com/ituracingdriverless/TTVS.

Jayasinghe, O., S. Hemachandra, D. Anhettigama, S. Kariyawasam, T. Wickremasinghe, C. Ekanayake, R. Rodrigo, and P. Jayasekara (2022). Towards real-time traffic sign and traffic light detection on embedded systems.

Kardkovács, Z., Z. Paróczi, E. Varga, A. Siegler, and P. Lucz (2011, 08). Real-time traffic sign recognition system real-time traffic sign recognition system.

mykola (2018, nov 25). Gtsrb - German Traffic Sign Recognition Benchmark. https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign.

Rahaman, M., M. Mahin, M. Ali, and M. Hasanuzzaman (2019, 04). Bhcdr: Real-time bangla handwritten characters and digits recognition using adopted convolutional neural network.