COMP 4102A
Project Report
Apr. 10th, 2024

# Road Sign Recognition

Group 3

Bardia Parmoun
Liam Gaudet
Siddharth Natamai

## Abstract

The goal of this project is to detect and identify common traffic signs. The project takes in an image as input that can include multiple traffic signs and it aims to detect all instances of traffic signs in the image. The tool locates and identifies each traffic sign.

## Background

There are various computer vision models that already exist that can help us detect command road signs. An example of these models is the model(s) used by comma.ai which has been proven on the road with several driver assistance features: openpilot — open source advanced driver assistance system. comma.ai's open pilot has been used for over 100 million miles with over 10,000 active users and 450 contributors. It offers automated lane centering, adaptive cruise control, lane change assist, driver monitoring (which reduces the need to hold the steering wheel), and can drive for hours without any intervention. They are able to achieve these features by harnessing a vehicle's existing hardware and leveraging all the data collected from their users to continuously improve their models. Their model includes a very reliable traffic sign detection model which has helped guide our project. Using their efforts as inspiration we started looking for a reliable dataset to work off of as we cannot make our own. Our team used the following dataset: German Traffic Sign Recognition Benchmark which has been used for several similar projects and was instrumental in our development process by offering a comprehensive collection of traffic signs.

## Architecture

To help solve this problem, the problem was broken down into multiple steps. Here is a quick summary of the system:

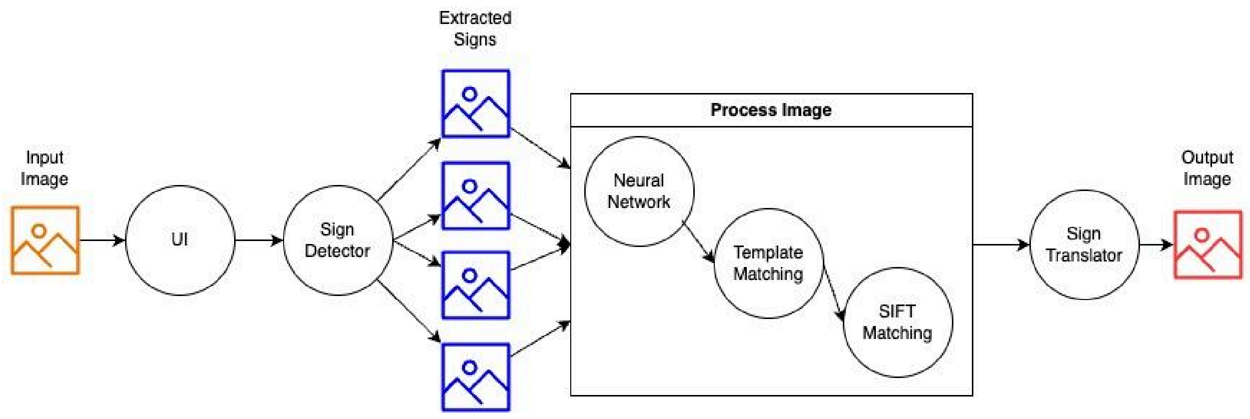


**Figure 1:** A quick overview of the system and its various components.

As shown in the diagram, the input image is first fed to the system through the UI. The image is then passed to the sign detector. This module extracts parts of the image that might have a traffic sign in them. Those images are then processed and a sign is assigned to each segment. Those values are then finally translated to the proper labels and added to the input image. Here is a detailed explanation of each unit:

**UI:** This module was developed using PySimpleGUI. This module allows the users to load their desired image and then calls the process image function to process it.

**Sign Detector:** This module is in charge of detecting areas within the image that might include a traffic sign. It achieves this by first grabbing the contours of the image. In this module the input image is converted to grayscale and Canny Edge detection is performed on it. The contours are then compared against the contours of existing shapes (triangles, rectangles, hexagons, and circles). This allows the module to find areas where a sign might be located at.

**Process Image:** This module includes all the code that is in charge of processing the input image and identifying the sign. It has 3 steps:

- **Neural Network:** This module includes a CNN model that has been trained on the GTSRB database. This module takes in an RGB image as input and tries to match it to one of the 43 categories within the GTSRB database. It then returns a list of the top 3 guesses sorted by their confidence score.
- **Template Matching:** This module uses OpenCV's match template function to match the given image to the 43 different categories. This module has the option of taking a set of templates as input and only performing the comparison on those. If no templates are provided, this module compares the input against all the possible templates. It returns the template that matches the input the best.

- **SIFT Matching:** This module is very similar to the template matching one. It uses the same set of templates but instead, it uses the SIFT function of OpenCV to compare the given image to the template. Similar to the template matching module, it has the option of comparing the image with a set of templates that are returned from the neural network. If that is not the case, it compares the image with all the available templates. The function will then return the image with the highest number of SIFT matches.

**Sign Translator:** Throughout the processing phase, all the traffic lights and templates are labeled with values between 1 to 43. This is to keep them consistent with the neural network and the GTSRB dataset; however, since these values are not clear to the user, a CSV file was prepared to match each number to its proper name. This module takes in the number for the sign and returns the proper name for the sign.

**Output:** Once all the analysis is done, a new copy of the input image is created with the sign area marked with a rectangle. This area is also labeled with the output of the sign translator module. In addition, a JSON file is created with the position sign and its label.

## Methodology

To evaluate the system, three different evaluation processes were performed. Each of these modules tested a different part of the system to allow for an in-depth identification of the strengths and weaknesses of the system. Additionally, a test dataset had to be created for us to test on. We cannot use the GTSRB dataset for this because that dataset does not contain any opportunity for traffic sign detection, which is an important part of our system. Instead, we downloaded a subset of the Mapilary Traffic Sign dataset and filtered it to 50 images taken in Germany, since that's where our model is meant to work with. Each image was annotated with the "True" sign locations in a JSON file, as well as what sign was present at each dimension.

For starters, the detection module had to be tested. In order to do this, we had to compare the sign rectangle created by our system to the true sign locations in each image's JSON description. To do this, we made use of a metric called a Jaccard score. A Jaccard score, also known as Intersection over Union, is calculated as the number of intersected pixels from two shapes divided by the union of those two shapes. In other words, a Jaccard score of 1 represents a perfect overlap between two images, a Jaccard score of 0.5 would represent a 50% overlap between the two images, and a Jaccard score of 0 would represent two disjoint shapes. Using the Jaccard scores, we set a threshold for the Jaccard score, and any pair of "true" sign locations to the "detected" sign location that had a Jaccard score above that threshold would be classified as a True Positive. Any true sign that was not matched was treated as a False

Negative, and any detected sign that was not matched was treated as a False Positive. Using these three values, the Precision and Recall of the detection were calculated. Precision represents the proportion of identifications that were correct, and recall represents the proportion of positives that were correctly identified. This process was repeated with Jaccard thresholds of 0.95, 0.9, 0.8, and 0.6 to evaluate how common it is for the detection to get close to, but not entirely on, each sign.

Next, the accuracy of our model was evaluated. Since we had three separate recognition models, we had to identify a way to determine how accurate each model was in our system. This meant we had to test each model separately using the "true" sign locations for each image in the test dataset. For this, each "true" sign from each image was parsed and fed through each of the Neural Network, SIFT, Template Matching, Neural Network + SIFT, Neural Network + Template Matching, and all three combined. The accuracy for each trial was calculated as the number of successful sign identifications divided by the total number of signs present.

Lastly, the overall system had to be evaluated. While it's important that each system individually can work on its own, lane detection and recognition have to involve both of these parts together. Therefore, to test the system as a whole each test image from our dataset was fed through our system. Each test image would have the detection module detect where the signs are to calculate True Positives, False Negatives, and False Positives. These values were then used to calculate the detection recall and precision. Then, only the True Positive pairs would be considered for calculating accuracy. The reason why only true positives were considered for accuracy was because the recognition algorithm always outputs a sign for any detected sign, so if a "detected" sign was not actually a sign then it would output a false value, heavily skewing the accuracy towards 0.

After each of these evaluations took place, the results from each were displayed on the console. The results of each of these evaluations can be found in the Results section of the report.

## Results

With the help of the evaluation module, the team was able to collect various results. Here is a summary of the results:

**Table 1:** Analyzing the accuracy of the detection algorithm with different thresholds.

| Threshold | Precision | Recall |
|---|---|---|
| 0.95 | 2.128% | 1.43% |

| | | |
|---|---|---|
| 0.9 | 4.255% | 2.857% |
| 0.8 | 23.40% | 15.71% |
| 0.6 | 27.66% | 18.57% |

Here we notice that the best value for the threshold of the detection algorithm is around 0.9 since it gives us a high value for precision and a low value for recall.

**Table 2:** Analyzing the accuracy of different recognition algorithms.

| **Method** | **Accuracy** |
|---|---|
| Neural Network Only | 24.29% |
| SIFT Only | 0.0% |
| Template Matching Only | 0.0% |
| Neural Network + Template Matching | 10% |
| Neural Network + SIFT | 2.857% |
| Neural Network + Template Matching + SIFT | 11.43% |

Here we notice that template matching and SIFT do not perform as well when they are not used without the neural network. This is mainly due to the similarities that exist between the different templates which can confuse the template matching algorithm.

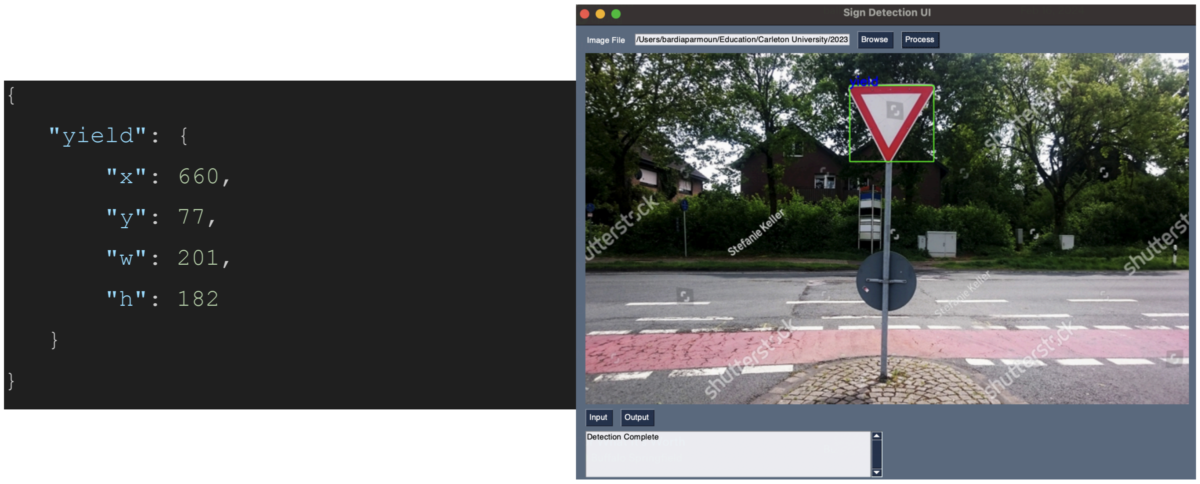
**Table 3:** Analyzing the accuracy of the overall system.

| **Metric** | **Result** |
|---|---|
| Recall | 15.71% |
| Precision | 24.44% |
| Accuracy | 9.091% |

Finally, we notice that the overall system has an accuracy of around 10%. This is consistent with our findings for the detection and recognition algorithms. This consistency shows that the different approaches are in sync with each other, however, they all need to be further improved and customized to result in better overall accuracy.

In addition to these evaluations, the team also conducted some individual tests to check the overall flow of the system. Here is an example test case:

**Figure 2:** Showing a sample detected traffic sign along with its generated JSON file.



```
{
   "yield": {
       "x": 660,
       "y": 77,
       "w": 201,
       "h": 182
   }
}
```

Image File /Users/bardiaparmoun/Education/Carleton University/2023

Detection Complete

We can see that the tool correctly identified the image and generated the JSON file.

## Conclusion

As described in the results section, the accuracy of the project is not very optimal. This can be due to various reasons. Firstly, the CNN model that was chosen for the system is not very accurate. Secondly, the sign detection that occurs at the beginning of the pipeline can sometimes miss certain signs. This might be because the image is noisy or the signs are not oriented properly. Thirdly, both template and SIFT matching proved to be very inaccurate. This can be because the images might not be oriented properly. In addition, it can also be because the templates are the ideal representations of the signs.

To improve the project the team recommends finding a better dataset that supports signs in different orientations. In addition, the team recommends adding some homography analysis to the template matching step to resolve misoriented signs. Finally, it is recommended that a different approach is used for locating the signs.

## References

[1] "Road signs in Germany," Wikipedia, https://en.wikipedia.org/wiki/Road_signs_in_Germany (accessed Apr. 7, 2024).

[2] Mapillary, https://www.mapillary.com/dataset/trafficsign (accessed Apr. 7, 2024).

[3] "GTSRB Dataset," German Traffic Sign Benchmarks, https://benchmark.ini.rub.de/gtsrb_dataset.html (accessed Apr. 7, 2024).