Embedded module for detection and recognition of miniature traffic signs using neural networks

Mihai-Bogdan Bindila
Computer Science Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
b.mihaibogdan@yahoo.com

I. Introduction

Traffic signs recognition is one of the hottest problems in the field of autonomous driving cars. A lot of solutions have been tried starting with detection based on hand-crafted features, color segmentation and finishing with convolutional neural networks that can detect and classify road signs. These modules have been integrated into advanced driver-assistance systems since 2009, and the aim is to use them to completely recognize road signs in various conditions. A traffic signs recognition module needs to be able to run on an embedded system, so at the same time a high accuracy of detection and power efficiency must be maintained. The biggest problem of these modules is represented by the detection method, which implies color segmentation. A data-driven approach is more robust, considering the improvement of deep learning methods. That being said, this paper is focusing on presenting an environment-invariant method of detecting and classifying road signs based on convolutional neural networks that can run on an embedded system like Raspberry Pi 4. This module will be integrated in a 1/10 scale autonomous car that will recognize small traffic signs.

II. Related Work

Various papers were studied before and were selected to be discussed just those where convolutional neural networks are used. Related works will be presented evolutionarily, so the ideas will be more complex and brilliant. In [1] is shown a method that consists of two steps. The first one is represented by ROIs extraction based on color segmentation of an RGB normalized image. These shapes are filtered based on hard-coded features like aspect-ratio, size and number of edges of the shape, that are computed for a specific resolution. For classification a CNN based on LeNet5's architecture has been used. The accuracy reached is 96.2%, but techniques like image augmentation and dropouts are not used. This module runs on Nvidia Jetson TX1 with a poor performance of 1.6 fps on images of 1360x800 pixels resolution.

The previous method had the weakness of using hard-coded values for sign extraction. Also on the GTSB data set, the network presented in [2] is trained, which uses a similar architecture, but with an interesting change. Data from the first convolutional layer is sent to the first fully connected layer, so more features are taken into consideration. An improvement from the previous classification method is represented by data augmentation. The images from classes with less than 1000 images are augmented and the dropout technique is used to avoid overfitting. The accuracy reached is 97.1%. Despite the changes, some issues are not considered, like data balancing across the classes and data augmentation that is performed only by geometrical operations.

An end-to-end solution has been presented in [3]. This solution consists of three steps. First, the image is converted to HSV color space and then the noise is removed. Second, the Generalized Hough Transform is used for sign detection. The third step is represented by image classification, where just 16 classes of GTSRB have been used. For training, batches of 50 images are fed into the network and an accuracy of 99.94% is reached. The weak point of this method is depicted by the lack of image augmentation, but the GHT method for detection seems to be a method that doesn't require specific features to work with.

Pierre Sermanet and Yann LeCun have proved that the color component in images doesn't bring a higher accuracy of the trained model [4]. A precision of 99.17% was reached in 2011 on German Traffic Sign Recognition Benchmark using images in YUV color space and taking in consideration for prediction just the Y channel. In this paper appears the technique of image normalization of grayscale images that increases the accuracy.

The scientific community's perception of object detection has been changed by 2016 when You Only Look Once [5] was published. This is one of the most impressive ways of object detection and classification. With one network the localization and prediction are made. This network can be trained to detect any object, so road signs too. The accuracy is 57.9% on the Pascal VOC 2012 data set that includes 20 classes of images and is the first object detection approach that can be used in real-time, unlike R-CNN or Deformable Parts Models. The algorithm is outlined by the division of the image in an SxS grid cell and each of those can predict multiple bounding boxes. A prediction implies five numbers: x, y, h, w of the box and the attached probability.

In 2018 an improved version of the network was published: YOLOv3. Anchor boxes along with logistic regression are used to predict bounding boxes. The last layer of the network [6] that was a softmax was replaced and the problem is now seen as a multiple classification problem. In that manner errors between close classes are excluded. The biggest problem of YOLO detecting the precise location of the object, so now three sizes of bounding boxes are used to locate even small objects. Even if the architecture has been changed to Darknet-53 which implies an architecture with 53 convolutional layers, the performance is still excellent compared to ResNet-101 or SSD.

III. Proposed Solution

A. *Solution description*

To accomplish the proposed requirements, the system must be composed of two modules: the first one will extract the proposed ROIs and the second one is represented by a model that will classify the images. In order to extract regions of interest two separate pipelines will be used, one based on poles detection and a back-up method based on color segmentation. The second pipeline is employed when the pole is not visible because of a dark background or the sign doesn't have one. Hence the algorithm will be robust enough to detect signs in various lighting conditions. For the first pipeline, the color component isn't evaluated, thus this method will work even for a camera with unreliable color representation or for scenarios when the sign is in a shadow cone or the light hits the sign directly.

B. *Description of the implementation flow*

The target system is represented by a Raspberry Pi, therefor the network will be developed on a desktop and then deployed on the embedded system, together with the module that will extract the proposed regions. "Fig. 1" represents the implementation flow of the system and illustrates the steps of model creation. To obtain a high performance the main module is implemented in C++. The convolutional neural network is developed in Python, using Tensorflow and Keras, and then converted to JSON format using an open-source library named Frugally-Deep. This conversion ensures that the model can be deployed in a C++ code. In the following paragraphs the stages of the module's creation will be presented.

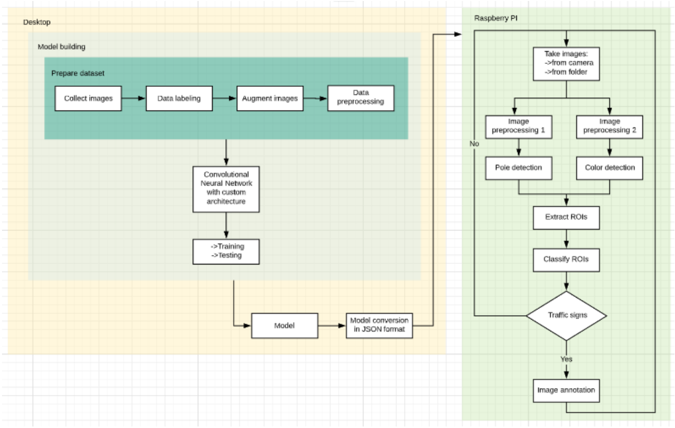

Fig. 1. System architecture with implementation flow

C. *Datasets preparation*

The module proposes to detect 4 types of road signs: stop, priority, parking, and pedestrian, so the used method employs 5 classes, including a residual class named "other" where are included samples of extracted images that are false calls.

To create a model that can generalize extremely well, a set of diverse images is needed. Images captured in real-time with a Raspberry Pi Camera V2.1 in different scenarios were used. The initial set contained over 33000 images that were filtered in order to use samples that distinguish one from another. To create a rich dataset, images from the German Traffic Sign Recognition Benchmark were added. After a fine filtering, around 1000 samples remains in the dataset.

The dataset is augmented using techniques like rotations, zoom-in, zoom-out, illumination alteration, and shifting of color channels. After augmentation every class has 600 images for training and 200 for testing. These images are resized to 64x64, converted to grayscale, and then a contrast limited adaptive histogram equalization operation was applied to each of them. As seen in [4], using grayscale images with a histogram equalization applied increases the accuracy of the network. For the histogram equalization different sizes for patched were used and the best results are achieved when the operation is applied with 8x8 patches. The results can be visualized in "Fig. 2" where the sizes used are 2, 4, 8, 16 and 32 from left to right.

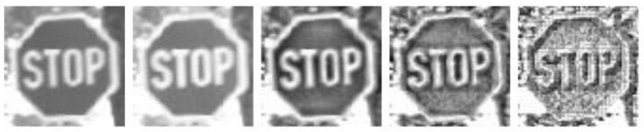

Fig 2. Different patch sizes for CLAHE

D. *Network architecture*

The network has a relatively simple architecture that can easily reach a high accuracy and that can eliminate the overfitting phenomenon. Three types of layers used are: convolution, max pooling and fully connected. As a regularization technique, a dropout of 0.5 is utilized. The last layer is represented by a fully connected with a Sigmoid as activation function. The loss function complements the last layer, thus using a binary cross-entropy loss, the domain problem is transformed into a multiple label one. After a classification, the greatest probability is taken into consideration. In this manner just a classification with a probability greater than 99% is valid for the system. Using this kind of loss makes the process of every validation more robust, unlike in the case of a softmax activation function where a distribution of probabilities is used.

E. *Application pipeline and implemented algorithms*

Images are taken from a folder or a camera and then two preprocessing pipelines are applied to the images.

In the first pipeline "Fig. 3", a contrast and brightness enhancement using a gamma correction is applied to highlight all the details in the dark spots. The image is converted to grayscale format and then a smoothing is done with a Gaussian Blur which uses a 5x5 kernel. To extract all the edges a Canny filter is then applied. For this filter, an adaptive parameter is used as a threshold [7]. Because the edges that have an inclination angle aren't contiguous, a dilation with a kernel of 3x3 is applied to fill the gaps.

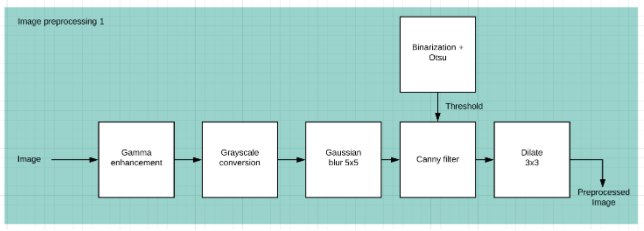

Fig 3. First image preprocessing pipeline

The preprocessed image is then used to extract the best lines that fit the pillars. To get these lines Hough Transform

Probabilistic function is used. The segments are filtered based on multiples criteria. First, just those that form an angle with Ox axis between 75 and 105 degrees are used, considering that a pole is detected as being tilted when the camera is getting closer to the sign “Fig. 4”.

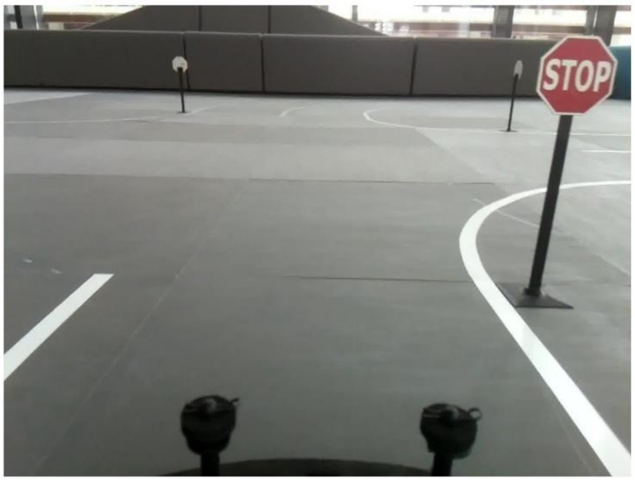

Fig 4. Example of a stop sign titled with 10 degrees

The remaining lines are passed into a second filtering that has the purpose to keep from every set of lines that are neighbors, just the one with the smallest y coordinate. The probability that this line extends to the sign's base is the largest. To implement this algorithm a copy of the preprocessed image is used. In this copy every segment is colored with a different color and a matrix of 5x5 is centered sequentially in every upper end of each segment to determine the corresponding colors of the lines that will be compared with the current segment. The coloring can be observed in “Fig. 5”.

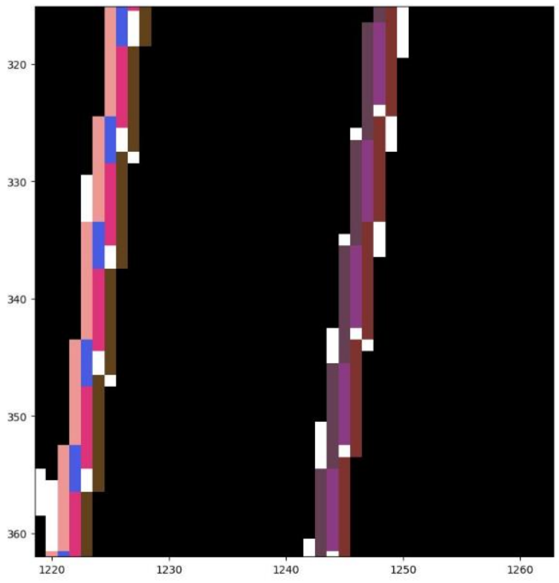

Fig 5. Coloring the adjacent segments

The remaining segments are sorted in decreasing order based on Ox coordinate. Each line is grouped with the closest one that is positioned on the same level. The lines that are isolated are rejected. This algorithm’s pipeline is represented in “Fig. 6”. For every pair of segments the distance between the top ends is computed. Based on this distance and knowing the sign's size and the pole's width, the bounding box is determined. A translation is applied to this box based on the pillar's inclination.

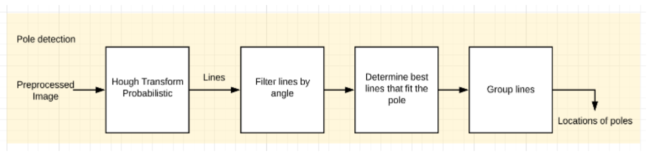

Fig 6. The pipeline used for pole extraction

The second pipeline "Fig. 7" is used to determine the regions of interest based on color segmentation, therefor on the image is applied a gamma correction transformation to enhance the colors, then the color space is converted from BGR to HSV and color masks are applied for yellow, blue and red colors. After applying the masks, the bounding boxes of the connected components are extracted.

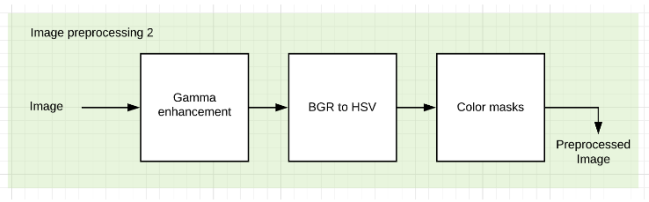

Fig 7. Second image preprocessing pipeline

All the bounding boxes that have resulted from the previous algorithms are filtered based on area and aspect ratio. Knowing the coordinates of the boxes, ROIs are extracted from the original image and then preprocessed. In “Fig. 8” and example of ROI detection based on the pillar is presented. The image presents the result of the first pipeline of preprocessing.

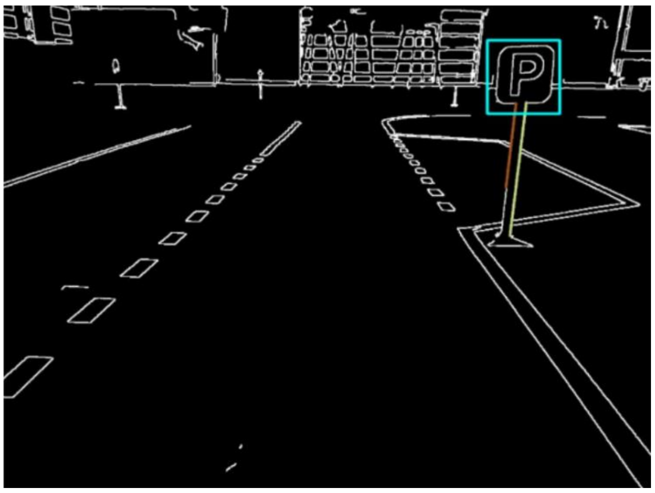

Fig 8. Region of interest detected based on the sign’s pole

The preprocessing consists of a resizing to 64x64, conversion to grayscale and of a histogram equalization application in order to have the same transformations on the image that will be classified as the images from the dataset. A classification is considered valid if the biggest probability returned by the model is greater than 99% and the predicted class is different from “Other”. Finally the image is annotated by representing the bounding box and the predicted class.

IV. Results

A. Statistical results

The convolutional neural network was trained using a batch size of 16 images and the Adam optimizer. To find the optimal number of epochs, different learning rates were used to observe the monotony of the loss function. In this manner the network was trained until the loss function began to grow, so the overfitting didn't occur. The network reached an accuracy of 99.33% for the training set and 98.98% for the testing dataset. These results will give robustness to the module.

After the conversion of the model in a JSON format, the code that was written in Python was then implemented in C++ for a performance boost. The number of frames per second increased by 270%. The algorithm was tested on a Windows machine with an Intel i7 7700hq processor and 8gb of ram. On average, an image is scanned and the regions of interest are classified in 0.0043 seconds, so approximatively 23 images are processed per second. These results are generated for the worst-case scenario, so this translated to a minimum of 4 frames per second in the worst case for a Raspberry Pi 4. If the algorithm process just images in which the poles of the signs are visible and distinguishable from the background, the color segmentation algorithm can be turned off, so there will be a significantly less number of classified regions. This will increase the performance up to 39 frames per second on the same Windows machine in the worst-case scenario, leading to a minimum of 6 frames per second on a Raspberry Pi 4.

B. Testing in different scenarios

To observe how the module behaves it was tested with different images and videos with different lighting conditions. To test the detection algorithms of the regions of interest were used images in which the pole is not visible, pictures in which the color is not represented correctly and frames in which the sign is partially occluded.

In figure 9 is presented an image with an sign whose pillar isn't visible because of the background. This is an edge case in which the region of interest can't be extracted based on the pole. The backup solution based on color segmentation detects the blue color of the sign, so in the end, it is classified as a pedestrian road sign.

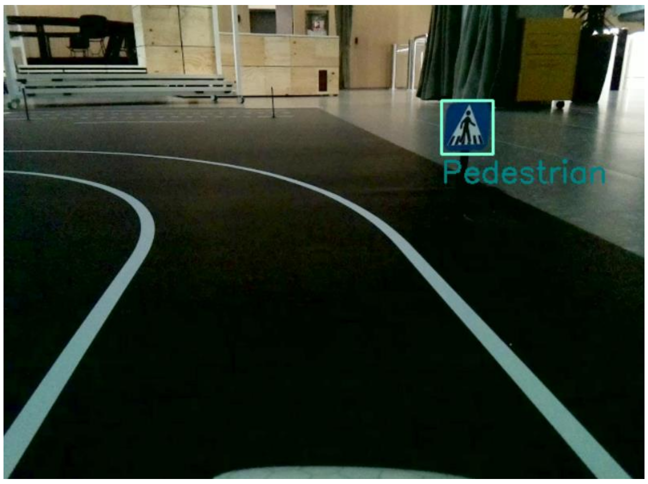

Fig 9. Image with an indistinguishable pole because of the background

In the figure below an extreme case is presented. In this special case, a beam of light hits the camera's lens and the colors in the image are bearly represented, so the parking sign has a grayish color. In this situation, the sign is recognized based on the pillar's detection. These two scenarios expose the robustness of the module. It can recognize traffic signs even if the color isn't represented correctly or if the pole isn't visible.

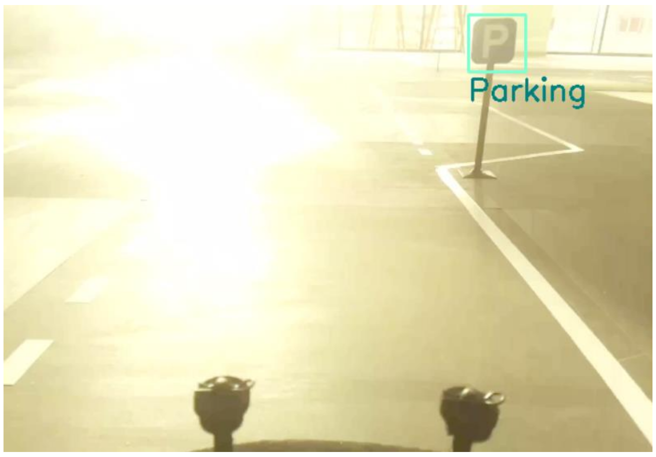

Fig 10. An extreme case in which the light hits the camera's lens

The algorithm can perform the detection even on partially occluded signs, as can be seen in figure 11. This can be very effective in real case scenarios when a large majority of signs are semi-covered by threes or large cars.

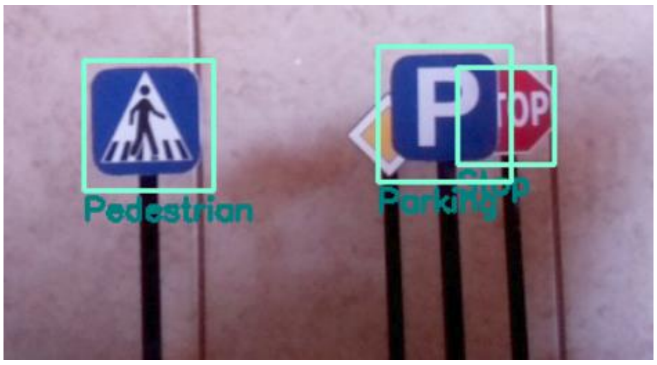

Fig 11. Detection of an partially occluded stop sign

V. Conclusion

This paper has described a robust solution for the detection and recognition of miniature traffic signs. Due to the C++ implementation, the module is optimized to run on an embedded system with limited capabilities and can perform the detection of areas of interest using two distinctive methods in parallel. This approach ensures the system that even in edge-case scenarios when the color reproduction is very weak, or when the pole isn't distinguishable from the background the sign will be detected. The idea of detecting a sign based on the pillar is novel and brings to the field a new approach of signs recognition. Due to the configurable parameters of the algorithm, this module can be scaled to detect and recognize real-size road signs. The recognition is empowered by a convolutional neural network that can generalize the appearance of a sign, so this aspect gives the module a great flexibility in various scenes.

References

[1] Y. Han and E. Oruklu, "Traffic sign recognition based on the NVIDIA Jetson TX1 embedded system using convolutional neural networks," *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Boston, MA, 2017, pp. 184-187.

[2] D. Yasmina, R. Karima and A. Ouahiba, "Traffic signs recognition with deep learning," *2018 International Conference on Applied Smart Systems (ICASS)*, Medea, Algeria, 2018, pp. 1-5.

[3] Shustanov, Alexander & Yakimov, Pavel. (2017). CNN Design for Real-Time Traffic Sign Recognition. Procedia Engineering. 201. 718-725. 10.1016/j.proeng.2017.09.594.

[4] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale Convolutional Networks," *The 2011 International Joint Conference on Neural Networks*, San Jose, CA, 2011, pp. 2809-2813.

[5] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 779-788.

[6] Redmon, Joseph & Farhadi, Ali. (2018). YOLOv3: An Incremental Improvement.

[7] Fang, Mei & Yue, GuangXue & Yu, QingCang. (2009). The Study on An Application of Otsu Method in Canny Operator.