Detection of Road Signs

Ali Mohammad Jafar & Mehrad Haghshenas

15/05/2022

1 Introduction

In this paper our goal is to develop a computer vision algorithm for the recognition of a particular class of road signs in images. More precisely, our research is devoted to creating a pipeline to detect prohibition signs within an image. The prohibition signs we have included in our dataset are signs A1, C1, C6 to C22a, F1, F3, F5, F7, and F10 in the overview of Dutch road signs at https://en.wikipedia.org/wiki/Road_signs_in_the_Netherlands [1]. The main feature all these signs have in common is that they are in the shape of a circle and consist of a red annulus; except for the C2 (No Entry) sign which is slightly different. Therefore, in simple words, we aim to develop an algorithm to spot the red annuli that appear in an image. By doing so, we conclude that the image contains a prohibition sign. Accordingly, our research project consisted of the following steps: (1) Practical field work (collecting data) (2) Increasing the size of our dataset by using image processing technics (3) Preprocessing and preparing our data for recognition of prohibition signs (4) Applying *masking* technics for prohibition sign detection (5) Applying *Hough* circle transformation for prohibition signs detection. These sections were the core of our research. However, given the relatively high error ratio of our technique we went further and examined more sophisticated computer vision techniques. (7) Applying *template matching* for prohibition signs detection (8) Applying *Feature Matching* for prohibition signs detection (9) Applying *Blob Detection* for prohibition signs detection (10) Finally, using *Machine Learning* we built a model to detect whether an image contains a prohibition sign or not. After developing our ML algorithm, we will run it on the dataset to calculate the precision and recall of our algorithm. In all sections we will compare the technics and discuss how they effect the quality of the detection. As a result, we will aim to reach an optimized algorithm with a high detection accuracy rate. In all steps of our research, we only aimed to detect traffic signs, i.e., whether in an image a traffic sign is found or not. However, we will at the end discuss further work that can be applied on our algorithm for traffic sign recognition, i.e., what kind of traffic sign is in our image.

2 Practical field work (collecting data)

We have collected the images in our dataset with a cell phone camera. The images were mainly taken from Middelburg, Amsterdam, Rotterdam, and some from Brussels. To expand on our dataset, we also downloaded some images from the internet searching for Dutch prohibition signs on google. The reason we decided to obtain images from the internet is because, the wider our data set is the better it would be to fully test our pipeline especially in the ML section. Our images in the dataset consist of images shot from a straight angle to the prohibition sign, thus having a clearly visible prohibition sign. We also included images containing partly hidden signs, i.e., signs which were partly blocked by objects such as trees. Furthermore, we also considered signs shot at an angle (making the signs elliptic rather than circular). This will challenge our algorithm in the circle Hough Transformation stage. Our images also contained annuli with different colors, such as car tires or traffic signs with a blue annulus. Lastly, we ensured that we took photos using the camera from our smart phones under different lightings and different distances (from camera to sign).

As we travelled around The Netherlands, we were able to capture various types of photos for the mentioned subcategories. The majority of the prohibition traffic signs in the images would still be visible and in reality, humans should be able to identify the type of prohibition traffic sign it is. The test will be if the pipeline that we have built will be able to detect the prohibition traffic sign. In the following we will show some of our photos as an example.

Our first example is an image with *obstruction* which can be seen in figure 1. By obstruction, we mean that the traffic sign was not in complete clear view; it had for example a small object that would partially block the view of the prohibition traffic sign. Figure 2 is another example of the images we took. This image is an A1 sign, i.e., a speed limit sign, which is seen in figure 3 taken from website 1. For further observations of our images we have attached our *dataset* folder containing of three folders: (1) *train* (2) *test* and (3) *Extra_dataset*. Folders (1) and (2) each contain two folders: *Prohibition_Signs* folder and *Other_Signs*. In the former (*Prohibition_Signs*) we have several images of prohibition signs and in the latter (*Other_Signs*) we have several images of other traffic signs which are not considered as prohibition signs. In total the number of images in folders *test & train* is 302 images. The two folders *test & train* were mainly created for training our machine learning algorithm in section 10 which we will discuss in more details. In Folder *Extra_dataset*, we have 54 images all taken from our own cell phone in Middelburg. However, due to using an iPhoneX for capturing the photos, they were captured as ".heic" file which openCV does not support. Therefore, we manually converted them to the ".jpeg" format. Take note that even by doing so, some files are still not supported, but we included them in our dataset for further research. Lastly, as it is noticed in our dataset, we have converted all image files to ".jpeg". The two main image file extensions are ".png" and ".jpeg" and due to the jpeg format being a lossy compressed file

format, which means that it is useful for storing photographs at a *smaller size*, we have preferred this format.

Figure 1: image with obstruction

Figure 2: Prohibition Traffic Sign A1 taken from camera of mobile phone

Figure 3: A1: Speed limit (50 km/h)

3 Increasing the size of our dataset by using image processing technics

To increase the variety and diversity of our dataset, after collecting the images we applied computer vision technics on them. Throughout this stage we manually added foreground noise, background noise, flipped, and rotated our dataset images. We aimed to challenge our pipeline to identify its limitations with feeding it images that have noise, that is images which do not have a clear prohibition sign. The noise adds a different kind of diversity to our data set which is beneficial to us. By adding noise to our images, we ensure that we are able to simulate what the real world is like, because when the camera is detecting the prohibition traffic sign, it never is the most clear image and it always has some sort of "obstruction" or static or has some sort of "fuzz" to the image.

For example figure 4 is one of the images in our dataset. By using the codes shown in figure 6, we have got the image in figure 5. This image is the result of figure 4 with manually added foreground noise. Thus we have added the resulted image to our dataset to add diversity.

Figure 4: Initial Image

Figure 5: Foreground Noise Filter Applied to the image

The way we added speckle noise filter to our images was using the openCV

```python
import cv2
import matplotlib.pyplot as plt
import numpy as np

img = cv2.imread(r"C:\Users\alimo\OneDrive\Desktop\Dataset\10) Vehicle axis weighing Over/10.jpg")

gauss = np.random.normal(0, 1, img.size)
gauss = gauss.reshape(img.shape[0], img.shape[1], img.shape[2]).astype('uint8')
img_gauss = cv2.add(img, gauss)

cv2.imwrite('20.jpeg', img_gauss)
cv2.imshow('a', img_gauss)
cv2.waitKey(0)
```

Figure 6: Foreground Noise Filter Code

library in Python. Figure 6 shows the code that we used to add foreground noise to our images. Basically, what is happening is that, in lines 7 and 8 of the code, we are generating a random array with random pixel values. Once we have created this noise, in line 9 we add the noise to our image.

Another example is flipping our images in the dataset to expand on our dataset. Accordingly, we flipped the images in different directions. We flipped it according to the x-axis, y-axis and the origin. For more information see the "flip" section of the python codes. Figure 7 is an example of expanding our dataset using flipping.

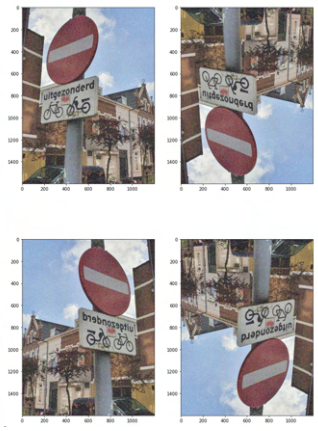


Figure 7: flipped images: top left is our initial image. top right flipped over the origin. bottom left flipped over the y-axis. bottomw right flipped over the x-axis

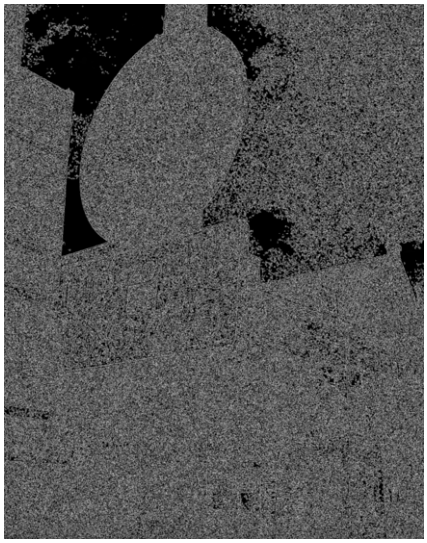
4 Preprocessing and preparing our data for recognition of prohibition signs

This is the first stage of our algorithm. In this stage we feed our data to our algorithm. The first step is to use several image processing technics such as blurring, histogram equalization, thresholding, contour detection and edge detection. All these technics are used to reduce the noise of our data without losing the structure and main features of the image. As a result, our detection algorithm will become less prone to noises in images. Likewise, when we apply this step for instance we apply blurring, our data store is reduced. As a result the speed of our algorithm increases in the next stages given that there are less unnecessary details and edges to deal with.

Now for the first step of the preprocessing, I have taken an image from the data set, which can be seen in figure 8 (a) below. Applying the python built-in blur on the image will result into figure 8 (c). Moreover, applying the gaussian blur, median blur, bilateral blur, morphological openning operator, morphological closing operator will result into figures 9 (a), 9 (c), 10 (a), 10 (c), and 11 (a) respectively. All these blurs were applied using an 11*11 kernel to provide the same condition. For more information and also full sized images see the "blur" section in python codes. As it is noticed even in full sized images, eventhough the blurring does not effect the RGB images that much, the amount of data and details is very much reduced. This can be seen by applying the Canny edge detector on the blurred images. (which are shown to the right of each image). It seems like *the median blur* reduces the details the best; that is, we have reduced the amount of edges but the whole structure of the image is still kept. More importantly, the circle in the image is very much obvious using the median blur. Therefore, out of all our blurring options we have settled with median blur as our preprocessing method.

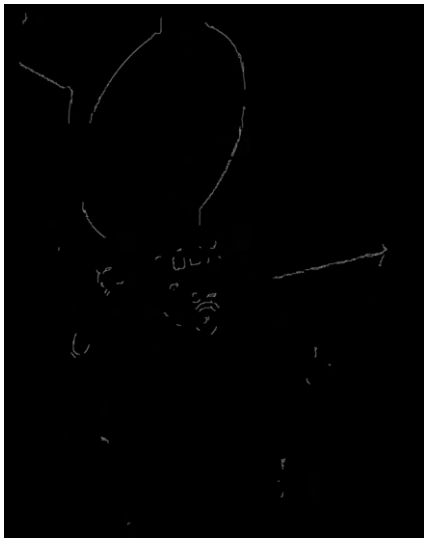
Take note that for using Canny edge operator in OpenCV we have to define an upper and a lower threshold bound. Normally the default and most popular amount of these thresholds is obtained by first of all, finding out the median pixel value in the image. Then multiplying it by 0.7 and 1.3, the results are your lower and upper thresholds respectively. Remember that your lower threshold must not be lower than 0 and the higher threshold is maximum 255. The reason that we have used Canny edge detector over Sobel, Prewitt or Laplacian edge operators is that the Canny edge operator is a multi step process and for our purpose provides a more accurate operator.

(a) 8.initial image



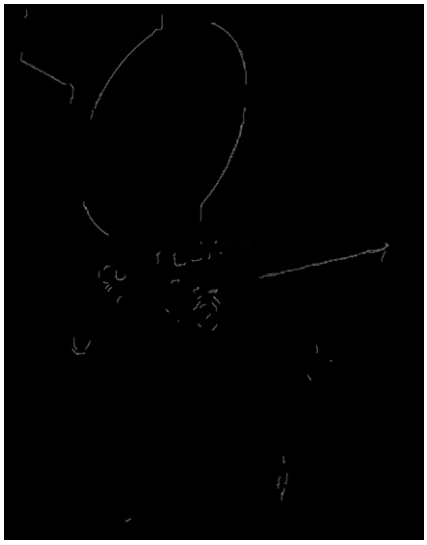
(b) 8.initial image with Canny edge operator

(c) 8.image blurred with buil-it python blurring function



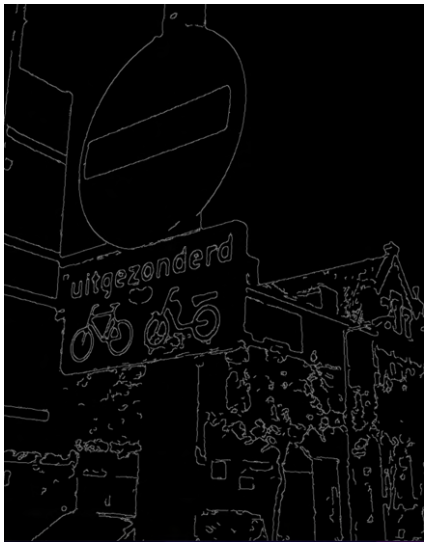
(d) 8.image blurred with buil-it python blurring function with Canny edge operator

(a) 9.image gaussian blurred



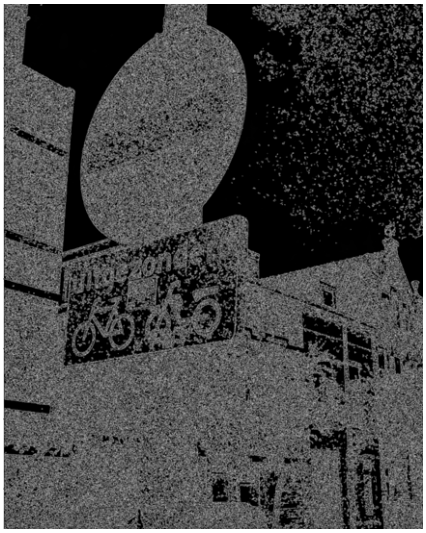
(b) 9.image gaussian blurred with Canny edge operator

(c) 9.image median blurred



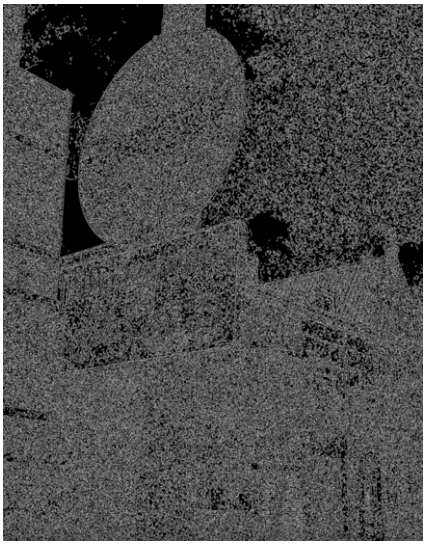
(d) 9.image median blurred with Canny edge operator

(a) 10.image bilateral blurred



(b) 10.image bilateral blurred with Canny edge operator

(c) 10.image morphological operator opening filtered



(d) 10.image morphological operator opening filtered with Canny edge operator

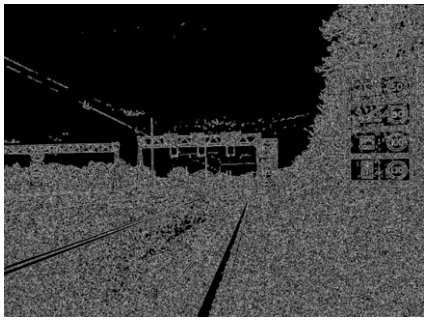
(a) 11.image morphological operator closing filtered

(b) 11.image morphological operator closing filtered with Canny edge operator

Moreover, applying histogram equalization, we noticed that our edges do not become less. In fact the number of our details increases, thus this method is not useful for our preprocessing. This can be seen in the below images. For more information and full-sized images see section "Hist_equal" section of the python codes.

(a) 12.initial image

(b) 12.image with histogram equalization

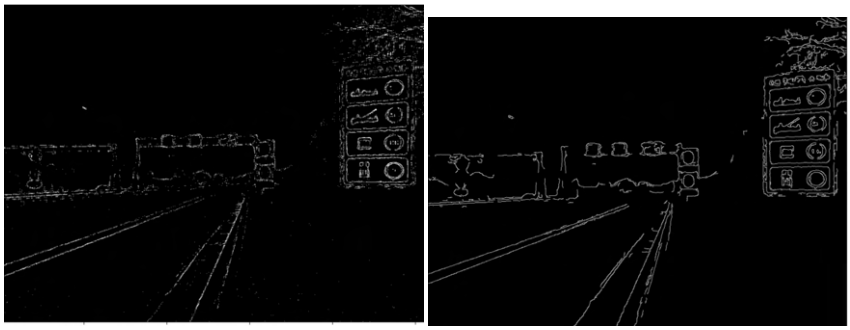


(c) 12.image with Canny edge operator

(d) 12.image with histogram equalization with Canny edge operator

Lastly, we wanted to experiment that will using adaptive thresholding benefit us compared to Canny edge operator or not. Therefore, after reading an image from the dataset and applying median blur, we applied Canny edge operator and adaptive thresholding separately. The images below resulted. As it is seen the difference is very low, but if you see the traffic signs at the far end, the circles are more clear using the canny edge operator compared to adaptive thresholding. For full information see the "thresh" section of the python codes.

All in all comparing the technics, we concluded that using median blur combined with the Canny edge operator will remove the unnecessary details without damaging the whole structure. Also the main edges in the image sill be kept.



(a) 13.adaptive thresholding (b) 13.Canny edge operator

5 Applying masking technics for prohibition sign detection

Masking images is a very popular technique. It is mainly used for object detection and object tracking. Given that prohibition signs have a red annulus, we are going to using masking technics to only indicate the red parts of our image. We will apply this examination on various images with different pixel values for the red channel and will observe the result.

In masking we first convert our image to HSV. Then given that the red channel in HSV contains hues from 0-10 or 170-180 we create the two masks using these ranges. We get our ultimate mask by combining these two masks. For further information see the "Masking" section in the python codes. Obviously by creating the mask, and blending the mask and the initial image together we will get a form of the initial image with the red more emphasized. This can be seen in figure 14.

As our prohibition sign contains a red annuli, masking will help us in detecting the prohibition sign. However, as it can be seen in figure 14 (b), if there are alot of red, brown, yellow, etc. parts in our image we will have a lot of noisy red parts in our image. This will negatively affect our masking compared to if we only had limited red areas such as figure 15 in which by solely masking the image we have detected the traffic sign.

To sum up our first algorithm, based on sections 4 and 5, for an image in the dataset we will first apply median blur with kernel size 11*11 on it. Subsequently, canny edge detector on the median blurred image will be applied. This will give us an image with the main edges of the initial image most likely preserving the circle of the traffic sign. Lets call this image result image 1. We will then create a mask indictaing the red parts of our initial image. We will blend the mask and our result image 1 to get our masked image; i.e, an image

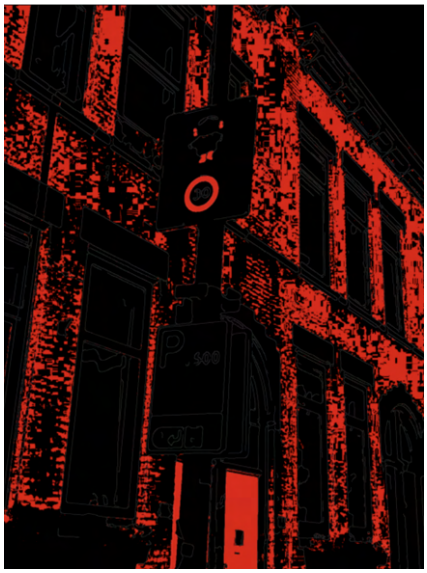
(a) 14.Initial image

(b) 14.masked image

(a) 15.Initial image

(b) 15.masked image

(a) 16. image resulted from blending the mask and the initial image



(b) 16. image resulted from blending the mask and the edges (edges is the result image from median blur followed by Canny edge operator)

with the edges and the red parts much more emphasized compared to the initial image. Therefore, the traffic signs are more indicated in the image. Figure 16 (a) compared to Figure 16 (b) has more unnecessary details. Concretely, in Figure 16 (b) the edges of the circle and the red parts are much more emphasized.

6 Applying Hough circle transformation for prohibition signs detection

This is where we apply Hough Transformation on our dataset image. In this section we will first discuss and explicate what Hough Transformation is. We will then run Hough Transformation in OpenCV and explain how to use Hough Transform in python to detect a circle inside an image. We will also change the parameters to reach an optimal algorithm.

6.1 Definition of Hough transform

The Hough Transform is a method for finding lines, circles, or other simple forms in an image. In our project, we only use Hough Transformation to detect circles. We know that a circle can be defined by having the location of the center and the radius of the circle. We, therefore, need three parameters to define a circle: x, y, r where (x, y) defines the center of circle's position and r is the radius of the circle.

6.2 Hough Transform in openCV

OpenCV for efficiency implements a method slightly different than the standard Hough Transform, which is called the Hough gradient method. This method consists of two main stages: 1- Edge detection and finding the possible circle centers. 2- Finding the best radius for each candidate center. The Hough circle transform function for OpenCV is cv.HoughCircles(). The first argument is the input image which is an 8-bit image, i.e., it must be a gray-scale image and we cannot give a RGB image as an input to the Hough transform function. The cv.HoughCircles() function internally has a built-in cv.Sobel() function inside. Thus, when calling the Hough circle transformation, it will apply automatically apply the Sobel edge detection operator as well. The method argument must always be CV_HOUGH_GRADIENT, as OpenCV has no other method arguments for Hough Transformation. The parameter "dp" is the resolution and takes a number equal to or greater than one. If dp=1 then the resolutions will be the same; if set to a larger number then the resolution will become smaller by that factor. The parameter min_dist is the minimum distance that must exist between two circles for the algorithm to detect them distinct circles. This is probably the most important parameter affecting our Hough Transformation algorithm. param1 is the edge Canny threshold, thus cv.HoughCircles() has a built-in Canny edge detection operator as well. param2 is the accumulator threshold which is used to threshold the accumulator. The final two parameters are the minimum and maximum radius of circles that can be found.

As it can be seen in figure 17 (a) the initial image is shown. In 17 (b), the initial image with Hough circle transformation applied, is shown. The parameters of our function were cv.HOUGH_GRADIENT, 1, rows/6, param1=100, param2=50, minRadius=50, maxRadius=200. For 17 (c) again we applied

(a) 17. initial image

(b) 17. initial image with hough

(c) 17. initial image with hough

(d) 17. result of previous algorithm with hough

Hough circle transformation. The parameters we used were exactly the same except for min_Dist which we changed to rows. (rows here as the name indicates is the number of rows of our image). When we multiplied our min_Dist by 6 we got a exactly correct detection, however, in 17 (b) along with the actual prohibiton sign, we got a false positive. The most disappointing result was 17 (d), where we only got one asnwer which was incorrect. We already said that by applying the pre-mentioned final algorithm in section 5 we will emphasize more on the traffic sign as we emphasize more on the general edges and red part. Therefore, our expectation was that the Hough tranformation will act better when receiving such an image instead of the initial image. However, our expectations turned out wrong as can be seen in figure 17. Having said that, in some occassions masking does seem to give better results. in the figure below, the right images are Hough transformation applied on masked images. On the left hand side, Hough tranformation is applied on the image. We can see that masking has improved our results.

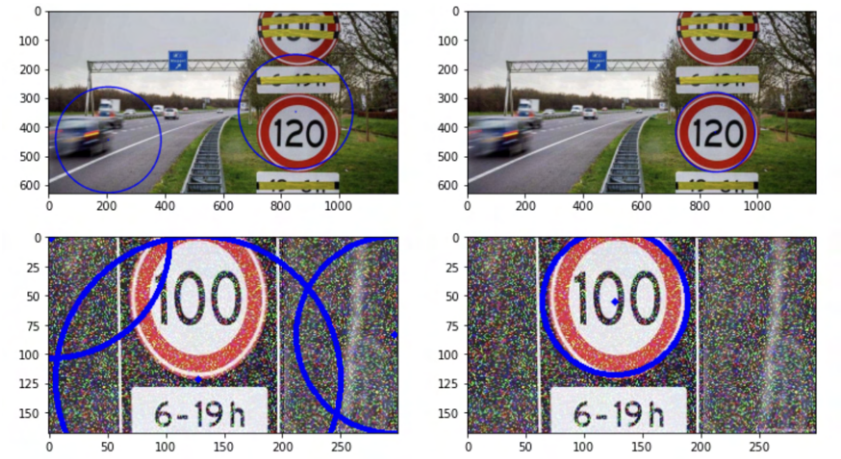


Figure 18: Hough Tranformation results

We have run Hough tranformation on a high proportion of our dataset and also on the masked version of the images. The results can be found in the "research" section of the python codes. Images on the right are Hough tranformation on the masked images. On the left hand side, are the Hough tranformation on the initial image. The results show that the Hough tranformation does not necessarily apply better on images where the edges have been sharped or the image has been saturated.

In figure 19, we can see Hough transformation applied on an artificial image download from google. The codes can be seen in the "experiment" section of the python codes. The image contains a set of traffic signs. Using Hough

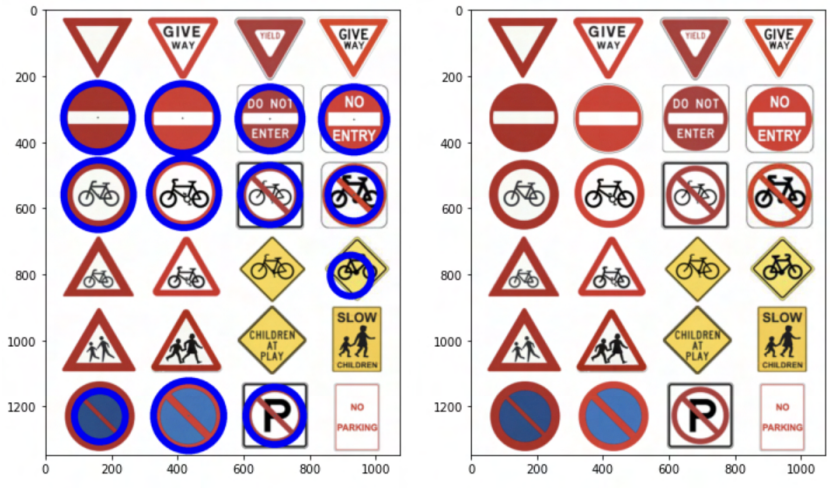


Figure 19: Hough Tranformation results

transformation and playing with the parameters, we have got exactly all the circles in the image. However, we have got one false positive but the accuracy of our predicition is very decent. This is mainly due to the fact that Hough transformation acts very well when the images are taken from straight forward like figure 17. That is with giving it the right parameters and playing around with the parameters you can get a decent detection. Nonetheless, the parameters are not similar for each image. They very much depend on the size and shape of the image and number of traffic signs. The distance of the image from the lens is very important as well, as the more the distance is the smaller the traffic signs are and the min radius has to be set to a lower threshold. Basically Hough transformation does not give you a sophisticated way for detecting traffic signs in all images.

7 Applying template matching for prohibition signs detection

In this section we will be applying template matching for road sign detection. First let us define query image and target image. A query image is the image which we want to find out whether it exists in the larger image named as the target image. Finding out whether it exists or not is the process of template matching. How can template matching be used for prohibition sign detection? If you have an image of a prohibition sign as your query image. Then if it exists in you target image, you will conclude that the target image has a prohibition sign. If it does not exist in the target image then you conclude that the target image does not contain a prohibition sign. For template matching to actually work your query image has to be part of the target image that is the pixel values must be really the same. Any rotation, blurring, shifting etc. will affect the template matching process. Thus, this algorithm will not be practical for road sign detection in most real-world applications.

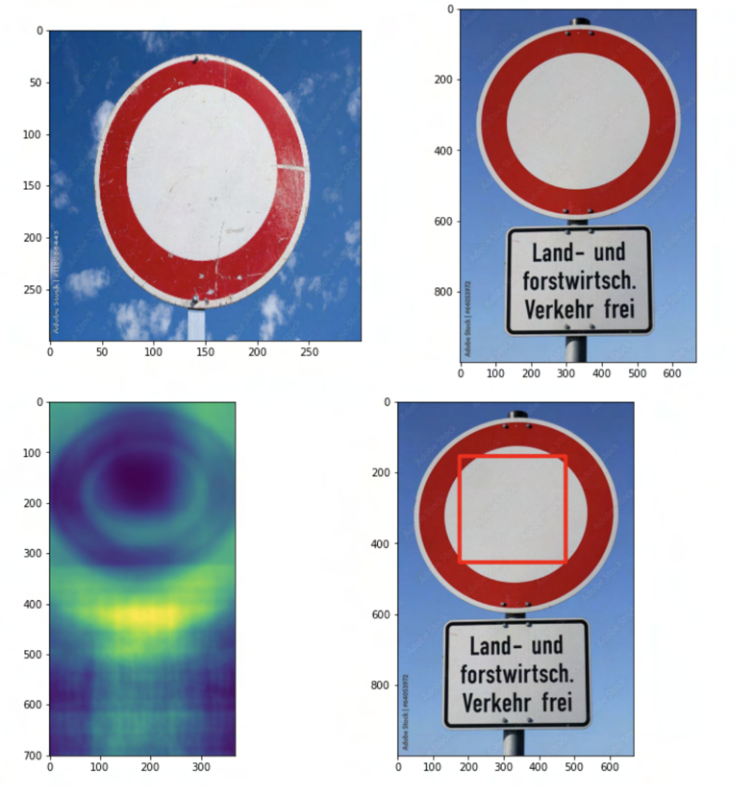


Figure 20: Template matching result

In the figure 20 we can see the query image in the top left. It was resized to (300,300) as the query image must be smaller than the target image. Then we used the cv.matchTemplate() function. The method we gave it was the sum of squared differences. Basically what this algorithm does is that it parses over the target image and calculates the sum of the squared differences of the pixels values and window (in this case query image). It then generates a heat map like image bottom left. The black parts show the minimum difference indicating that our query image, prohibition sign, is located there in the target image. We then draw a rectangle in that area to show the position. In this example template matching did pretty well. However, in the next example it did not given that the query image was slightly different than the target image altough both being a speed limit sign. The codes for this section can be found in "templateMatch" section of python codes.

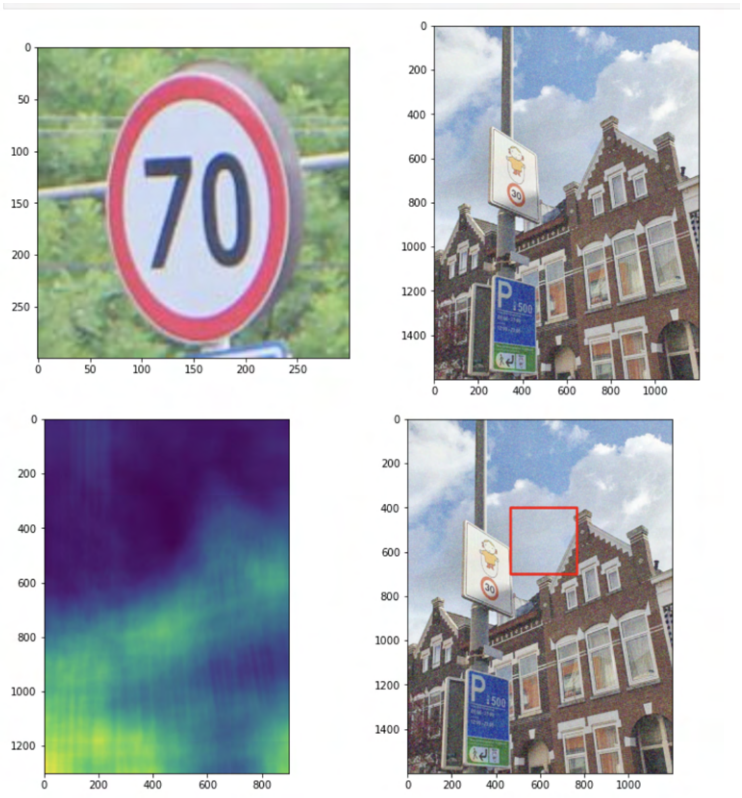


Figure 21: Template matching result

8 Applying Feature Matching for prohibition signs detection

We have reached one of the most sophisticated computer vision technics for road sign detection, Feature Matching! Here we will discuss the Brute-force matching with SIFT descriptors with additional ratio test technic. In this section like the previous section, we have a query image and a target image. However, here we find the key points of each image and match them to each other. Concretely, if the number of keypoints matching is high this indicates that the target images consists of the query image or some form of it.

Figure 22: SIFT result

As it is noticed in figure 22, most matching features were discovered in the center to the top of the target image (larger image). This was approximately accurate.

However, in figure 23, SIFT did not act accurately eventhough the two images in a humans eye have many features in common. For more information see the "SIFT" section of the python codes.

Figure 23: SIFT result

9 Applying Blob Detection for prohibition signs detection

In particular, we will address how to detect the prohibition traffic signs where it somewhat resembles an ellipse in the image rather than a circle. The reason for this change in "shape" is due to the angle that the prohibition traffic sign was taken from. To detect the ellipses, we are going to use a method known as Blob Detection.
To put it simply, "A Blob is a group of connected pixels in an image that share some common property." The four parameters important in blob detection are area, circularity, convexity, and inertia. Below is a description of how each parameter is varied:

9.1 Filter by area

In order to avoid small dots in the image that may be mistaken for a circle, we instead then set a particular number of pixels for an approximate area of a circle that we are looking for.

9.2 Filter by circularity

The closer the shape/blob is to a circle, the more likely it is that is the object that we are looking for in the image. To be able to implement this, we use the following formula:

$$Circularity = (4 * pi * Area)/((perimeter)^2)$$

The closer the value is to 1, the more likely the blob is to resemble the shape of a circle.

9.3 Filter by convexity

As convexity increases, the closer the blob is to a circle. Concavity can be calculated by:

$$Concavity : (Areaofblob)/(AreaofConvexHull)$$

9.4 Filter by inertia

The inertial value of a circle = 1 however, we are looking for ellipses. Thus, the inertial value of ellipses are between 0 and 1 however, the higher the inertial value, the closer it is to resembling a circle. Hence, in our case in many of the images (taken from an angle), the prohibition traffic signs resemble that of "an ellipse" but the inertial value will be very close to 1.

To test out if we can detect how accurate our code detects ellipses, we decided to use the filter by convexity, filter by area, filter by circularity and filter by inertia functions. Through trial and error, we were able to find set the values for each of the parameters. Below is an example of the blob detector on one of our images: For the purpose of highlighting where the blobs were spotted, the image was turned to greyscale however, the blob detector has the same effect on coloured images.

Figure 24: Blob Detection result

As it can be seen the blob detection did not have any false positives, but it did have false negatives.

10 Using Machine Learning

The last method we will use is machine learning. Over the past sections we saw that by explicitly programming the algorithm, the error of our results in detecting the traffic signs were high. Machine learning is used especially in situations where explicitly programming does not get the desired result. For this we have created the dataset folder and the *train* and *test* folder. For a Machine Learning program, the number of examples we train our model should be approximately 70% of the whole data. (It can be more or less but this is a good default value). Therefore, we have allocated 40 images to our "Other_Signs" folder in the "test" set (30.76%) and 90 to the "Other_Signs" in the "train" set (69.24%). Moreover, we have allocated 118 images to the "Prohibition_Signs" folder in the "train" set(70.24%) and 50 images to images to the "Prohibition_Signs" folder in the "test" set (29.76%).

The first step in our Machine Learning algorithm is preparing the data. Moreover, we will diversify the images in our dataset so that if the machine sees an image rotated, flipped, shifted etc. it will still recognize the image. All these processes are done with built-in functions in the keras library of python. Next

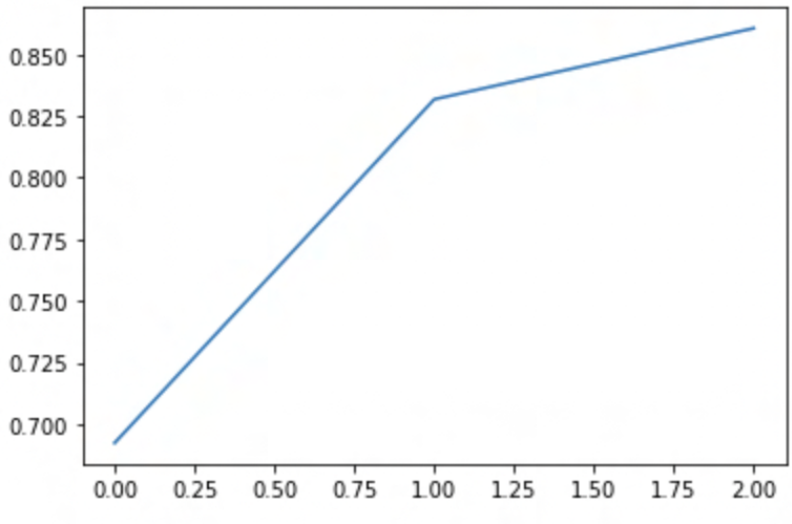


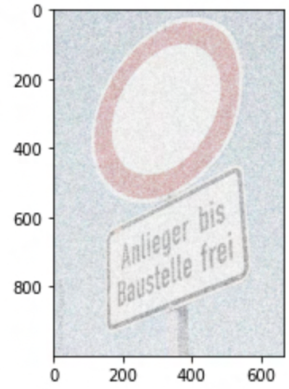
Figure 25: Accuracy

step is building out neural network model. For this we have used one input layer, one output layer and one hidden layer. The activation functions of the layers are the "Relu" function. We also take care of overfitting by using the Dropout method. (Overfitting is that your model is well-trained on your data but does not operate well on unseen data). For the training of our model we have set epochs=3, that is for training the model it goes through the network 3 times. Take note that the more the epochs is set to, the more your model is trained on your dataset, and consequently the higher the accuracy of your trained model. In each of my three epochs, I have received accuracies, 69.23%, 83.17%, and 86.06% respectively. The plot corresponding to these numbers is shown in figure 25.

Lasltly after training the model on our "train" dataset, I have tested the model on the "Prohibiton_Signs" of the "test" dataset (evaluation). For this I have put a threshold of 50%, that is if our model predicts an image is a prohibition sign with a probability higher than 50% it returns it as a prohibition sign otherwise it indicates that there is no prohibition sign in the image. The answer for all of them should be True as they are selected from the "Prohibiton_Signs" folder. Interestingly, our model indicates 47 out of 50 True and only 3 were misdetected as shown in figures (26), (27) and (28):

For figure 26 and 27 the reason can be that the blurring is affecting the detection adversely. Or maybe we have limited images of this kind (just a red annuli) in the training data set so that the model is not trained to detect this

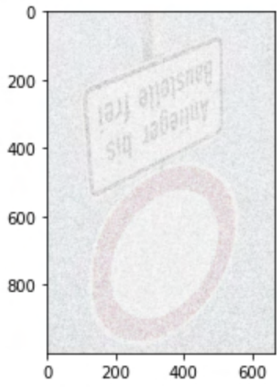
```
Mistakenly detected as Not a prohibition sign
```




```
Probability that image is a prohibition sign is: [[0.498931]]
------------------------------------------------
```

Figure 26: Misdetected as not being a Prohibition sign

```
Mistakenly detected as Not a prohibition sign
```




```
Probability that image is a prohibition sign is: [[0.49636963]]
------------------------------------------------
```

Figure 27: Misdetected as not being a Prohibition sign

```
Mistakenly detected as Not a prohibition sign
```


```
Probability that image is a prohibition sign is: [[0.48562133]]
-----------------------------------------------
```

Figure 28: Misdetected as not being a Prohibition sign

kind well. Lastly, given the low amount of our dataset our model predicted well and we had 47 True Positives and 3 False negatives.

Figure 29: Misdetected as being a Prohibition sign

In the last step we aim to find out the number of False positives. For this we run our algorithm over the "Other_Signs" of the "test" dataset. All these signs should be detected as not prohibition sign, however, we have 14 false positives out of 40 images. Some of the false positives are shown in figures (29), (30) and (31)

Our percision $(TP/(TP + FP))$, TP means True Positive and FP means False Positive, will be 47/61 (77.04%) and recall $(TP/(TP + FN))$, FN means False Negative, is 47/50 (94%). The F1 score being the harmonic mean of percision and recall will be approximately 84% which is a decent percentage considering the size of our dataset.

For the last part we gave a tire image to our trained model and unfortunately it detected it was a traffic sign with a probability of above 91 percent. For further information go to the "ML" file of the python codes folder. Take note that when running the python codes change the directory to the dataset location in your computer.

```
Mistakenly detected as a Prohibition sign
```


```
Probability that image is a prohibition sign is: [[0.7560091]]
-----------------------------------------------
```

Figure 30: Misdetected as being a Prohibition sign

```
Mistakenly detected as a Prohibition sign
```

```
Probability that image is a prohibition sign is: [[0.81799126]]
-----------------------------------------------
```

Figure 31: Misdetected as being a Prohibition sign

11 Conclusion

Between explicitly programming complex programs and using machine learning to train a prediction model, the later is much more efficient. For explicitly programmig we have different methods. Feature matching detects the key points and matches them, Template matching sees whether a smaller image is present in a larger one. Bother are relatively sophisticated. Hough tranformation does not generalize well on ellipses and other circular liked shapes. The shape must be a very clean circle and the photo should somewhat be taken straightly for the Hough tranform to detect it. Ultimately, when we increase the sharpness of edges and the redness of our image using Canny edge operator and masking respectively, opposite to our expectations, Hough transformation does not operate better, that is in detecting the circle it does not benefit from the sharped edges and the efficiency is relatively the same. All these methods are used to detect the red annuli which is the most important feature of prohibition signs and can be used further to detect prohibiton signs.

12 Further Work

For further work we can first of all expand on our relatively small dataset. Moreover, we just did prohibition sign detection. One step further we can do image classficiation that is detecting the kind of prohibition sign in an image. To do so, the process is somewhat similar to the machine learning algorithm we use here. There is only two main difference: (1) in Each of our "test" and "train" folders there should be the folders of the different classes of prohibition signs. (2) we shall not use a binary nerual network as there are multiple classes.

13 References

[1] https://en.wikipedia.org/wiki/Road_signs_in_the_Netherlands
[2] https://developpaper.com/traffic-sign-recognition-based-on-opencv/
[3] https://docs.opencv.org
[4] stackoverflow.com
[5] geeksforgeeks.org
[6] https://www.youtube.com/watch?v=O2tCKwFLY5g openCV SIFT Feature Tracking
[7] https://learnopencv.com
[8] https://towardsdatascience.com/essential-things-you-need-to-know-about-f1-score
[9] Gary Bradski and Adrian Kaehler, Learning OpenCV, O'Reilly Media, 2008