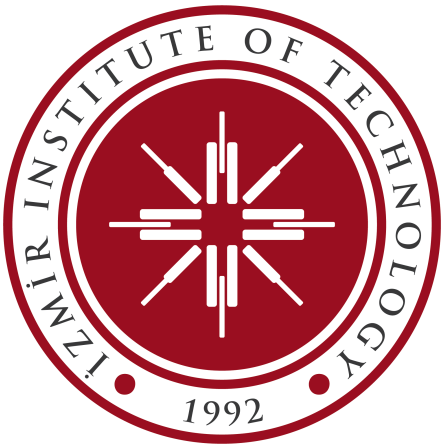
CENG483 Behavioural Robotics
Fall 2023

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

Final Report

February 25, 2024



Group Members

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

Abstract

This study provides a full summary of our project, which aimed to create a sophisticated traffic sign and light recognition system for older automobiles that lacked Advanced Driver Assistance Systems (ADAS). The research successfully overcame initial obstacles to develop an innovative cascaded deep learning model adapted to Turkey's specific traffic conditions. This model builds on the strengths of prior models, using YOLO, CNNs, and labelImg to recognize and classify traffic signs and lights. The technology performs well in real-world testing scenarios, indicating a substantial development in traffic sign recognition and providing a feasible approach for improving safety features in older automobiles.

1 Introduction

The advancement of vehicle technology has considerably enhanced road safety, primarily through the incorporation of Advanced Driver Assistance Systems (ADAS). However, a large number of older automobiles are still in service, missing these enhanced safety measures. This final project report describes our unique efforts to overcome this gap by creating a comprehensive traffic sign and light recognition (TSR) system designed exclusively for older vehicles. This effort aims to improve road safety while also democratizing access to modern driving assistance technologies.

Our project began with the ambitious goal of developing a cascaded deep learning model capable of reliably detecting and classifying a wide range of traffic signs and lights, a feature normally reserved for newer automobiles with built-in ADAS. Our system is built around a two-stage cascade paradigm, which differs from traditional single-class systems. In the first stage, complex algorithms are used to detect the existence of traffic signs and lights. The second stage is more complex, requiring the classification of detected traffic signs into several categories and traffic lights into red, yellow, and green. This dual-stage technique improves recognition precision and reliability, which is crucial for real-time traffic sign and light detection in a variety of driving scenarios.

Throughout the project, we encountered several hurdles ranging from data collecting and model training to real-world deployment. The first step involved considerable research and development to identify the best machine learning models and tools, such as YOLO, CNNs, and labelImg. We then modified these models to reflect the specific traffic conditions and sign variations prevalent in Turkey. The procedure was iterative, involving constant testing, tweaking, and optimization of the model's accuracy and efficiency. The paper describes these stages of development, the challenges we faced, and the inventive solutions we devised. This project exemplifies our team's technological expertise and represents a substantial development in traffic safety. It has the potential to transform how older vehicles interact with complex road settings.

In the following sections of this paper, we will go over the details of our technique, such as the architecture of the deep learning model, the datasets used, and the special issues connected with traffic sign and light recognition in Turkey. We also examine the many tools and technologies used to attain our objectives, emphasizing the novel parts of our approach as well as the modifications made to current technologies. The study describes our experimental results, which provide insights into the model's performance in numerous real-world circumstances, as well as the ramifications of these findings. Furthermore, we investigate our system's future possibilities, taking into account scalability, adaptability to diverse geographic regions, and connection with existing vehicle systems. Through this extensive research, we hope to provide a clear and complete description of our journey from concept to implementation, as well as helpful insights for future endeavors in this quickly evolving field.

2 Literature Review

Various studies that has been contributed to this research field having two approaches that differs according to the specified task they will be implemented. This two approach is mostly determined by the researchers environmental scale selection and computational limits to develop a learning model. For stationary environments like small towns, university campuses etc. one may choose basic approaches like fundamental machine learning methods of k-NN neighbors or SVM's. For our task we decided to keep operational scale a bit large, even though we selected our campus area as a plot, and tried to configure a deep learning model constructed with CNN's.

In addition to what we inferenced out of the referred study in our project proposal we made more observations during our own trials which we would like to share. In the referred study (Jayasinghe et al. (2022)) researchers developed a cascaded detection & recognition model like we inspired in our study. 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 detailed 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 observed that classes having less number of training data than the mean of data per classes over general dataset, are more prone to produce lesser precisions at the output, in order to solve it they apply additional augmentation on these classes to balance portions.

In another study that is focused on 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

In final version of the study, we configured a 2-stage cascaded deep learning model that is specified for two task; detection and recognition. Besides the scheme we choose there were other options like direct classification methods, but we found it a little bit needy in number of data terms to bring the model in an optimum level, with this approach we firstly filtered our image input by detecting box with YOLOv5 where it classified as either sign or light and continued with the classification of the upper class detected box with a CNN model. For further improvements of the project we are thinking of arranging another middle stage as middle class sign classifier where the signs assigned to their respective class such as; speed signs, warning sings, informative signs etc., for light classification we only have the idea of improving image processing with OpenCV for better classification.

Our intends in future mostly shaped by the study we referred afore. 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.

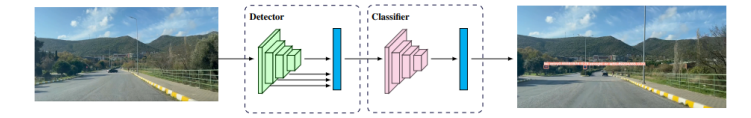


Figure 1: Architecture of learning model (Jayasinghe et al. (2022))

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 stage 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, such as scene images that are taken in driver's perspective and .txt files that includes the numerical data that is going to be processed along with the image makes is labelled. 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 specialized for classification of detected traffic sign. We solved the light classification with a simple approach under the detect.py source code of YOLO with the help of attributes of OpenCV library for image processing 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.

```
red_threshold_lower = np.array([0, 80, 80])
red_threshold_upper = np.array([10, 255, 255])
```

Code Snippet 1: Pre-determined thresholds for box's color contrasts for color red.

With the solution of light classification problem we only left with a proper CNN model which will perform the sign classification task. During the search of a dataset with 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 final 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 on image, as a result a 48x48 matrix with index values between 0 and 1 is created.

After this transformation we are ready to feed 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 by keeping a ratio between transformations and upgrading the weights that makes the connections between the layers 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 while preventing any overfitting situation.

3.1.1 Dropout

Dropout is a regularization technique that is currently using in many neural network implementations. It can be simply explained as on-off'ing neurons during training with a probability distribution determined to each. By this approach model weights forced to encounter unexpected situations and prevent any overfitting where some weight unboundedly gains increase in value. Dropout Equation:

$$\text{Dropout}(X_i, p) = \frac{1}{1-p} \times \begin{cases} X_i, & \text{with probability } 1-p \\ 0, & \text{with probability } p \end{cases}$$

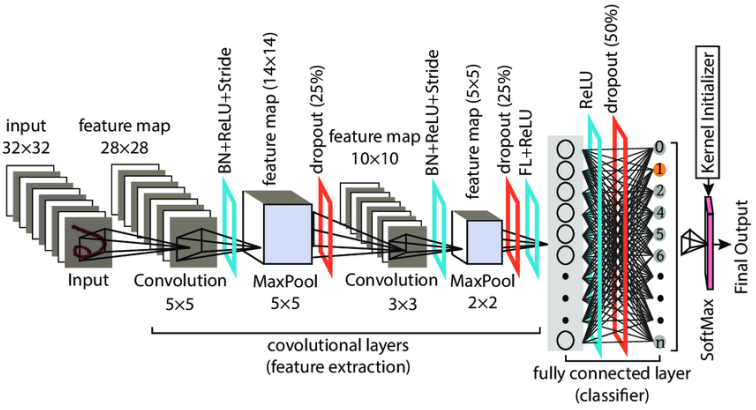


Figure 2: Example of an overall CNN model (Rahaman et al. (2019))

3.1.2 Local Response Normalization

CNN models are mostly repetitive models for some percentage of the structure where convolution pooling and any other regularization techniques like dropout and LRN performed in each cycle. In addition to the dropout a complex approach like local contrast normalization is performed after each pooling layer as an update. LRN basically is an image processing technique performed in order to enhance the contrast of specific regions in an image and obtain

normalized responses of neurons in a local neighborhood with a kernel in a similar size used in pooling and emphasize the characteristic of that region. For our case it is performed on numerical representations of the image that is fed after pooling in order to adapt the explanation in numerical representations.

Local Response Normalization (LRN) Equation:

$$B_{ij} = \frac{A_{ij}}{\left(\kappa + \alpha \sum_{p,q} (A_{pq})^2\right)^{\beta}}$$

Explanation of Terms:

A_{ij} is the value at position (i, j) in the original matrix.

κ (small positive constant) is added to avoid division by zero and stabilize the normalization.

α (scaling parameter) controls the strength of normalization.

β (exponent) controls the shape of the normalization.

The summation is over the local neighborhood of the element (i, j).

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.

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 as we mentioned afore. In addition 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. .

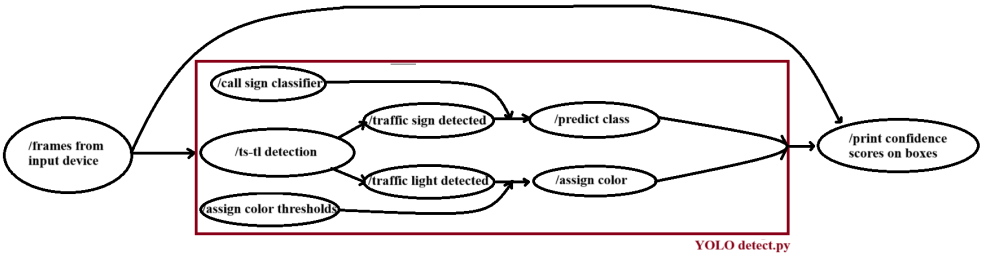


Figure 3: rqt graph modelling of overall flow

3.4 User perspective and hardware implementation

For real time implementation of cascaded model we needed essential materials like CPU/GPU's, RAM, Camera and a HUD display to keep user informed at any time. We had options like self constructed embedded system, which requires additional effort and time in order to fit the system and each component accordingly. Raspberry Pi platform since it's easy to upload and run when the requirements are set. In the end we had Raspberry Pi and conducted various test on it but had problems which will be explained in later parts. In addition to Raspberry Pi implementation we also conducted experiments with YOLOv5 detect.py run with computer camera.

3.4.1 Distance Measurement

We solved the measuring distance problem of the detected sign that was discussed in midphase representation with a simple approach. Parameters that are created during labelling for YOLOv5 includes the information of selected box class, x center, y center, width and height. We developed an idea that since in continuation of time changing parameters of a detected box are all except x center for most of the cases. So thinking that near boxes having greater and further boxes smaller areas, making an inverse relation between distance and center y point of the box has a straight relation with the distance resulting in division between y center and area of the box. Normalized numerical values that are assigned to these parameters needs no resolution scaling since having two parameters in division neglects the effect we also needed a root factor to suppress large scale change in calculation that we observe and a constant to scale and adjust the value.

The equation for the approximate real distance (d) of the sign is given by:

$$d = c \cdot \sqrt{\frac{y}{w \cdot h}}$$

Where:

d : Approximate real distance of the sign
c : Adaptive constant for adjusting
y : y-center of the box
w : Width of the box
h : Height of the box

3.4.2 Don't Care Signs

During driving and processing the image that comes from the ride, along with the signs that we care, we also observe unnecessary signs that doesn't our concerns, for example signs for the opposite side of the road. For this issue we developed an approach, to describe it in short, we processed every five .csv file outputs of detected images by first determining the most common detected boxes at the same time and dividing the scene in half after that. As a result of this we grouped the signs as left and right and measured pixel distances in their respective regions in order to measure their relevance between the detected signs. If the determined threshold is exceeded we concluded that signs are irrelevant from each other and doesn't concerned.

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.

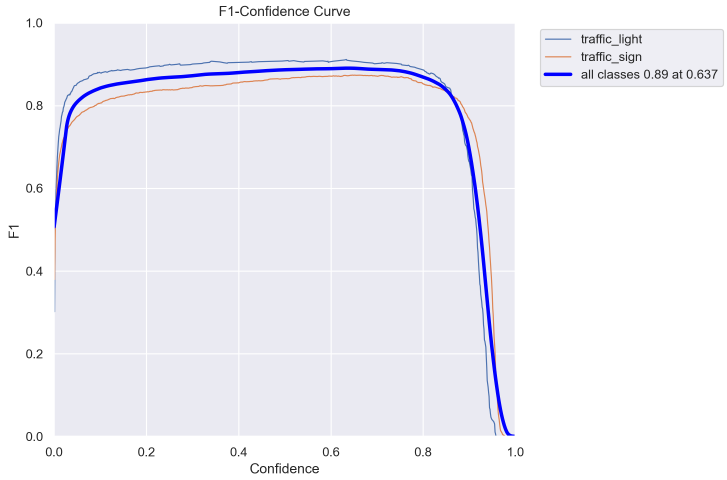


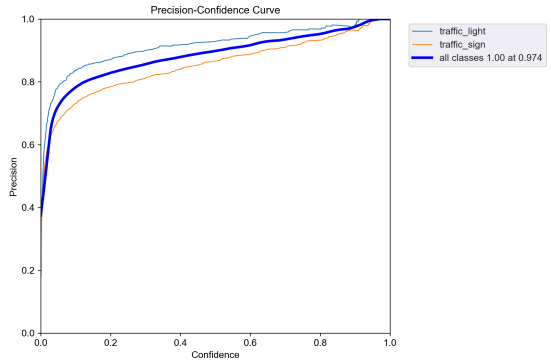
Figure 4: F1-Confidence Curve

- **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.

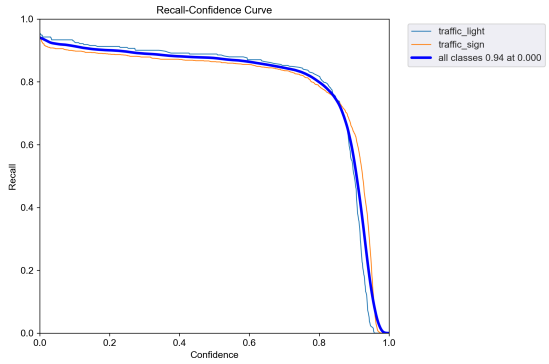
  The graph of figure 14b shows the recall of the model against various confidence thresholds. A high recall

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.

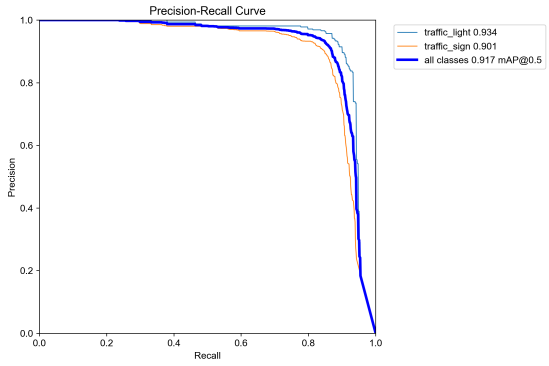
The graph of the figure 5c shows the trade-off between the model’s precision and recall in the predictions. The label ”all classes 0.917 mAP@0.5” in the legend means the maximum average precision (mAP), which is 0.917, is achieved at the IoU threshold of 0.5. We will use this hyperparameter for detection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

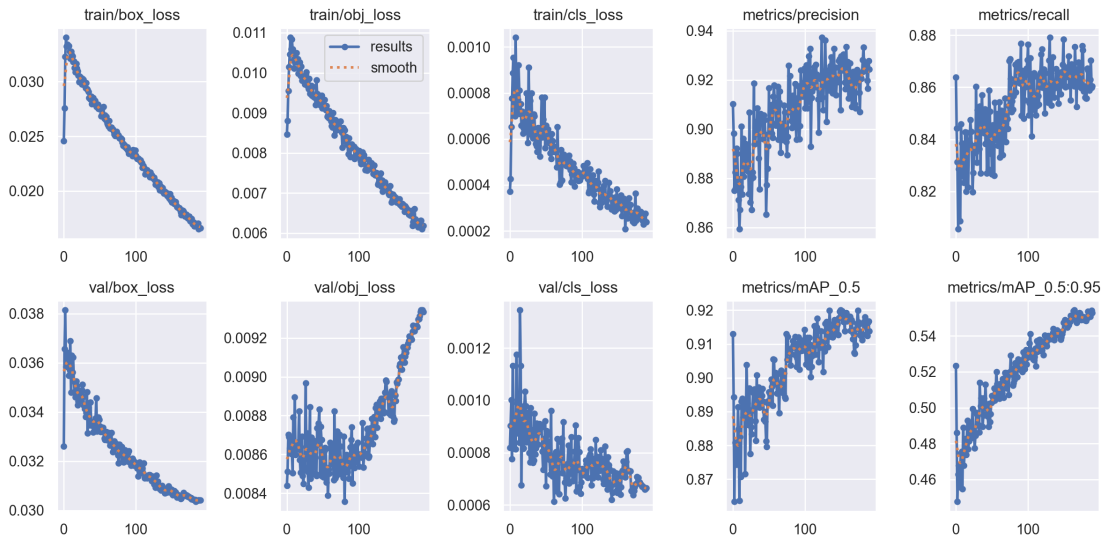


Figure 6: Loss and Metrics Comparison by Epochs

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.

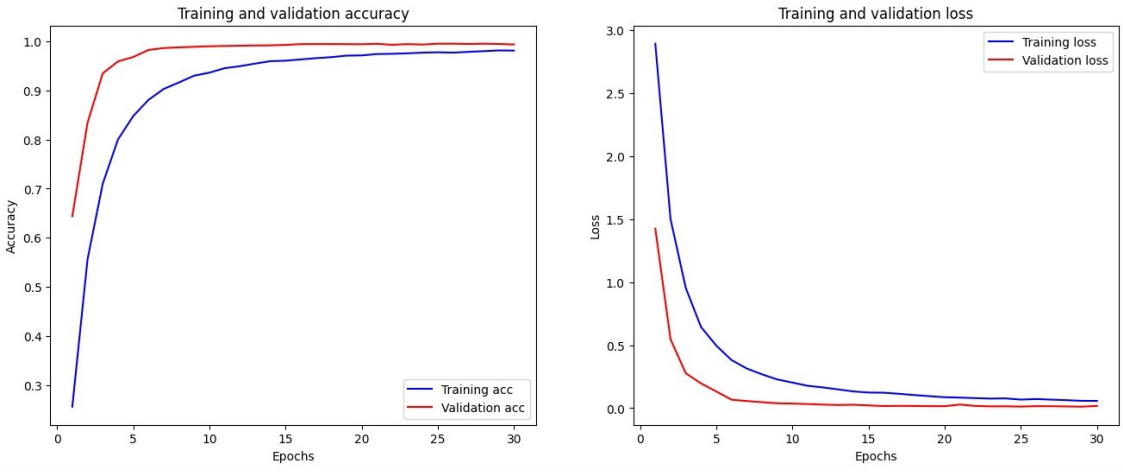


Figure 7: Training and Validation Accuracy and Loss Plots

4.3 Coalition

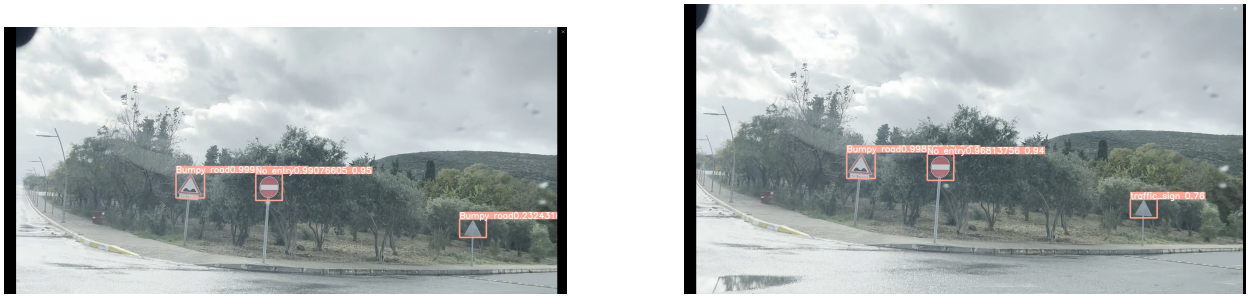
Third, we combined our trained YOLO and CNN models and ran some experiments. Those experiments covers how the combined model behavies under different scenes such as, day and night; rainy, clouded and clear weather; inner city and highways. Our trained YOLO model is behaving well with the threshold value 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 14 & 9).

(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

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 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

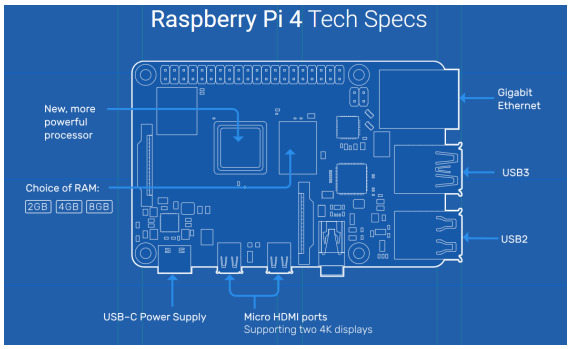
(d) Freeway

Figure 12: Various Experiment Outcomes

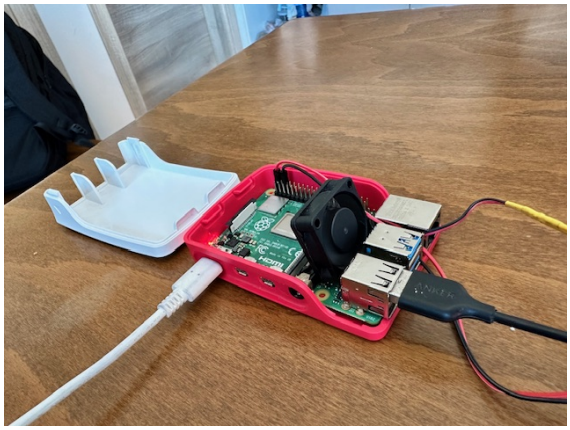
4.4 Implementation of Model in Real-Time

Following progress until the mid-phase, we aimed to embed traffic sign and traffic light recognition model in a suitable device and run it in real-time. We were considering to use a single board computer which has more powerful processor in it like NVIDIA Jetson Nano. Unfortunately, we were unable to get one so we decided to use Raspberry Pi 4 Model B with Coral Edge TPU.

We decided to use fourth generation of Raspberry Pi single-board computer Model B with the Coral USB Accelerator (Edge TPU) to enhance our machine learning interference tasks. It uses Broadcom BCM2711 quad-core Cortex-A72 (ARMv8) 64-bit processor and has 4 GB of LPDDR4-3200 SDRAM. The idea was to process the real-time data which is transmitted through the fish-eye camera SJCAM M20.



(a) Raspberry Pi 4 Tech Specs

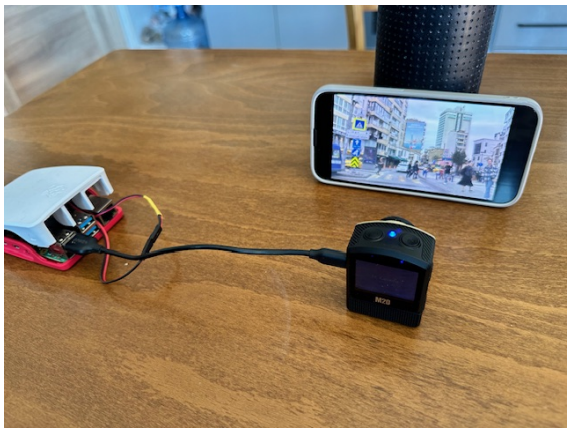


(b) Raspberry Pi 4 Model B Test Setup

Figure 13: Raspberry Pi 4 Model B with Coral Edge TPU

We have set up the project environment in order to collect data in real-time by utilizing the fisheye camera SJCAM M20 and using the processing power of Raspberry Pi 4 Model B. Unfortunately, things did not go as planned. According to our experimental observations, the processor of the Raspberry Pi was not powerful enough to process the given data. The frame per seconds (FPS) rate of Raspberry Pi was approximately between 0.5 and 1.5 which means that the data loss was huge. In addition to all these, we could not transmit the output of CNN to the Raspberry Pi 4. We transformed our models (YOLO and CNN outputs) into the TenserFlow Lite to run them inside the Raspberry Pi 4. We observed that YOLO gives the expected output, on the other hand, the CNN output model was not working inside Raspberry Pi 4. From both perspectives, utilizing the Raspberry Pi 4 Model B was useless in our project.

(a) SJCAM M20 Fisheye Camera for Real-Time Data Collection



(b) Raspberry Pi 4 Model B and Fisheye Camera Test Setup

Figure 14: Integration of Components

When we thought about a different way of processing the real-time data, we decided to use the computer which has NVIDIA RTX-4060 in it. According to the observations, the performance was more than enough to process the real-time data transmitted through the camera. Thus, the work environment of our model consists of three main components which are the fisheye camera that observes the ahead road continuously, the GPU in the computer that processes the real-time data in order to detect the traffic signs and traffic lights and thirdly, our model that runs inside the GPU. By combining these components, we could detect and classify the traffic signs and traffic lights in real-time.

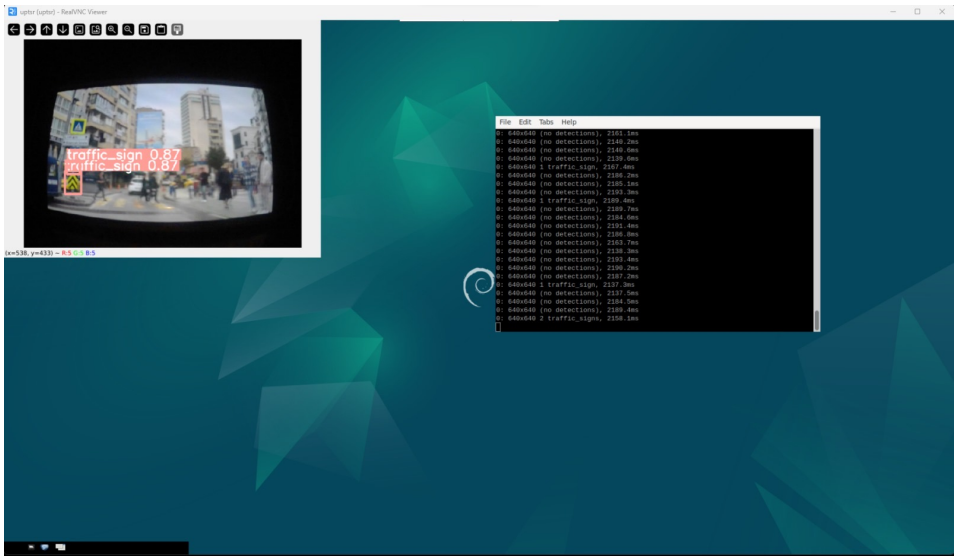


Figure 15: Output of YOLO Which is Embedded Into the Raspberry Pi 4B

Bill Of Materials

Product	Price
Raspberry Pi 4 GB	68.04 $
Camera	8.52 $
LCD Screen	4.01 $
Coral USB Accelerator	98.54 $
HUD Reflective Film	3.78 $
Other	18 $
TOTAL	**200 $**

Figure 16: Bill of Materials

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.

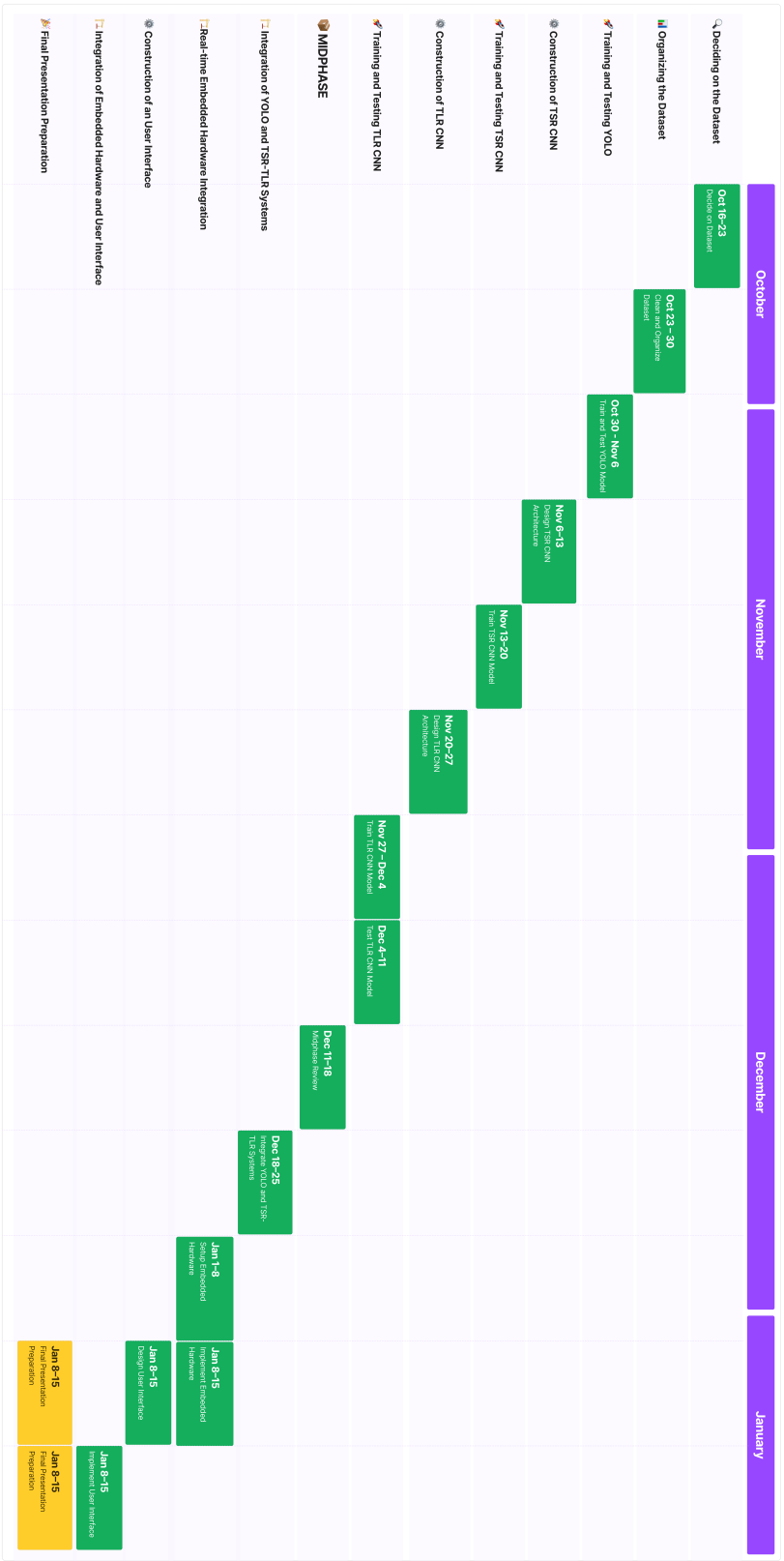
Additionally, we have designed a Head-Up-Display (HUD) model which projects information to the windshield seemingly hovering just above the road ahead from the point of view of the driver. This HUD will also project the traffic signs and lights which was recognized in real-time and sends instructions 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 our project, we added some features to the model and tried to embed our system in a Raspberry Pi 4 Model B to process the real-time data by utilizing the fisheye camera which is located at the level of car's rear-view mirror. We integrated our model to estimate the distance between the traffic sign which is detected and the camera in real-time so that it will continuously calculate the distance when the car approaches to the traffic sign. Additionally, by utilizing the coordinate information of the traffic signs, our model detects the traffic signs in

the lane where the car drives. By adding this feature, we prevent our model to detect the traffic signs which regards other drivers who plans not the follow the ahead road. For instance, in the city, if a vehicle is going to make a turn right, it drives in the right lane. The system in this vehicle does not need to detect the traffic sign in the far left lane. In order to optimize the number of significant traffic signs to be displayed to the driver, we have added such a feature.

After revising our model, we embedded it in Raspberry Pi 4 Model B. We suspected there would be FPS issues in real-time when Raspberry Pi runs our model, and that's exactly what we have expected. There is approximately 1.5 seconds of delay in real-time and that's not acceptable for such a system that has to detect and classify the traffic signs and traffic lights simultaneously. Thus, we decided to run our model in the computer which has NVIDIA RTX-4060 in it. We stabilized the camera at the level of rear-view mirror and connected it to the computer where the model runs. In this way, we could observe that our model can detect the traffic signs and traffic lights in real-time. In addition, we designed a User Interface (UI) which displays the icons of the corresponding traffic signs when the model detects it in real-time.

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.