IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Traffic Sign Detection using YOLOv4

Mr. Rishabh Singh, Mr. Momin Danish, Mr. Vipul Purohit, Prof. Ashraf Siddiqui

UG Student, UG Student, UG Student, Assistant Prof.

Dept. of Computer Engineering

Universal College of Engineering, Mumbai, India

Abstract: Traffic sign detection is an important aspect of autonomous vehicles, this helps in avoiding accidents. In this project, a deep learning model is implemented with YOLO's (You Look Only Once) latest version YOLOv4. YOLO is one of the fastest object detection algorithms for real-time detection. In YOLOV4, CSPDarknet54 is used which acts as the backbone, this promotes the learning capability of CNN. We prepared two models, one that was trained on the GTSDB dataset and another on an independently made dataset consisting of 4 classes. We achieved a 92% mAP on the GTSDB dataset and 94.40% mAP on the CSST dataset.

Key Words: Traffic Sign Detection, YOLOv4, GTSDB, CNN, Machine learning, Darknet

1. INTRODUCTION

In recent years, with the development of autonomous vehicles and the increasing demand for self-driving cars, the need for traffic sign detection has come into the picture. Traffic sign detection (TSD) will help drivers be more aware of their surroundings and help them recognize signs in the distance, during bad weather, or during nighttime when visibility is less. And a good detection system is needed to make all of it possible. TSD is valuable so as to maintain road safety, prevent road accidents and traffic issues. But TSD is still a challenging real-world problem because of the challenges that arise in real life-like image quality, lighting, etc.

Thus to deal with all these challenges, many new machine learning methods and algorithms have been introduced. In the past, TSD mainly relied on traditional object detection algorithms. The pipeline of TSD generally used hand-crafted features to extract region proposals and then combines classifiers to filter out the negatives [6]. In recent times, deep learning methods are on the rise and this has contributed to the significant advance for target detection and identification tasks. Most of the studies regarding image recognition and object detection revolve around deep convolutional neural networks (CNNs) so as to increase precision, speed, and accuracy. CNN possesses the ability to learn features from large data without preprocessing, which avoids the process of designing hand-crafted features and absorbs generalized features [6]. CNN has already cropped up as a classifier in machine learning and also been implemented in traffic sign classification.

Among the recent advancements in object detection, algorithms like SSD, Fast R-CNN, Faster R-CNN, R-FCN, and YOLO have used CNN. We used "You Look Only Once" (YOLO) architecture which is a single-shot detection network and has low propagation latency and excellent detection performance. Many accurate modern neural networks do not operate in real-time and require a large number of GPUs for training [12]. YOLO addresses such problems by creating a CNN that operates in real-time on a conventional GPU, and for which training requires only one conventional GPU [12].

Various datasets like GTSDB [13], LISA Traffic sign dataset(LISATSD), Swedish Traffic Signs Dataset (STSD), and Chinese Traffic Sign Dataset (CTSD) are available to evaluate the relative methods for TSD. GTSDB is the most popular one for detecting traffic signs. We prepared two models, one that was trained on GTSDB and the other was trained on an independently made dataset found on Kaggle made by Andrew Mvd which consists of 4 classes (crosswalk, stop, traffic light, speed limit), we'll refer to it as "CSST" (based on the initials of the signs in it).

The paper is organized as follows. Section II talks about the related work, in Section III we discuss the dataset used in this research, Section IV gives information about the characteristics of the YOLO algorithm. In Section V Detailed analysis of experimental results is given while Section VI gives conclusion remarks.

2. RELATED WORK

In traffic sign recognition systems there are two main components: classification and recognition. The classification of an image means determining the type of traffic sign after the traffic sign has been detected and recognition means locating the traffic sign in a sequence of images. TSD methods can be divided into two main categories, one is based on traditional object detection algorithms where the properties of traffic signs such as color-based and shape-based are combined. The other is the learning-based methods, such as machine learning and deep learning techniques, which possess the ability to self-learn various features.

Traditional methods include low-level feature extraction techniques to recognize or detect signs and require primary colors and shape features. But these solutions were limited to a small number of different traffic sign categories. The algorithms used in these methods for determining the location of traffic sign instances in images were time-consuming, or they concentrated only on the classification of pre-cropped traffic sign instances. Later machine learning-based detection techniques came into the picture.

They focused on the extraction of hand-crafted features for both the detection of a traffic sign and the classification into one of several classes. These extracted features were then used to train classifiers such as support vector machines (SVMs), Bayesian classifiers, or logistic regression models. Wang et al. [14] made use of the histogram of oriented gradient (HOG) feature with an SVM classifier to detect traffic signs and obtained great grades in the German Traffic Sign Detection Benchmark competition organized by the International Joint Conference on Neural Networks (IJCNN) in 2013. Although hand-crafted features have obtained a higher precision for traffic signs, it is observed that traditional detection techniques have higher pertinence, however, they lack the robustness for the overall system.

After 2013, researchers mostly used CNNs for TSD and TSR. Modern detectors are made up of two parts, a backbone and a head, backbone is pre-trained on ImageNet, and the head is used to predict classes and bounding boxes of objects [12]. For the detectors which run on GPU platforms, their backbone could be VGG, ResNet, ResNeXt, or DenseNet. For the detectors which run on the CPU platform, their backbone could be SqueezeNet, MobileNet, or ShuffleNet. The head part is differentiated into two kinds, a one-stage object detector, and a two-stage object detector. R-CNN series is the most typical two-stage object detector, it includes fast R-CNN, Faster R-CNN, R-FCN, and Libra R-CNN. As for one-stage object detectors, the most representative models are YOLO [15], SSD, and RetinaNet. The two-step strategies prove better in recognition accuracy and localization accuracy. But the efficiency of the computation is poor, and the process demands large amounts of time and resources. The single-stage methods are much quicker because of the unified network structures, although the process accuracy falls down a little. In addition, the amount of dataset is a key factor in deep learning-based methods.

YOLO is a straightforward and effective model for real-time object detection. Firstly, YOLO is a simple convolutional neural network that can simultaneously predict multiple bounding boxes and class probabilities. Initially, it is trained based on full images and the performance is optimized [15]. Secondly, YOLO is extremely fast which can achieve more than twice the mean average precision (mAP) of other real-time systems [16]. However, YOLO has a poor performance in accuracy when compared to Faster R-CNN. In 2020 further YOLO versions were released. And YOLOv4 is considered to be the fastest and most accurate real-time detection model. It uses the CSPDarknet54 as its backbone and obtained 43.5% AP on the MS COCO dataset at a real-time speed of ~65 FPS on Tesla V100. The AP and FPS of YOLOv4 improved effectively compared to the previous version YOLOv3.

3. DATASET

Various datasets like German Traffic Sign Detection Benchmark (GTSDB), LISA Traffic sign dataset(LISATSD), Swedish Traffic Signs Dataset (STSD), and Chinese Traffic Sign Dataset (CTSD) are available to evaluate the relative methods for TSD. We focused our experimentation on the GTSDB dataset and the independently made dataset CSST. First, we'll discuss the GTSDB dataset.

There are various reasons for choosing this dataset over the others, also it is highly accepted and is widely used for comparing traffic sign detection approaches in the literature. Furthermore, its authors and the organization behind them organize a public challenge, where scientists and researchers from various fields contribute their results and test the GSTDB dataset. The GTSDB dataset contains natural traffic images shot on various types of roads (highway, rural, urban) during the daytime, around dusk, and various weather conditions are featured. The dataset contains 900 full images containing 1206 traffic signs, it is further split into a training set of 600 images (846 traffic signs) and a testing set with 300 images (360 traffic signs). Each of these images contains 0,1 or multiple traffic signs which normally suffer from differences in orientation, light conditions, or occlusions. The dataset has four main classes/categories namely mandatory, prohibitory, danger, and other, however, signs labeled as other fall in minority and are not relevant to the challenge itself, and hence are discarded. The training set contains 396 prohibitory, 114 mandatory, and 156 danger traffic sign samples while the testing set comprises 161 prohibitory, 49 mandatory, and 63 danger traffic sign images. The CSST dataset contains 877 images of 4 distinct signs namely crosswalk, stop, traffic light, speed limit. The reason for choosing this dataset was to see how well our model performs while identifying individual signs.

4. PROPOSED METHOD

4.1 INTRO TO YOLO

The YOLO models are end-to-end deep learning models and are preferred due to their detection speed and accuracy. A YOLO network has a structure similar to that of a normal CNN, consisting of several convolutional and max-pooling layers, ending with two fully connected layers. YOLO follows a very unique approach where it employs a single neural network to look at the entire image just once, because of this it has various advantages over classifier-based systems. To predict each bounding box, it uses features from the entire image.

YOLO model divides the input image into S × S grids as shown in Fig 1. Each grid cell predicts k bounding boxes and confidence values of bounding boxes, as well as C conditional class probabilities [14]. The confidence score means how confident the model is about the object in the box and also the accuracy of the box it's predicting. Each bounding box makes 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center offset of the box w.r.t boundaries of the grid cell. The w and h are width and height, predicted relative to the whole image. Confidence (cfd) prediction is defined as $P_r(\text{Object}) \times IOU^{truth}_{pred}$. When a grid cell contains a portion of a ground truth box the value of Pr (Object) is 1 else it is 0. IOU stands for the intersection over the union between the predicted bounding box and the ground truth box. With the help of these predictions, we derive the class-specific confidence score of individual bounding boxes and finally select the bounding boxes with high confidence scores in each grid cell in order to make global predictions of a traffic sign in the image.

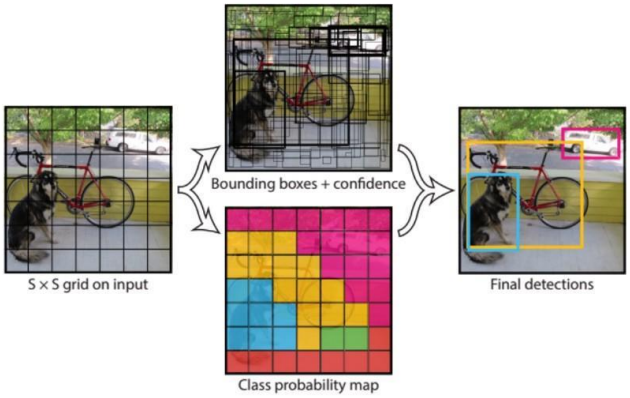


Fig 1: How YOLO detects

4.2 YOLOv4

Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao developed YOLOv4 and released it in April 2020 and is claimed to be one of the state-of-the-art real-time object detectors at the time. In comparison to YOLOv3, YOLOv4 is 12% faster and 10% more accurate. The new architecture of YOLOV4 is built with CSPDarknet54 as a backbone, which increases the learning capability of CNN. The YOLOv4 model can be trained on a single GPU.

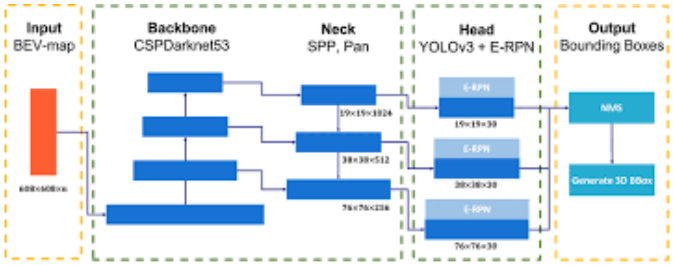


Fig 2: YOLOv4 architecture

Let's elaborate on the details of YOLOv4 [12].
Yolov4 consists of:

- Backbone: CSPDarknet53(feature-extraction architecture)
- Neck: SPP, PAN
- Head: YOLOv3

To make YOLOv4 a state-of-the-art model and to be at the top of deep learning, the authors developed techniques like bag-of-freebies and bag-of-specials to improve the accuracy of the model while training and in post-processing.

Bag of Freebies (BoF): Bag of Freebies are set of techniques used during training that help increase the model's accuracy without increasing inference time.

- BoF for backbone: CutMix & Mosaic data augmentation, Class label smoothing, DropBlock regularization
- BoF for detector: CIoU-loss, CmBN - Cross mini-Batch Normalization, DropBlock regularization, Mosaic data augmentation, SAT - Self-Adversarial Training, Eliminate grid sensitivity, Multiple anchors for a single ground truth, Cosine annealing scheduler, Optimal hyperparameters, Random training shapes.

Bag of Specials (BoS): Bag of Specials changes the architecture of the network and increases inference time a little but helps in improving the accuracy of the model.

- BoS for backbone: Mish activation, CSP - Cross-stage partial connections, MiWRC - Multi-input weighted residual connections
- BoS for detector: Mish activation, Spatial Pyramid Pooling SPP-block, Self Attention Module SAM-block, PAN path-aggregation block, DIoU-NMS.

4.3 Dataset to YOLO format

To feed the dataset to YOLOv4, they must be in a format that YOLOv4 accepts. Images in GTSDB are in ppm format. To be able to use GTSDB for YOLOv4 we have to convert the ppm format images to jpg files. Our CSST dataset is already in jpg format. Next we have to label the images and create .txt files for each image, labelling is done as : <object-class> <x_center> <y_center> <width> <height>. Next we create 4 files: obj.names, obj.data, train.txt, test.txt. The obj.names file contains all the classes of the dataset, train.txt and test.txt files contain paths to your training and testing images. We split the dataset into training and test sets and train our model on the training set and validate it on the test set. If the loss on our training data is high, it means our model is underfitting and it needs to be trained longer. If the loss on our training data is low and high on our test data, it means our model is overfitting and more data needs to be added.

5. RESULTS AND DISCUSSION

5.1 Training Discussion

We trained our model on Google's Colab Notebook. Colab permits users to write and execute discretionary python code through the browser and is useful for machine learning, information analysis, and education. More technically, Colab is just a hosted Jupyter notebook service that needs no setup to use, whereas providing free access to computing resources as well as GPUs. Colab offers 12GB of GPU. The kinds of GPUs that are available for users to use in Colab vary over time. This is often necessary for Colab to be able to give access to those resources for zero cost. The GPUs provided by Colab are Nvidia K80s, T4s, P4s, and P100s. The only way you can choose GPUs of your choice is if you get Colab Pro.

Both our datasets have 4 classes each. Before beginning our training we need to configure our files and set some parameters as described in Table 1. In our .cfg file we set batch = 64 and subdivisions = 16 for ultimate results, width = 416, height = 416. The rest of the changes like max_batches, steps, filters should be done based on the number of classes you are training your detector on. We set max_batches = 8000 (# of classes * 2000) max_batches should not be less than 6000 though; steps = (80% of max_batches), (90% of max_batches); filters = 27 ((# of classes +5)*3).

Table -1: Parameters for the training

Parameter	Value
Network size (input width,height)	416 *416
Batch	64
Subdivisions	16
Momentum	0.949
Decay	0.0005
Learning rate	0.001
Saturation	1.5
Exposure	1.5

5.2 Experimental Results of YOLOv4

Tables 2 & 3 show the mAP results calculated of our datasets every 1000 iterations. We also depict the TP and FP. TP represents True positives which means the true prediction of the positive class, FP represents False-positive which means the false prediction of the positive class. The table also shows precision, recall, F1-score, Average IoU. IoU represents the relationship between the result of the detection, the reality of the ground truth, and its relation. It measures the projection ratio.

$$IoU = \frac{Area_{pred} \cap Area_{gt}}{Area_{pred} \cup Area_{gt}}$$

The precision value represents the number of positive class predictions that actually belong to the positive class.

$$Precision\ (P) = \frac{True\ Positive\ (TP)}{True\ Positive (TP) + False\ Positive (FP)}$$

Recall value represents the number of positive class predictions made out of all positive examples in the dataset.

$$Recall\ (R) = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)}$$

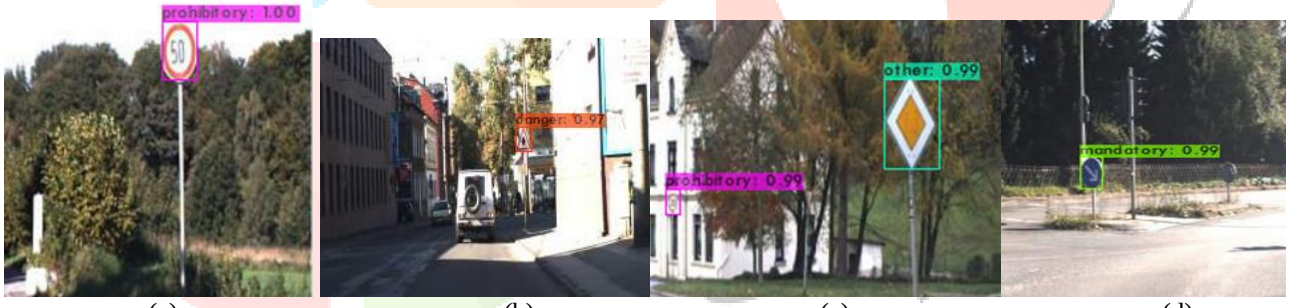
F1-Score value is a single score that balances both the concerns of precision and recall in one number.

GTSDB Dataset:

In Table 2 we can see mAp's of all iterations and of each class (Prohibitory, Danger, Mandatory, Other). The highest mAp we got was 91.96% after 3000 iterations, TP = 219, FP = 32 After that the model mAp didn't really go up and stayed the same and probably started overfitting. Fig 3 shows the image detection results.

Table -2: GTSDB mAP results during training
(P - Prohibitory, D - Danger, M - Mandatory, O - Other)

Iterations	Average Precision (%) (TP, FP)				Precision	Recall	F1-Score	Avg IoU (%)	mAp(%) (TP, FP)
	P	D	M	O					
1000	94.26 (101, 22)	87.39 (32, 15)	63.57 (24, 16)	64.76 (40, 17)	0.74	0.82	0.78	54.95	77.49 (197, 70)
2000	95.26 (104, 9)	92.84 (33, 5)	81.89 (30, 9)	87.89 (46,11)	0.86	0.88	0.87	71.79	89.48 (213, 34)
3000	98.05 (106, 12)	96.57 (35, 6)	83.25 (31, 6)	89.95 (47, 8)	0.87	0.91	0.89	74.95	91.96 (219, 32)
4000	97.05 (106, 5)	97.22 (35,3)	83.32 (31, 6)	86.70 (46, 3)	0.93	0.90	0.92	78.79	91.07 (218, 17)
5000	97.03 (105, 6)	96.77 (35, 2)	84.62 (30, 5)	83.64 (46, 1)	0.94	0.90	0.92	80.4	83.64 (216, 14)
6000	97.80 (105, 6)	97.07 (35, 4)	77.31 (29, 8)	86.44 (48, 3)	0.91	0.90	0.91	77.13	89.65 (217, 21)
7000	97.81 (106, 6)	96.74 (34, 4)	83.10 (29, 5)	86.16 (47, 4)	0.92	0.90	0.91	80.67	90.95 (216, 19)
8000	98.50 (107, 6)	96.16 (34, 4)	83.07 (29, 6)	87.18 (47, 4)	0.92	0.90	0.91	80.39	91.23 (217, 20)



(a) (b) (c) (d)

Fig -3: GTSDB detection results showing
(a) danger sign, (b) prohibitory sign, (c) prohibitory & other sign, (d) mandatory sign.

CSST Dataset:

In Table 3 we can see mAp's of all iterations and of each class (Crosswalk, Speed limit, Traffic Light, Stop). Highest mAp we got was 94.78% after 8000 iterations. But we can see that we got mAp of 94.40% around 2000 iterations. After that, it mAp didn't really change which shows that the model definitely overfitted after 4000 or 5000 iterations, so we are free to choose any weight from 2000 to 4000/5000. Our TP & FP values are 242 and 14 respectively. Fig 4 shows the CSST image detection results.

We also tested our model on videos and got a good response with our model being able to detect the signs and predict their class. We did the testing on a 12GB GPU powered by Colab and also using Tensorflow on our 8GB Nvidia GeForce GTX 1050ti GPU. On Colab we got fps between 20-40 and on our local system we got around 11-12 fps.

Table 3: CSST mAP results during training
(C - Crosswalk, SL - Speed Limit, S - Stop, TL - Traffic Light)

Iterations	Average Precision (%) TP, FP				Precision	Recall	F1-Score	Avg IoU (%)	mAp (%) (TP, FP)
	C	SL	S	TL					
1000	86.90 (37,21)	98.31 (151,24)	86.81 (18,9)	75.08 (32,16)	0.77	0.93	0.85	59.96	86.77 (238,70)
2000	96.96 (38,4)	99.87 (151,4)	100.00 (18,2)	80.76 (35,4)	0.95	0.95	0.95	80.87	94.40 (242,14)
3000	96.87 (38,2)	99.94 (152,3)	100.00 (18,3)	79.96 (33,3)	0.96	0.95	0.95	82.33	94.19 (241,11)
4000	96.67 (38,5)	99.97 (152,3)	100.00 (18,2)	78.75 (35,5)	0.94	0.95	0.95	82.15	93.85 (243,15)
5000	96.85 (38,4)	99.33 (151,4)	100.00 (18,1)	78.51 (36,3)	0.95	0.95	0.95	82.83	93.67 (243,12)
6000	97.08 (39,5)	99.97 (152,2)	100.00 (18,1)	80.51 (36,3)	0.96	0.96	0.96	84.83	94.39 (245,11)
7000	97.14 (39,5)	99.32 (151,3)	100.00 (18,1)	82.55 (35,3)	0.95	0.95	0.95	84.79	94.76 (243,12)
8000	97.03 ((39,5)	99.32 (151,5)	100.00 (18,2)	82.77 (36,2)	0.95	0.96	0.95	84.98	94.78 (244,14)

(a) (b) ©

Fig 4-: CSST detection results showing
(a) Stop & Crosswalk sign, (b) Speed Limit, (c) Traffic Light

6. CONCLUSIONS

In this paper, we proposed a real-time YOLOv4 Traffic Sign Detection model on the GTSDB dataset and an independently made dataset that we referred to as CSST in this paper. We used the end-to-end deep learning model to achieve fast detection and high accuracy. We tested both the dataset and found that our model performed better with the CSST dataset. It can be because of the fact that CSST (877) has more images per sign than the GTSDB (900) dataset. The 877 images of CSST contain just 4 signs and thus there was enough data for the model to train on each image as compared to GTSDB which for 4 classes has 43 signs of images. But the results of both the training datasets are impressive and promising. Our model performed better in real-time video detection as well. However, our model is still not perfect or ideal for detections as it is still prone to the missed detection of some vague targets.

Our work can be further improved by using a dataset that contains more signs and has a good number of images for each sign. We can also fine-tune the model and thus aim to achieve even better detection accuracy. With technology and research getting better and achieving new heights every day, we can also hope for some new detection model or technique to emerge as well.

REFERENCES

1. Qin Z, Yan WQ, "Traffic-Sign Recognition Using Deep Learning. Geometry and Vision," 2021 Mar 18;1386:13–25. doi: 10.1007/978-3-030-72073-5_2. PMCID: PMC7971780.
2. A. Avramović, D. Sluga, D. Tabernik, D. Skočaj, V. Stojnić and N. Ilc, "Neural-Network-Based Traffic Sign Detection and Recognition in High-Definition Images Using Region Focusing and Parallelization," in IEEE Access, vol. 8, pp. 189855-189868, 2020, doi: 10.1109/ACCESS.2020.3031191.
3. Baojun Zhang, Guili Wang, Huilan Wang, Chenchen Xu, Yu Li, Lin Xu, "Detecting Small Chinese Traffic Signs via Improved YOLOv3 Method", Mathematical Problems in Engineering, vol. 2021, Article ID 8826593, 10 pages, 2021. https://doi.org/10.1155/2021/8826593
4. Tai, S.-K.; Dewi, C.; Chen, R.-C.; Liu, Y.-T.; Jiang, X.; Yu, H. "Deep Learning for Traffic Sign Recognition Based on Spatial Pyramid Pooling with Scale Analysis", Appl. Sci. 2020, 10, 6997. https://doi.org/10.3390/app10196997
5. Sichkar V.N., Kolyubin S.A. "Real time detection and classification of traffic signs based on YOLO version 3 algorithm", Scientific and Technical Journal of Information Technologies, Mechanics and Optics, 2020, vol. 20, no. 3, pp. 418–424 (in English). doi: 10.17586/2226-1494-2020-20-3-418-424
6. Zhang, J.; Huang, M.; Jin, X.; Li, X. " A Real-Time Chinese Traffic Sign Detection Algorithm Based on Modified YOLOv2" . Algorithms 2017, 10, 127. https://doi.org/10.3390/a10040127
7. Saleh S. M., Khwandah S. A., Hardt W., Hilbrich M. and Lazaridis P. I., "Estimating the 2D Static Map Based on Moving Stereo Camera," 2018 24th International Conference on Automation and Computing (ICAC), Newcastle upon Tyne, United Kingdom, 2018, pp. 1-5, doi: 10.23919/IConAC.2018.8749004.
8. Wang C.Y., Yue R.C., "Traffic Sign Detection Using You Only Look Once Framework" . [(accessed on 27 September 2018)]; Available online: http://cs231n.stanford.edu/reports/2016/pdfs/263_Report.pdf.
9. Arcos-García, Álvaro & Alvarez-Garcia, Juan & Soria Morillo, Luis. (2018). "Evaluation of Deep Neural Networks for traffic sign detection systems" . Neurocomputing. 316. 10.1016/j.neucom.2018.08.009.
10. Shustanov, A., Yakimov, "P.: Cnn design for real-time traffic sign recognition". Proc. Eng. 201, 718–725 (2017). https://doi.org/10.1016/j.proeng.2017.09.594
11. 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), 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.
12. Bochkovsky, Alexey, Chien-Yao Wang and H. Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection." ArXiv abs/2004.10934 (2020): n. pag.
13. Stallkamp, J.; Schlipsing, M.; Salmen, J.; Schlipsing, M. "The German traffic sign recognition benchmark: A multi-class classification competition". In Proceedings of the IEEE International Joint Conference on Neural Networks, San Jose, CA, USA, 31 July–5 August 2011; pp. 1453–1460.
14. Wang, G.Y.; Ren, G.H.; Wu, Z.L.; Zhao, Y.Q.; Jiang, L.H. "A robust, coarse-to-fine traffic sign detection method". In Proceedings of the 2013 International Joint Conference on Neural Networks, Dallas, TX, USA, 4–9 August 2013; pp. 754–758.
15. Redmon, J., Divvala, S., Girshick, R., Farhadi, " A.: You only look once: unified, real-time object detection". In: IEEE CVPR, pp. 779–788 (2016)
16. Redmon, J., Farhadi, "A.: YOLO9000: better, faster, stronger". In: IEEE CVPR, pp. 7263–7271 (2017)