Traffic Sign Recognition

Jose Pablo Cereceda and Diego Martin, CS512 Computer Vision, Fall 2018
Illinois Intitute of Technology (IIT)

***Abstract*— This paper presents a graduate course project on computer vision. The aim of the project is to detect and recognize traffic signs in images. This is a demanding problem, given that traffic sign recognition is one of the most important thing on driving assisted vehicles. It is also an extra motivation for the students, as it offers the opportunity to work on a real-word problem and apply the techniques learned along the course. The main objectives of the project are introduced, as well as the steps performed to arrive to the final approach. The results obtained are evaluated and some improvements are proposed.**

I. INTRODUCTION

Nowadays technology has reached a point in which intelligent systems are substituting some of the daily activities that humans perform. From highly precise actions to game related interests, the influence of artificial intelligence is on the rise.

One of the fields in which this kind of technology is in increasing adaptation is the automotive industry. With the appearance of companies such as Tesla, vehicles have introduced the use of intelligent systems for a great variety of utilities. However, the main goal that every car related company wants to reach is the autonomous driving. To be able to get to that goal, the problem has been divided in several parts. One of them is the recognition of traffic signals, so that the car can behave adequately to the road regulation.

The purpose of this project is to program a simple traffic sign recognition program based on computer vision and learning techniques. This program would be able to detect on an image where the main traffic sign is located and also to recognize which traffic signal is. This order of actions is based on the approach followed by [1].

Since there are a lot of traffic signs, the students are going to use a limited number of these to create the program, in order to simplify the problem. Therefore, the program should distinct which signal is detecting between a short amount of them. The images are obtained from the dataset KUL traffic sign database [2], created by Dr. Radu Timofte from the Katholieke Universiteit of Leuven, Leuven, Belgium. This database is now owned by the ETH Zurich [3]. The dataset was adapted to the specific needs of the project, selecting the number of images and those traffic sign types with more samples, and picking the traffic signs that would be easier to detect on shape.

On section III the changes performed to the database are going to be explained, but the final data used is composed by 3,732 images, divided in 14 classes not equitably. From these images, the 90% is going to be used to training and the remaining 10% is going to be used for testing purposes.

II. APPROACH

To face the mentioned problem, the students have decided to divide the problem on three main phases: image preprocessing, detection of candidate and classification. This are going to be explained in this section.

A. Image preprocessing

Given an input image, image preprocessing consist on making several operations on that input image with the objective of facilitate the work to the next phase as much as possible.

The proposed solution does the following steps:

- Limit the contrast of the image. This operation allows the image to have a more uniform range of colors and avoid really bright or really dark images that could lead to an incorrect detection. This operation should be performed without damaging the edges of the objects on the image.
- Edge detection. This operation is a key part of the process, as it is the main pillar in which the detection phase will support to realize its work. It should clearly show the edges of the image on a proper way.
- Convert the image to binary. This operation allows to work with a simpler image in which colors are not longer a feature to take into consideration when detecting a sign.

After performing this steps on a sequential way, the input image is ready to go through the detection phase, as it only consist on borders defined by white pixels or background defined by black pixels.

B. Detection of candidate

A detection system consist on an algorithm that given an input preprocessed image, it outputs a possible sign object to be classified. To reach that goal, it is typically used the color of the images or the shapes of the objects that are being detected.

Since the usual shape of the signals is well defined, as they mainly are either triangular or circular, the second approach has been selected to implement this step. However, since the images that are going to be analyzed are pictures of roads, cities and highways, they are likely to have a lot of features involving shapes and edges. Therefore, a first operation of cleaning small contours is necessary. After this cleaning is done, the image should be ready to find the main shapes of the signals.

To simplify the classification process, it has been decided that the output of this phase is a single detected sign. This sign will be the one with larger area. This decision has been made from the idea that the signals are usually placed one after the other, so the car should focus on the closer sign first. This closer sign is bigger in size on the captured image than the rest of signs, therefore, it has a larger area.

C. Classification

The classification system consist on an algorithm that loads a series of traffic sign images. Those images belong to one of the defined categories or labels and they are already cropped with the traffic sign contour. The algorithm will use these images to create a model for future prediction and classification of the images processed by the detection algorithm.

The model uses a Histogram of Oriented Gradients (HOG). A HOG is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

- Gradient computation. The first step of calculation in many feature detectors in image preprocessing is to ensure normalized color and gamma values.
- Orientation binning. The second step of calculation is creating the cell histograms. Each pixel within the cell casts a weighted vote for an orientation-based histogram channel based on the values found in the gradient computation.
- Descriptor blocks. To account for changes in illumination and contrast, the gradient strengths must be locally normalized, which requires grouping the cells together into larger, spatially connected blocks. The HOG descriptor is then the concatenated vector of the components of the normalized cell histograms from all of the block regions.
- Block Normalization.
- Object recognition. HOG descriptors may be used for object recognition by providing them as features to a machine learning algorithm. Navneet Dalal and Bill Triggs used HOG descriptors as features in a support vector machine (SVM); however, HOG descriptors are not tied to a specific machine learning algorithm.

This way the features for each image are extracted and then fed to the Support Vector Machine (SVM).

The reason for using a SVM are:

- It has been to use a new classification method that has not been implemented yet in the course of Computer Vision.
- It has high accuracy and high flexibility.
- It can naturally handle large dimensional data.
- It can made a sparse representation of the solutions (via support vectors), which is fast for making future prediction.
- This method involves less computational load than others like a Convolutional Neural Network (CNN)

Once the model is created, the detected images processed by the detection stage are passed to the SVM and it will predict its category or label.

III. Implementation details

In the following section the implementation of the approach will be explained. The structure, programming code and its functions will be detailed.

A. Image Data Modeling

To create a model the election of the training data is essential in order to define a good classifier. The set of images have been taken from ”Linkpings University

traffic sign dataset". Initially 17 classes where defined according to the different images from the dataset. The images were saved into folders by categories so later it would be easier to load them to the model.

Once this structure of folders was created the programming began. After several tests it was noticed that some of the images were not labeled or classified correctly. The reason was that the dataset was not cleaned correctly (there were some useless images that only made a worse model). Moreover, there were some of the categories (entire folder of one category) that produced errors in the classification.

After cleaning the dataset by removing some of the useless images and some of the categories the classification improved. Finally the data modeling was composed by 14 categories. The model will classify 13 different traffic signs. There was an extra category for possible misclassification. This category contained images of possible road and traffic objects that may be detected as traffic signs when they are really not one of them.

B. Image processing

The image processing was performed following the three steps exposed on section II. On this section, a more profound explanation of this steps is going to be made.

- Limit the contrast of the image. To be able to achieve this part, four *OpenCV* functions have been used. The first one is *cvtColor*, in order to change the color system of the image from BGR to YCrCb. This new color systems allows to access the value of luminace easier, which is the one that affects more to the contrast of the picture. The function stretches the histogram representation of the image, adjusting its contrast. In figure 1 can bee seen a representation of the action of this function. Finally, the modified channel is merged with the chrominance ones and changed back to a BGR representation.
- Edge detection. The first action to do on edge detection is noise reduction. To achieve this, the OpenCV function of *bilateralFilter* has been used. This type of filter is useful for this purpose because it reduces the noise of the image while keeping the edges sharp. Then, Canny edge detection algorithm has been chosen as the edge detection algorithm due to its ability to discard "free" edges.
- Convert the image to binary. It uses the OpenCV function of *threshold* to convert the image to binary.

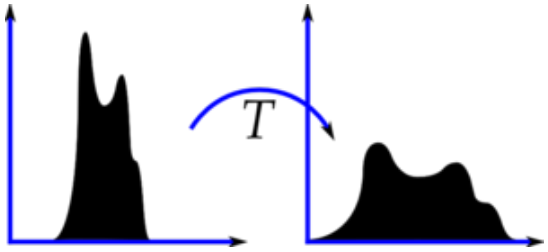


Fig. 1. Histogram equalization

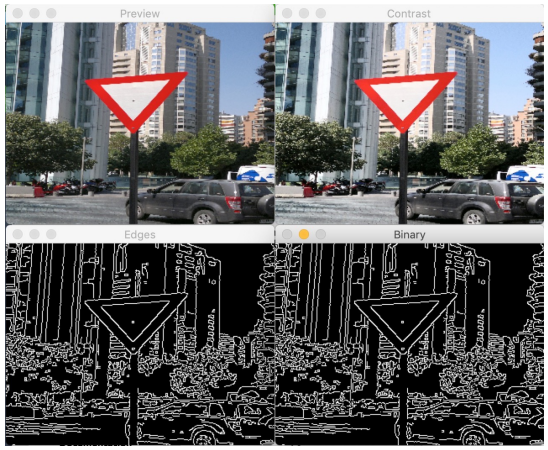


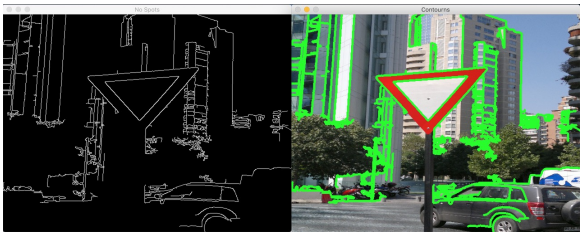
Fig. 2. Image preprocessing

The Images obtained by this three methods can be seen in figure 2.

C. Detection stage

To implement the detection phase, some sub-steps have been followed. This are the following:

- Remove small components. This function obtains information about all the components of the image by using the OpenCV function *connectedComponentsWithStats*. This function returns several parameters, but only the size and the number of components are useful for the purpose that concerns this method. Using these two parameters and a threshold, an image without the small components can be composed.
- Find contours. This method uses the OpenCV function of *findContours* to return the coordinates of the contours of the image.
- Find sign candidates. As mentioned on section II, the approach to detect possible sign candidates was based on shape. To do so, a method that uses the previously obtained contours are used. this method uses the functions *arcLength* and *approxPolyDP* to recognize triangular and circular



(a) Spot removal and contours

(b) Contours candidates and final choice

Fig. 3. Candidate detection

shapes. The first method calculates a curve length from a given contour. This result is passed scaled as a parameter to the second function, which will approximate the given contour and curve length to a polygonal curve. By calculating the length of this result the type of polygon can be deduced.

- Select sign candidate. This process is done following the rule that the biggest detected candidate is a sign. Therefore, this method iterates on the candidates found and calculates their corresponding area. The one with larger area will be considered the sign to classify. In order to pass the classifier the correct portion of the image that contains the sign, a function that crops that portion has been implemented. This function takes the coordinates of the contour and the image and selects the minimum rectangle that contains the sign.

A graphical representation of the steps of this phase on a sample image can be seen on figure 3

D. Classification stage

In the classification stage the following steps have been followed:

- Loading data images. Once the structure of the folders was created the function *load-traffic-dataset* was implemented. It load all the folders, each one with its different categorized images from the folder of *../data/Samples/*. The variable CLASS-NUMBER = 14 tells the function the number of labels (folders of different category). The variable MAX-FILES = 80 defines the number of images of each label that the model will take. The function resizes the images to the defined size = 32 of 32x32 pixels for later train the SVM model. It returns two arrays, one with the images and other with the correspondent labels.
- SVM class definition. This class defines the SVM model functions. There are 4 main functions. The initializer function that created the SVM model with the different parameters. For instance the type C-SVC that can be used for n-class classification ($n \geq 2$) is chosen here. Also there are created other three functions to train the model, predict a sample or save the model as a file.
- Get HOG descriptors. In the function *get-hog* all the parameters of the HOG descriptor are defined. Here the parameters like the winSize, blocksize or cellsize are defined. All of the mentioned should be smaller than the size of the image of 32x32 pixels. Also at the test process the parameters of the HOG will be variated in order to get a better accuracy.
- Deskew images. The function *deskew* takes as input an image. It computes the second order moments of the image using OpenCV *moments* function. This function makes images to have the same deskew. It returns the deskew image. In Figure 4 it is showed the effect of the deskew process (deskewed image at the right side of the figure). Also in the next Figure 5 it showed the effect in an input tested image.
- Training process. The function *training* uses all the previous functions to classify the images. First all the data and labels are loaded and both are shuffled. After the HOG features are extracted from the images. Finally 90% of HOG images features are used for training the model. The other 10% is left for testing.
- Test process. The function *evaluate-model* takes the left 10% of the data in the training process and calculates the accuracy for the model. In this

Fig. 4. Image Deskewed

Fig. 5. Test Image Deskewed

process parameters from the model such us the number of bins from the HOG or the MAX-NUMBER of images will be modified to see how the accuracy of the model performs.

- Get label. The function *getLabel* returns the label of the image to be predicted. It is called in the detection stage when the images are loaded. This function takes as inputs the model and the image to be predicted. Previous to get the HOG descriptors using the *det-hog* function the image needs to be converted to grayscale, resized to the same model size and skewed the same way of the trained model. Then the HOG function is called and it returns an array of length 324 items with the features of the image to be predicted. This array will be fed to the model to predict the label of the image.

IV. Results and discussion

In this sections, the results obtained after testing the implementation are going to be shown. A discussion of how the results were obtained in each phase is going to be done, specifying the problems encountered and the solutions taken.

A. Image preprocessing

Since this section is the first one of the program, it was important to obtain a good result of the edges of the image. Several approaches were tested in order to have the best result possible, mainly on the edge detection step. Some of them are:

- Apply a Gaussian Filter to reduce noise and then use the Laplacian edge detector. This approach was not good for two main reasons. The first one is related with the Gaussian filter, that blurs the image more than needed, not allowing to detect the borders properly. The second one is related with the Laplacian detector, that detects a great amount of edges that are useless for the purpose of sign detection.
- Apply a bilateral Filter to reduce noise and the Laplacian edge detector. This approach was not good either, as it had the same problem stated previously, but with more edges, as the bilateral filter provides with a sharper view of them.
- Apply Canny edge detection without previous filtering. The Canny algorithm already uses a Gaussian filter to reduce noise, so in theory, it does not need an additional filter. However, as mentioned on the first point, this filter could remove edges that may be important for the sign detection task.

After testing with these different approaches, it was decided that the best approach was using a bilateral filter in addition with the Canny edge detection algorithm.

B. Detection Stage

To test this phase, three different steps were taken: check that the contours were found correctly, check that the detected polygons have the correct shape and check that the largest shape is taken. Since the first step has not any difficulty as it uses the OpenCV function with the standard parameters to reach its goal, in this section is going to be discussed the two remaining steps.

As it was mentioned on section III, the OpenCV functions of *arcLength* and *approxPolyDP* were used to obtain the polygons with the desired shapes. The difficult part of this section is to find an adequate value of the epsilon that is passed as a parameter to the second function. As it can be seen in [4] this epsilon is calculated as a percentage of the arc length of a contour. This percentage indicates the distance from contour to approximated contour. This percentage is usually in between the values 0.01 and 0.1, being the fist one a low approximation and 0.1 a close approximation. In figure 6 an example of this can be seen. The middle image uses a factor of 0.1 of the arc length to find polygons and the right image a factor of 0.01. Knowing this and taking into account the types of images that are going to be used, some experiments were performed. To detect adequately triangular signs, a low value of the percentage is needed, as this type of signs are not sharp triangles, as the vertices are usually round. On the other hand, the circular signs are more recognizable, as circles have a more defined shape. Therefore, it can be used a higher value of the percentage to detect them. The final values used on the program were 0.01 for triangle detection and 0.07 for circle detection. Then, the value obtained from the *approxPolyDP* is used to

Fig. 6. Effect of epsilon value

detect how many sides the polygon has. if it has three is a triangle, if it has more than seven, it is considered as a circle. However, the obtained result is not as good as expected, as there are several contours in the images that also have a similar circular or triangular shape, or at least that the program mistakes them for that, for example is there are shapes with irregular form with more than seven sides. This problem is difficult to solve, as the contours of the signs usually overlap with the background or with some external object (for example tree branches or dirt sports).

Once selected the possible sign candidates from its shape, its time to select the main object on the image. The final approach taken is explained on section III, but other possibility was also considered. This possibility consisted on finding the centers of all the contours detected and compute the distance of all the points of the contour to that center point. Then, the maximum value of all the distance was taken and the contour of the image with the largest value of this parameter will be considered as the main object of the image. This approach generally worked well, but there was a problem in some images that the program detected large irregular shapes and set them as the main element of the image. The main characteristic of these elements was its thin elongated shape, that made the distance parameter rise. Therefore, the maximum are approach was used instead, with better results on testing.

Finally the detection stage crops the image on a rectangle to pass it to the classification stage. This is done manually to give some margin to the cut. However, some other techniques where raised. For example, to crop the image on a rotated rectangle with minimum area or to crop the sign following its shape (a triangle for triangular signs or a ellipse to circular signs). However this techniques were rejected for two main reasons. The first one is that the SVM classifier is trained and tested with images of signs with a bit of rotation and from different angle views, and takes an input of a rectangular image to perform its prediction. the second one is that the cameras that are going to capture the images are supposed to be located on cars.

Fig. 7. Two results of the detection stage

Fig. 8. ERROR Image Classification

It is very unlikely that the car will capture a sign in different orientation than the straight one.

On figure 7 it can be seen two examples of the final output this stage returns. On the left side, a sign correctly located can be appreciated and on the right side an image in which the detection result is undesirable.

C. Classification

As mentioned before in the implementation of the classifier, *classification.py* program, there was a previous test done with same images of the created model. The results for the test were that the model classified the testing images correctly. Of course this is just a first preview about correct work of the program. It needs to be tested with the images detected by the detection stage to get the final conclusions.

Once both parts of the program are finished and joined to work properly the program is tested for classification with the selected images from *../data/Test/* folder. The results were that not all of the images were classified correctly. When an image is not classified as one of the defined labels it is classified as label 0 or ERROR which does not display the name on the image. The causes for these errors were mainly four:

- Detected area was not the traffic sign. Therefore the resulted label is ERROR. See Figure 8 right side. Technically this will be an error of detection because it is well classified.
- Detected area was the traffic sign. However, as it has not enough similarities to one of the labels of

Fig. 9. Good Image Classification

the trained model, it is classified as ERROR. See Figure 8 left side.

- Detected area was the traffic sign. However, the predicted label was not ERROR and was one of the other labels. Therefore, the image was wrongly classified.
- Detected area was not the traffic sign. However, the predicted label was not ERROR and was one of the other labels. Therefore the image was wrongly classified.

The last two cases of error were solved by cleaning the data set and setting a maximum number of images of each label. The final model does classify well some of the images and some wrong. Therefore it might be some improvement in the algorithm. Or also as it was previously commented, a different classifier like a CNN could behave better. Figure 9 shows the results for two well classified images. The left one is classified as label *do not turn right* and the right as label *school*.

According to the accuracy of the model obtained in the test process. Using the less possible number of bins at the HOG in which the model works equal to 5 (increases the accuracy by 3% from the previous 9 bins). And restricting the MAX-NUMBER parameter to 80 images per label. The accuracy of the model obtained is 93.52%. It is true that if it is use the entire dataset the accuracy increases up to 99.22%. However, this is not a true value since the are some of the categorized images that have many more samples than others. Therefore if the hole dataset is used the model will not be balanced and there will be more misclassifications.

V. Conclusion

From the educational point of view, the project has been a great opportunity to work with several tools and techniques related with computer vision and artificial intelligence. It has been a useful experience to learn how to work with real life data and to learn how to manage time and resources.

From the project results point of view, the objective of the project has been completed. However, it is true that the obtained results and the results from the tests performed by the students with external data from the database differs considerably. This may have various reasons, but the ones that are more likely are that the detection and classification phases could be improved and that the SVM model is overfitting the data used.

VI. Future improvements

This project was subjected to a strict time constrain and the students schedule did not allow to apply more complex techniques to this solution. The designed program is susceptible to improvements in many phases. Some of them are going to be proposed.

- Change the SVM classifier for a Convolutional Neural Network. This will improve the classification accuracy drastically, as this kind of technology has proved to be better at learning the key issues of the image in order to make more accurate predictions. One possible example of the improvement of this technique is shown at [8], in which the author Alex Staravoitau obtains an accuracy of 99.3% with a dataset containing 43 classes. However, this method will need of higher compute capabilities, that can be provided by the use of GPUs or online tools such as Google Collab.
- Sign detection by top-down sliding windows algorithm (scanning the image with all the possible candidate windows). This approach allows to obatin better candidates than only studying the form factor of the contours obtained.
- Adapt the program to be able to detect signs on video or live footage. This adaptation will allow to test the program on more accurate circumstances, as it can be tested with video provided from a dash-cam of a car.

APPENDIX

A. Job division by students

The division of the work for this project has been done as follows:

- **Jose Pablo Cereceda** has implemented and written all the operations related with image preprocessing and sign detection. He has also implemented the part in which the result of the classification is displayed on the original image.

- **Diego Martin** has implemented and written all the operations related with the image classification model.
- **Together** they have made the design decisions that concerned the global project and have tested the functionality of all the program.

B. User Manual

The implemented program runs on terminal when the command *"python3 detection.py"* is input. The command also accepts two arguments, a *"-h"* to display the help of the command and a directory name, that would be the folder in which the test image are. By default this directory is *"../data/Test"*.

When running the program, the first image of the test folder is displayed as a preview. When pressing any keyboard button, the result of the sign recognition is displayed. If any button but *"q"* is pressed, then the next image on the folder will be displayed. Else, the program terminates.

The signs recognized are saved on the a directory called *"../data/temp"* that should be created before executing the program for the first time. Every execution of the program, this files will be deleted and rewritten, in case there are some new images that are been used to test the program.

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

REFERENCES

[1] D. Gernimo, J. Serrat, A. M. Lpez and R. Baldrich, "Traffic Sign Recognition for Computer Vision Project-Based Learning," in IEEE Transactions on Education, vol. 56, no. 3, pp. 364-371, Aug. 2013.

[2] R. Timofte, KUL Belgium traffic signs and classification benchmark datasets, 2012, [http://homes.esat.kuleuven.be/~rtimofte]

[3] Traffic Sign Recognition - ETH Zurich http://www.vision.ee.ethz.ch/~timofter/traffic_signs/

[4] OpenCV contours features [https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html]

[5] S. Maldonado-Bascn, S. Lafuente-Arroyo, P. Gil-Jimnez, H. Gmez-Moreno, F. Lpez-Ferreras, "Road-sign detection and recognition based on support vector machines", IEEE Trans. Intell. Transport. Syst., vol. 8, no. 2, pp. 264-278, Jun. 2007.

[6] R. Timofte, KUL Belgium traffic signs and classification benchmark datasets, 2012, [http://homes.esat.kuleuven.be/~rtimofte]

[7] J. Khan, S. Bhuiyan, R. Adhami, "Image segmentation and shape analysis for road-sign detection", IEEE Trans. Intell. Transport. Syst., vol. 12, no. 1, pp. 83-96, Mar. 2011.
[8] Traffic signs classification with a convolutional network, by Alex Staravoitau [https://navoshta.com/traffic-signs-classification/]
[9] OpenCV (Open Source Computer Vision Library) [https://opencv.org/]
[10] NumPy, the fundamental package for scientific computing with Python [http://www.numpy.org/]
[11] Python 3.7.1 documentation [https://docs.python.org/3/]
[12] StackOverFlow [https://stackoverflow.com/]