IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 20, 2020, accepted October 1, 2020, date of publication October 14, 2020, date of current version October 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3031191

Neural-Network-Based Traffic Sign Detection and Recognition in High-Definition Images Using Region Focusing and Parallelization

ALEKSEJ AVRAMOVIĆ[1], DAVOR SLUGA[2], DOMEN TABERNIK[2], DANIJEL SKOČAJ[2], (Member, IEEE), VLADAN STOJNIĆ[1], (Member, IEEE), AND NEJC ILC[2], (Member, IEEE)

[1]Faculty of Electrical Engineering, University of Banja Luka, 78000 Banja Luka, Bosnia and Herzegovina
[2]Faculty of Computer and Information Science, University of Ljubljana, 1000 Ljubljana, Slovenia

Corresponding author: Nejc Ilc (nejc.ilc@fri.uni-lj.si)

This work was supported in part by the Ministry of Scientific and Technological Development, Higher Education and Information Society of Republic of Srpska, under Contract 07.051/68-14/18 and Contract 19/6-020/961-144/18, in part by the Bilateral Academic and Technological Cooperation between Bosnia and Herzegovina and Slovenia under Contract 19-6-020/964-25-1/18, and in part by the Slovenian Research Agency through the Bilateral Collaboration Project Grant BI-BA/19-20-047 and under Grant P2-0241.

ABSTRACT Recent trends in the development of autonomous vehicles focus on real-time processing of vast amounts of data from various sensors. The data can be acquired using multiple cameras, lidars, ultrasonic sensors, and radars to collect useful information about the state of the traffic and the surroundings. Significant computational power is required to process the data fast enough, and this is even more pronounced in vehicles that not only assist the driver but are capable of fully autonomous driving. This article proposes speed and accuracy improvement of traffic sign detection and recognition in high-definition images, based on focusing on different regions of interest in traffic images. These regions are determined with efficient and parallelized preprocessing of every traffic image, after which convolutional neural network is applied for detection and recognition in parallel on graphics processing units. We employed different ''You Only Look Once'' (YOLO) architectures as baseline detectors, due to their speed, straightforward architecture, and high accuracy in general object detection tasks. Several preprocessing procedures were proposed, to achieve real-time performance requirement. Our experiments using a large-scale traffic sign dataset show that we can achieve real-time detection in high-definition images with high recognition accuracy.

INDEX TERMS Traffic sign detection, traffic sign recognition, CUDA, CNN, deep learning, high-definition images, GPU.

I. INTRODUCTION

Modern vehicles are getting smarter. Equipped with various sensors, powerful computers, and state-of-the-art algorithms for traffic estimation, they provide reliable driving and parking assistance. Furthermore, the latest achievements in the car industry allow autonomous vehicles to safely participate in everyday traffic without human interference, only relying on information extracted from multiple visual and motion sensors. Vehicles must be equipped with hardware able to process vast amounts of data from different types of sensors in real-time. Usually, these are custom-tailored computers capable of tackling large computational loads reliably. Such special hardware packages make autonomous vehicles much more expensive in comparison to non-autonomous ones. Upcoming versions of modern vehicles will inevitably include even more sensors requiring even more computational power. We can expect future car computers will provide us with more advanced driving assistance soon. For example, the car can use front and rear cameras for traffic jam prediction, estimation of safe distance from other vehicles, blind spots monitoring, warning on fast-approaching vehicles, or the sudden appearance of cyclists and pedestrians on the road. Also, it can utilize front cameras for real-time detection and recognition of multiple objects, i.e., traffic signs, street names and numbers, and more

The associate editor coordinating the review of this manuscript and approving it for publication was Razi Iqbal.

accurate estimation of local orientation. The integration of the collected data may further lead to speed correction assistance to avoid traffic jams. Additionally, vehicles can be equipped with sensor systems capable of real-time traffic sign inventory maintenance and observation of near-road objects. In most of the mentioned scenarios, a fast and accurate object detection on images is crucial.

It is well-known that deep learning techniques, e.g., convolutional neural network (CNN), offer the best performance for object detection and recognition in images. Considering practical applications described above, it is essential to consider the processing time of an individual image and object detection accuracy (objects can be of traffic signs and other traffic-related items). The processing time is correlated with the network architecture: generally, more time is required when more layers are used and the number of generated features grows with the input image size along with the needed computational power and memory. Modern deep learning algorithms often rely on the computational performance of powerful graphic processing units (GPUs) capable of performing a large number of simple operations in parallel. Propagation of a single image through a deep convolutional network may take only several milliseconds using the latest GPU hardware. Also, CNN can be implemented on embedded hardware devices with limited computation power, which negatively affects the processing time or/and the recognition performance. The ongoing improvements of GPUs are making them more powerful, cheaper, and more accessible, so it is reasonable to expect that similar hardware will appear in vehicles as an extension to the car's computers shortly. For example, GPU-like NVIDIA Drive AGX is a computationally powerful card customized for processing data from multiple devices such as cameras, sensors, lidars, and radars.

This article examines the possibilities of implementing real-time and large-scale traffic sign detection (TSD) and traffic sign recognition (TSR) based on the processing of high-definition (HD) traffic images. The proposed approach could be adapted for general traffic object detection and recognition. Experiments were done to examine the impact of the number of features on detection and recognition accuracy and processing time.

We propose efficient and parallel preprocessing of HD traffic images to detect regions-of-interest (ROI) that contain traffic signs with high probability. These ROIs can be then sent to one or multiple GPUs for processing, thus parallelizing TSD/TSR for one particular traffic image. This way the backbone network can focus on one particular region in an image, neglecting background features. Of course, one traffic image can contain traffic signs in different places. The proposed algorithm detects up to three ROIs. If multiple GPUs are available, each ROI can be processed in parallel, thus significantly increasing processing speed for one particular traffic image. Among several single-shot detection networks, we used the different ''You only look once'' (YOLO) architectures as the basis due to the low propagation latency and excellent detection performance. We experimented with simple architecture TinyYOLO, intermediate architecture YOLOv3, and more complex YOLOv4. We compared TSD/TSR performance in the case when ROIs are fixed on predefined positions, based on *a priori* knowledge of traffic sign positions, with the case when the ROIs' positions are adjusted using an estimation of traffic signs positions in each frame. The method for positioning the ROIs is a simplified salience detection algorithm, which slightly adjusts the predefined position of ROIs so that we achieve better coverage of traffic sign instances.

The paper is organized as follows. Section II gives an overview of the related work, Section III describes the dataset used in this research, while Section IV presents the most important characteristics of the YOLO algorithm. In Section V, initial experimental results are given, showing the performance of our proposed image processing pipeline, which we describe in Section VI. Detailed analysis of experimental results is given in Section VII, while Section VIII gives conclusion remarks.

II. RELATED WORK

According to the accepted terminology, images captured from the cameras mounted on cars in traffic (containing traffic signs) will be referred to as *traffic sign images* and term *traffic sign instances* refers to physical, real-world traffic signs. Detection and recognition of traffic sign instances received notable attention from the computer-vision community in the past decade. Early research included low-level feature extraction to detect and recognize the most important traffic signs, namely mandatory and prohibitory signs (circles, squares, and triangles), using primary color and shape features. Besides using low-level features, these solutions were limited to a small number of different traffic sign categories. These methods relied on time-consuming algorithms for determining the location of traffic sign instances in images, or they concentrated only on the classification of pre-cropped traffic sign instances. Initially, the research on TSR was inspired by the necessity for automatic traffic sign inventory management [1]–[3], and later to provide more reliable driving assistance [4]–[7].

In later years, the research focus was mostly on mid-level features for traffic sign recognition to achieve light-, rotation-, scale-, and distortion-invariance. In [8], [9], a systematic benchmark TSR dataset was collected containing more than fifty thousand images of traffic sign instances, manually labeled into 43 different categories. An excellent performance was demonstrated on the classification of image cutouts or patches containing traffic sign instances using both mid-level features and shallow depth networks. Another interesting overview was presented in [10], where authors explored the influence of different aspects such as visual sensors, camera calibration, color model, and global illumination on recognition of traffic sign instances. In [11], complementary mid-level features were augmented and used for extreme learning machine algorithms and demonstrated a remarkable performance. Still, localization of traffic sign instances inside

an image remained an open issue. In [12] authors give a complete overview of mid-level TSD algorithms with a comparison and discussion of different approaches to TSD.

Lately, researchers mostly used convolutional neural network for TSR and TSD tasks. Much research was done about using CNN for TSR (i.e., classification of pre-cropped traffic sign instances), either as a feature extractor or as a classifier [13]–[17]. The latest approach combines TSD and TSR, as presented in [18]–[20], where excellent performance was demonstrated on localization and recognition of traffic sign instances. Since we focus on the utilization of YOLO-architecture in TSD/TSR, we carefully evaluated related references. An evaluation of the YOLOv2 version for TSD/TSR was done in [21], where authors demonstrated high precision and small latency. However, they used traffic sign datasets with a limited number of different sign categories. An example of YOLOv3-based TSD/TSR is given in [22], where authors used their dataset with 22 categories from the Vietnamese traffic sign dataset. However, they reported a large number of misdetections, and there is no time consumption analysis. To the best of our knowledge, no other papers on YOLO-based methods for TSD/TSR have been reported. Moreover, it is noticeable that although several papers report real-time TSD/TSR [23]–[25], none of them considered real-time detection and recognition in HD images entirely based on CNN. In [17], [26], real-time detection is reported, where detection is done by image filtering while CNN is used for recognition only.

Most of the research is limited in the sense of the variety of traffic sign instances, where the dataset contains only a subset of real-world traffic signs. Most common traffic signs were designed to vividly and instantly provide warnings, prohibitory or mandatory restrictions. Their simple design enables handcrafted image processing algorithms to perform well. On the other hand, there are numerous categories of different traffic signs that are informative and much harder to detect and recognize using handcrafted algorithms, as explained in [20].

The latest report regarding real-time traffic sign recognition came from the NVIDIA developer center [27], which developed a multi-level CNN for traffic lights and traffic sign detection and recognition. In essence, traffic sign detection and recognition is done using hierarchical convolutional deep network model, where different key features are detected separately and then combined into the final output class. Their network can detect 300 different U.S. and more than 200 European traffic sign categories, but the dataset is not publicly available, nor are there any detailed experiment reports.

III. DFG TRAFFIC SIGN DATASET

Several traffic sign datasets were made publicly available to enable benchmarking and testing of proposed algorithms. Some of these datasets provided images for TSR only, while others offered TSD as well. Although, some of them provide vast amounts of data, like the Tsinghua-Tencent 100k dataset [19], the dataset with the highest number of different traffic sign categories is The Belgium Traffic Signs (BTS) dataset [28] that contains 62 different categories for detection and recognition. Also, several reports on TSD/TSR results were given on private datasets [5], [29]. In [30]–[32], an extended TSD dataset is introduced to enable benchmarking of TSD under different real and simulated conditions. Although this dataset includes many traffic images and videos, there are only 14 different traffic sign categories. An overview of these datasets, their specifics, and the number of different traffic categories is given in [20]. The work mentioned above also provided the first public large-scale TSD/TSR dataset, called the DFG traffic sign dataset, containing 200 different traffic sign categories. The majority of the images are in full high-definition (FHD) format. Moreover, the authors extended the original dataset with numerous artificially created traffic sign images. Among all available traffic sign datasets, we focused on the DFG dataset, since it is the only one with HD images and a large number of different traffic sign categories.

DFG traffic sign dataset[1] introduced in [20] consist of approximately seven thousand anonymized natural traffic sign images captured on Slovenian roads, most of which have full HD resolution of 1920×1080 pixels. Traffic sign instances are given with more than thirteen thousand tight annotations that describe the location and shape of instances using polygons. The complete set of images is divided in the training set (5254 images) and the test set (1703 images). Also, images in the dataset were chosen so that every traffic sign category has at least 20 annotations. In total, 200 different sign categories are included.

There are different types of traffic signs, some of which give warning, mandatory and prohibitive instructions on how and where to drive. In contrast, others only give useful, but not critical information. Concerning road safety, it is much more important to recognize mandatory and prohibitive signs accurately. Consequently, they are designed to have a specific color and shape. In general, traffic sign categories within one specific type share a similar shape and color features and have specific meanings. For example, warning signs are triangular with a red frame and give us information about the curves on the road ahead, landslides, etc. (type I in Fig. 1), prohibitory and mandatory signs are mostly round with a white background and a red frame or with a blue background only (type II in Fig. 1). Traffic signs of this type give information about speed limits, priorities, and prohibitions. DFG dataset also contains informative signs (type III in Fig. 1, round or rectangular of different colors), supplementary signs (type IV in Fig. 1, rectangular shape with different inscriptions), road signs (some of type III categories and type VII in Fig. 1, rectangular with different background colors, different inscriptions, different arrow directions), as well as bumpers, mirrors and electric speed limits (type X in Fig. 1).

[1] Dataset is publicly available at https://www.vicos.si/Downloads/DFGTSD

FIGURE 1. Categories in DFG traffic sign dataset.

Examples of every traffic sign instance from the DFG dataset are given in Fig. 1.

We notice that some traffic sign instances of different categories (even different types) can have very similar visual characteristics. For example, we can compare the signs from category I-39-3 and VI-3.1-1 and notice that they have the same stripe pattern with a difference in thickness. Another interesting example is that the difference between III-14 and III-14.1 is an additional black frame in the latter category. Some sub-categories differ in one particular feature, such as the direction of the arrow.

DFG dataset also contains an extension with artificially created traffic sign images. This data augmentation is needed to ensure that every category contains at least 200 instances. Artificial images are created using traffic images without traffic sign instances, where tightly cropped instances are inserted into traffic images on random positions. Moreover, the instances were edited so that their geometry, shape, and brightness match the natural images. Overall, the extended dataset contains more than 15k traffic sign images and more than 43k different annotations. It is also important to emphasize the fact that traffic sign instances in natural images should be expected in regions where traffic signs are usually located - right from, left from, or above the vehicle.

In our work, we focus on TSD/TSR performance comparison with the results reported in [20]. The best score is achieved by using an adapted Mask R-CNN network trained on the extended dataset. The reported mAP^{50} is 95.5% and the processing of one single image takes approximately 500 ms on an NVIDIA GeForce GTX 1080 Ti GPU.

IV. FAST OBJECT DETECTION

The introduction of CNN made a significant breakthrough in object detection and recognition in natural images. Most prominent architectures that unify both detection and recognition of objects are Mask R-CNN [33], Retina-Net [34], SSD (Single Shot Detector) [35], YOLO [36]–[39], and Efficcient-Det [40]. Different anchor-free object detection approaches were introduced: RepPoint network [41], where representative points are learned to arrange themselves in the manner of object's bounds; CenterNet [42], where triplets are detected to improve localization; CornerNet [43], where keypoints are detected; and FCOS [44] as a fully convolutional one-stage detector. Most recently, Detection Transformer (DETR) [45] method encodes prior knowledge about the object detection task, removing the necessity for non-maxima suppression or anchors. Above all, it was reported that Mask R-CNN achieved an excellent performance in object detection, recognition as well as segmentation. It uses a two-stage network architecture as in Faster R-CNN [46], [47]: the first stage is a network that proposes candidate objects' regions, and the second stage then performs classification and bounding box regression. Faster R-CNN was extended to Mask R-CNN so that pixel segmentation of proposed regions is done in parallel with the classification. The disadvantage of a two-stage network architecture is long propagation time, making Mask R-CNN fairly slow when dealing with large images. Although presented as a single-stage detector, Retina-Net uses a network pyramid structure for feature extraction and two subnets for object classification and bounding box regression. It also utilizes a modified loss function to improve performance

when there is a category imbalance in the training set. Single Shot Detector [35] is an architecture designed to detect a large number of objects within one image quickly. It uses every generated feature map as input to the detector. The main disadvantage is that only low-level features extracted in the initial layer can be used for small object detection, thus reducing the performance. EfficientDet [40] introduced several optimizations (bi-directional feature pyramid network and compound scaling) to improve computational efficiency and improve performance.

In our experiments, we use three different YOLO architectures [38], [39], since they are proven to be fast and accurate one-stage detectors and capable of large-scale detections. YOLOv3 [38] is a one-stage detector that has a fully convolutional architecture with three different detection layers for small, medium, and large objects. All these detectors are given features obtained as a combination of the output of the initial and the later layers. It also separately predicts objectness (the presence of an object at one specific point in the image) and the conditional class of an object, which successfully deals with category imbalance. YOLOv3 unifies object detection and recognition into one propagation through the neural network. It is based on Darknet-53, which contains 53 convolutional layers for encoding. It also includes a decoding part with upsampling layers, skip-connections, and additional convolutions. One of the most exciting features of YOLOv3 is the ability to use input images of different sizes, as long as they are a multiple of 32. It means that the network can be trained on one specific input image size but used to detect objects in images of different sizes. In general, YOLOv3 was reported to achieve outstanding detection and recognition accuracy with shorter processing time in comparison to other network architectures dedicated to object detection.

As a part of the same project, authors proposed a lite version called TinyYOLO based on the same principle of feature extraction and object detection as YOLOv3, but with fewer layers and notably faster. TinyYOLO uses the same techniques as YOLOv3, except it contains 10 convolutional layers in the encoding part, does not use skip-connections, and make detections on only two scales. It also accepts input images of different sizes as long they sizes are multiples of 32.

Most recently, an upgraded and performance-tuned version YOLOv4 [39] was introduced. YOLOv4 is a more complex and deeper network architecture based on the same principles as YOLOv3 and TinyYOLO, but with incorporated improvements. The authors examined the possibilities to combine a large number of state-of-the-art improvements in neural-network-based object detection with the YOLO-based backbone architecture. Among many, they decided to use: (**a**) cutting edge techniques for model training, such as mosaic-based data augmentation and self-adversarial training; (**b**) continuously differentiable activation function (Swish and Mish) and function for self-normalizing networks (SELU); (**c**) additional term for bounding box regression loss (parametric IoU-based losses); (**d**) advanced regularization methods; (**e**) state-of-the-art normalization methods (such as Cross mini-batch normalization); (**f**) multi-input weighted residual connections and cross-stage partial connections; (**g**) hyper-parameter optimization based on genetic algorithm and (**h**) cross-stage-partial-connections, which reduced computational expenses. After additional modification and tuning, the authors optimized the performance of YOLOv4 for both CPU- and GPU- based processing and showed superiority regarding both detection accuracy and time requirements on general object detection dataset.

Any YOLO-based detector will try to find one object on each central point in every grid cell. The grid is made by the upright rectangular division of an input image. Each grid cell is defined with a 32×32 pixel neighborhood, so the number of grid cells depends on the size of an image. The vertical and horizontal number of grid cells equal the image height and width divided by 32, respectively. Therefore, the number of objects that can be detected depends on the input image size. The larger the images are, the more grid cells are used. Moreover, since the whole image context is used for training, objects from nearby grid cells can overlap, and each grid cell can be used to detect an object on different scales. During the propagation process, the input image is downsampled with the stride size equal to 32, so the first detection layer is used to detect large objects. Succeeding layers include up-sampling convolution, skip-connections, and feature concatenations. The second layer receives a feature map that is downsampled with the stride size equal to 16 and is used to detect medium-sized objects. Finally, the third detection layer receives a feature map that is downsampled with the stride size equal to 8 and is meant to detect small objects. The described detection scheme is implemented in both YOLOv3 and YOLOv4 architecture. Thus, we will evaluate the proposed preprocessing and parallelization method for consistent improvement based on all three different architectures.

V. TRAFFIC SIGN DETECTION AND RECOGNITION USING YOLOv3, YOLOv4 AND TinyYOLO

In this section, we investigate the performance of YOLOv3, YOLOv4, and TinyYOLO architectures on the DFG traffic sign dataset on full HD (FHD) images. Two criteria are relevant here: how accurate the detection and recognition is and how fast one particular image can be processed. As a measure of TSD/TSR accuracy, we used mean average precision mAP^{50} over all categories,[2] which is usual for general object detection and recognition [33], [35]–[39]. Processing time is the average time needed for the processing of one image.

The question is: can HD traffic images be processed fast enough for real-time applications? We often associate real-time video processing with at least 25 frames per second (FPS), i.e., 40 ms per frame. However, do we need to pursue 25 FPS also in the context of TSD/TSR? The data from visual

[2]True positive detection occurs when Intersection over Union of the detected and ground truth boundary box is more then 50%, which is denoted as IoU.5

sensors are often combined with the GPS data (most often with the precision around or above 1 m), and traffic signs are often visible and recognizable from several dozens of meters (and more on highways). Furthermore, the regulations in the country of Slovenia (should be similar across the whole European Union) state that the distance between road signs should be at least 15 m when driving 50 km/h, 30 m at 90 km/h, and at least 100 m when driving more than 100 km/h. If we want to capture a traffic sign in multiple consecutive frames to improve the recognition accuracy, it is reasonable to take an image every 2 m. Thus, if the vehicle speed is 50 km/h ($\approx$ 14 m/s), the computer needs to process at least seven frames per second to achieve this performance. If we increase speed to 90 km/h or 130 km/h, the target frame rate becomes 13 or 18 FPS, respectively.

For the initial experiment, we used a publicly available official implementation of YOLOv3, TinyYOLO [38], and YOLOv4 [39]. We initialized all three networks with pre-trained weights obtained by training on the ImageNet dataset. In order to make a fair comparison, we made two restrictions: (**a**) no random image resizing is allowed, and (**b**) no image flipping is allowed. Random image resizing during the training can help to localize and recognize objects on different scales more accurately, so in the first experiment, we skip this option to evaluate the capabilities of different architectures fairly. Image flipping is one option for data augmentation, which means that images are randomly flipped horizontally before propagating through the network. This option can decrease the performance when dealing with traffic signs since flipping can cause confusion (e.g., left and right arrows from categories III-107-1 and III-107-2 in Fig. 1).

Pre-trained networks were fine-tuned on the DFG dataset (without additional artificial examples) using a stochastic gradient descent algorithm, with the initial learning rate set at 0.001, decreasing after 40 thousand iterations. Additionally, the data augmentation was done by making random color transformations on the training images for YOLOv3 and TinyYOLO and mosaicing for YOLOv4. We examined the processing time (FPS) and detection performance, i.e., mAP^{50}, in cases when networks were fine-tuned using different input image sizes. The processing time is the average propagation time for one image on the GeForce GTX 1080 Ti GPU. Following the usual methodology for object detection using YOLO, we resized images into a square shape (608 $\times$ 608 pixels, 640 $\times$ 640 pixels, 672 $\times$ 672 pixels, 704 $\times$ 704 pixels), so the aspect ratios of traffic sign instances were altered. We also tested the case when we reduced the size of input images but preserved the original aspect ratio (960 $\times$ 540 pixels). Image size reduction before network training reduces memory requirements and decreases the training time, but uses less input features. Since TinyYOLO and YOLOv3 are less complex models, it is possible to train them with FHD images (reduced to 1920 $\times$ 1056 pixels to obtain image size that is a multiple of 32) using the GeForce GTX 1080 Ti. That is not possible for YOLOv4, at least not with the amount of memory available on the 1080 Ti graphic card.

Considering the results given in Tab. 1, we notice that more features (larger input) result in larger mAP^{50}, although there were some deviations for TinyYOLO and YOLOv3 architectures. Reasons for these deviations can be geometry distortions caused by aspect ratio change, unequal amount of training traffic sign instances in each category, or inadequate training. Also, we assume that in these cases, we lost some features useful for accurate detection and recognition. For example, YOLOv3 with images in FHD (1920 $\times$ 1056 pixels) achieves at least 1.4 percentage points (p.p.) better mAP^{50} compared to images that were downsampled by a factor more than two (704 $\times$ 704 pixels and below).

TABLE 1. Comparison of the performance of different approaches for TSD/TSR using YOLO architectures in terms of mAP^{50} and frame rate (FPS).

architecture	size	mAP^{50}	FPS
TinyYOLO	608 × 608	78.76	94
TinyYOLO	640 × 640	79.48	93
TinyYOLO	672 × 672	81.68	90
TinyYOLO	704 × 704	81.28	88
TinyYOLO	960 × 544	82.75	81
TinyYOLO	1920 × 1056	88.70	40
YOLOv3	608 × 608	87.33	34
YOLOv3	640 × 640	87.58	32
YOLOv3	672 × 672	89.08	29
YOLOv3	704 × 704	88.99	28
YOLOv3	960 × 540	89.11	27
YOLOv3	1920 × 1056	90.44	8
YOLOv4	608 × 608	91.90	35
YOLOv4	640 × 640	92.43	33
YOLOv4	672 × 672	92.49	30
YOLOv4	704 × 704	92.65	29
YOLOv4	960 × 540	92.68	28

It is interesting to notice that the detection performance when using TinyYOLO architecture and FHD images is comparable to the case when we use YOLOv3 with downsampled images, but is faster. The reason can be the larger number of features present in the original FHD image than in the downsampled images.

The most complex architecture YOLOv4 further improves mAP^{50} and achieves the best performance for downsampled images. Although YOLOv3 based detection with FHD images uses more features than YOLOv4, the latter can extract more discriminative features and propagate them approximately at the same speed as YOLOv3. On the other hand, the YOLOv4 model consumes much more memory. To fit into GPU's memory, original images were downsampled accordingly.

The best mAP^{50} was achieved using the YOLOv4 architecture and input image size of 960 $\times$ 540 pixels when it

reaches 92.68%. It is also interesting to notice the discriminative capabilities of YOLOv4 architecture, even in cases when fewer input features (smaller input image size) were used. YOLOv4 achieves mAP^{50} over 92% even after considerable image size reduction and modification of aspect ratio. Based on these results, it is recommended to retain as many original features as possible. As mentioned before, the YOLOv4 model requires more memory than YOLOv3, but the built-in optimizations made it at least as fast or even faster compared to YOLOv3.

VI. THE PROPOSED METHOD

As mentioned above, we can find the motivation for region-based traffic sign detection in the every-day-based experience. On the road, drivers mainly focus on what happens in front and beside them (i.e., looking through the front and side glasses) and what happens behind them (looking on rear-view mirrors). When a driver notices a traffic sign (or any other important traffic subject) in front of them, they concentrate on recognizing the meaning of the traffic sign for one short time interval, until the focus is back on the road again. We implemented a similar concept in the proposed solution. The traffic signs are located somewhere in a traffic image: mostly on the right side of the road, in some cases above the road, and rarely on the left side. In each case, they are on standardized height, so they are more probable to be found in specific regions of traffic images. Therefore, when searching for traffic signs, there is no need to analyze the complete traffic image. Instead, we can focus on one specific region where a traffic sign instance is more likely to be found. The proposed solution mimics the behavior of human drivers: it inspects the whole traffic image briefly and concentrates on one specific region if there is a necessity to recognize any critical traffic object. This suggests that a traffic image should be preprocessed to identify regions of interest (ROIs), which will be processed further. By selecting ROIs, we reject a large number of background features before traffic sign detection takes place. Another motivation for using ROIs is that the propagation of smaller images through the network is considerably faster.

Moreover, the analysis of ROIs will preserve the original aspect ratio of traffic sign instances and other essential objects in the traffic image. Most importantly, multiple image regions can be processed in parallel if multiple GPUs are available. Instead of processing one HD image, we propose the processing of a variable number of smaller ROIs in parallel. We introduce a preprocessing step, which includes parallelized image processing algorithms to detect regions where traffic sign instances are more likely to be located. These regions are sent separately to different GPUs for processing. The more GPUs are available, the more ROIs can be processed in parallel, and consequently, the more substantial speed-up can be achieved.

In the following experiments, we have exploited these facts to improve detection performance using fixed and movable ROIs. At first, we experiment with fixed regions, in which case their position is determined according to a predefined model. Later, we analyze the performance where movable ROIs are positioned by the output of low-level-feature based detection of traffic sign instances. We experimented with regions of 704×704, 672×672, 640×640 and 608×608 pixels due to the following reasons: (**a**) YOLO requires each region dimension to be a multiple of 32; (**b**) we wanted to investigate a region size that is sufficient to cover all traffic sign instances in every FHD image from the DFG dataset; (**c**) each region can be propagated fast enough for real-time performance for all four examined sizes.

A. DETECTION USING FIXED REGIONS

In Fig. 2, we present the heatmap generated from positional data of traffic sign instances from training DFG set. We can notice that most of the instances are located on the upper right side of traffic images. Of course, due to the arbitrary position of a car and the camera during driving, some signs can appear in unpredictable image areas at a specific moment. We propose cutting out fixed image regions based on the analysis of exact traffic instances positions in the complete training set, as shown in Fig. 3. This way, *a priori* knowledge about traffic sign instances' positions is used. Moreover, fixed ROIs may intersect with traffic sign instances. These issues are present while doing TSD/TSR on the DFG dataset since the images were not taken in close succession. However, it is much more probable that traffic signs will be present in predefined ROIs in at least some successive images while driving.

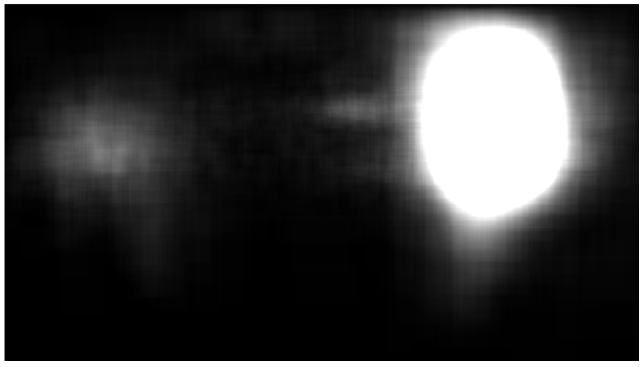


FIGURE 2. Probability heatmap of traffic sign instances coverage estimated from training traffic images.

B. DETECTION USING MOVABLE REGIONS

The second approach tries to increase detection performance by adjusting the positions of ROIs based on individual image analysis, Fig. 4. The goal is to address the images in which traffic sign instances are not in the *a priori* positioned regions. We estimate the best position of the ROIs, based on image processing for salience detection. This scenario requires some additional processing time, with the expected improvement of detection performance.

During preliminary testing, we achieved the best performance when we allowed ROIs to move freely along

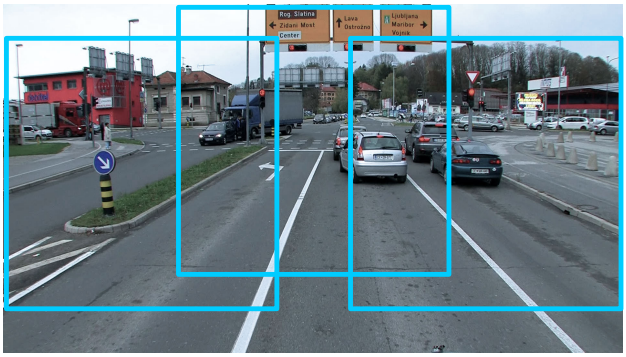


FIGURE 3. Fixed regions of interest shown as black squares. All three ROIs are set to predefined positions.

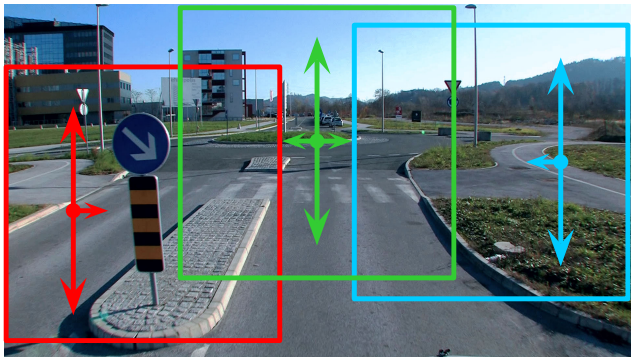


FIGURE 4. Movable regions of interest. All three regions can be moved from their initial positions depending on the predictions made by our algorithm.

the vertical axis and constrained their movement along the horizontal axis, as depicted in Fig. 4.

The proposed algorithm initially converts an image from RGB to HSV color space. Then it enhances the contrast using contrast-limited adaptive histogram equalization (CLAHE) [48], after-which thresholding is applied to obtain seven binary masks that are then combined to four masks that relate to red, blue, yellow and green color. Thresholds are determined heuristically from the training set of the DFG dataset and are given in Table 2. Each binary mask is then processed separately using morphological filters [49] and filtering of blobs detected by connected components analysis.

TABLE 2. Thresholds used for the extraction of binary masks from an image in HSV color space.

	thresholds		
	hue	saturation	value
red	0.915 - 0.030	0.450 - 1.000	0.100 - 1.000
red/brown	0.000 - 0.100	0.650 - 1.000	0.500 - 1.000
blue	0.520 - 0.700	0.620 - 1.000	0.250 - 1.000
dark **yellow**	0.050 - 0.130	0.600 - 1.000	0.300 - 1.000
light **yellow**	0.130 - 0.180	0.640 - 1.000	0.200 - 1.000
green	0.370 - 0.500	0.450 - 1.000	0.250 - 1.000
light **green**	0.170 - 0.260	0.630 - 1.000	0.500 - 1.000

After filtering, we merge all masks and get a set of blobs. Finally, we cover as many blobs' pixels as possible using greedy positioning of the three ROIs, defined as rectangles of size 704×704 pixels.

Morphological filters applied to each binary mask include image reconstruction, dilation, and erosion with a square-shaped kernel. Then, we fill in holes and employ connected components analysis on every mask individually. The latter includes finding blobs and their filtering based on different criteria (area, width, height, aspect ratio, extent, and area-to-squared-perimeter ratio). We also considered the context of traffic images and used different filtering thresholds for the upper and lower part of the image. Moreover, red blobs that touch image borders are filtered out as they most probably represent red roofs or bike lanes. Similarly, we removed the blue sky by ignoring large blue blobs in the upper part of the image.

We implemented these steps in CUDA – an application programming interface for GPUs, to achieve better performance. Existing computer vision libraries like OpenCV [50] offer CUDA implementations of some of the aforementioned image processing methods, but are in most cases too general and thus less efficient. By developing our own optimized parallel CUDA implementations for thresholding, blob filtering, connected component labeling, morphological filtering, and hole filling algorithms, we were able to reduce the execution time significantly. We streamlined our algorithm for positioning the ROIs and took care to minimize memory transfers between GPU and CPU memory, which are a bottleneck in such situations. By doing this, we were able to reduce the execution time by a factor of 50 compared to an implementation using only OpenCV functions, thus enabling us to achieve real-time performance.

After blob filtering, we perform the ROI positioning with a fast greedy algorithm that tries to cover as many blobs' pixels as possible. This step is performed on the CPU as it is inherently not parallel and already fast enough. The starting coordinates are the same as for the fixed ROIs, and the vertical movement of all three ROIs is limited only by the edges of the image. Considering horizontal movement: the left ROI is allowed to move 50 pixels to the right at most; the right ROI is allowed to move 50 pixels to the left at most, while the central ROI is allowed to move mostly 96 pixels to the left or right. We have discarded several bottom rows of the image because it is highly unlikely that traffic sign instances will appear there. We show an illustration of the described constraints in Fig. 5. In Fig. 6 we show some examples of movable ROIs estimation.

VII. EXPERIMENTAL ANALYSIS AND DISCUSSION

A. METHODOLOGY

1) THE TRAINING SET

To properly train the network for detection using ROIs, we built a new training set from the original DFG dataset. We used a sliding window method to extract all regions of

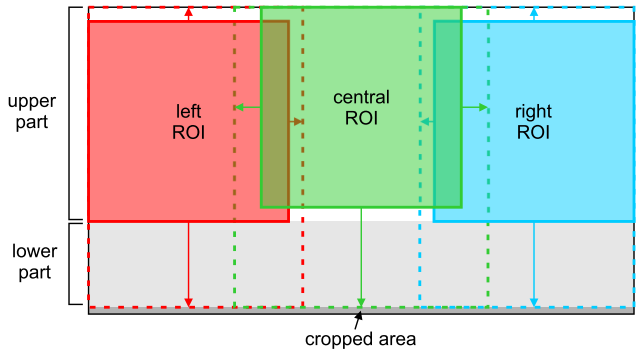


FIGURE 5. An illustration of the upper and lower part of an image and possible positions of all three ROIs.

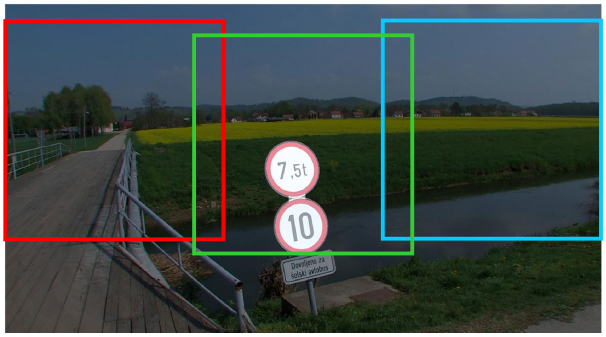
size 704 × 704, 672 × 672, 640 × 640, and 608 × 608 pixels that cover all traffic signs in an image, except those that were already incomplete in original images. Among all candidate regions, we select the one with the most traffic signs instances, without considering the position of instances. In other words, traffic sign instances can be found anywhere in a region. If any traffic sign instances remain after selecting the first region, the process is repeated several times, so one original image can give more than one region for training. This way, all instances from the original dataset are copied into the new one.

2) FIXED ROIs

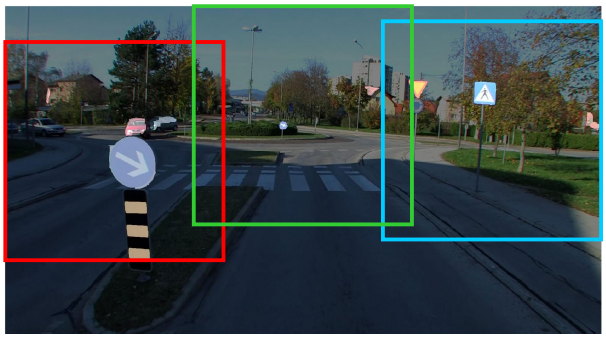
In this experiment, we examined the detection performance of different YOLO architectures and region sizes when using fixed ROIs. In all scenarios, we trained the neural network on the new training set. The parameters for training (learning rate and color-based data augmentation) are the same as in the initial experiment presented in Section V. We analyzed the performance using different input sizes: 704 × 704, 672 × 672, 640 × 640, and 608 × 608. The positions of ROIs are fixed for all four region sizes, i.e., the central pixel for each region is the same for every region size. We show the positions of regions of size 704 × 704 pixels in Fig. 3. The central ROI is aligned horizontally on the center and vertically to the top of the image. The left and right ROIs are vertically moved down by 50 pixels. The detection is made on each ROI, and all results are then merged.

3) MOVABLE ROIs

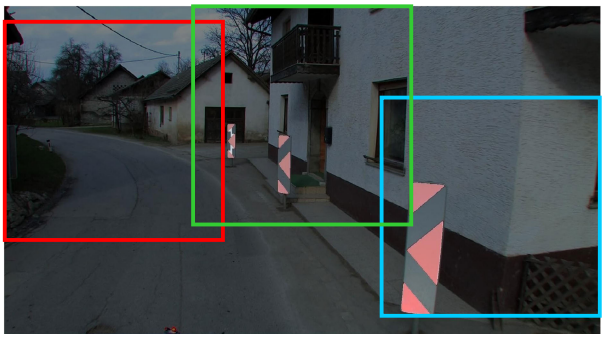
In this experiment, we replaced fixed ROIs with the proposed movable ROIs. The goal is to examine the possibility of a better coverage of traffic sign instances through conventional image processing. We estimated movable ROIs using the original FHD images and also their downsampled version (960 × 540). The idea was to evaluate the impact of downsampling on the algorithm for ROI placement. This experiment should show the benefits of movable ROIs in terms of



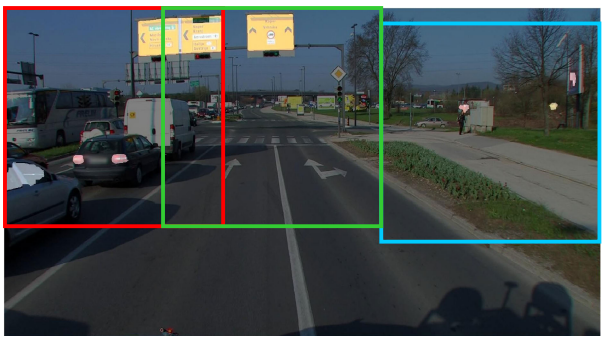
a) There are two entirely covered traffic sign instances and one partially. Overall covered area is 79.6% out of the whole instances area.



b) There are four entirely covered traffic sign instances and one partially. Overall covered area is 94.9% out of the whole instances area.



c) There are two entirely covered traffic sign instances and one partially. Overall covered area is 99.8% out of the whole instances area.



d) All four traffic sign instances are fully covered.

FIGURE 6. The results of the proposed algorithm for the positioning of movable ROIs. The movable ROIs are depicted as red, green, and blue squares. The original images were processed so that the detected blobs stand out from the rest of the image.

detection accuracy. However, we expect additional time costs for both original FHD and downsampled traffic images.

4) TRAINING ON EXTENDED DATASET AND USING MOVABLE ROIs

The drawback of the original DFG dataset is that some of the categories contain merely 20 instances, which may be insufficient for proper network training. As mentioned before, artificial traffic images were created to extend the dataset so that every category has at least 200 instances. However, traffic sign instances were artificially put on random positions, which means that our assumption about *a priori* sign position is no longer valid. This can result in poor performance of the proposed region-based approach. Thus, we did not include these additional artificial images in the test set to remain in line with real-world circumstances. To sum up, we trained TinyYOLO, YOLOv3, and YOLOv4 architectures on the extended dataset but evaluated them only on the original test set using movable ROIs.

5) MIXED-PRECISION TRAINING

Further improvements in both TSD/TSR accuracy and speed could be achieved by utilizing more sophisticated algorithms for finding regions of interest, using more training data, and using more computational power. In this experiment, we show how a hardware-based approach to speeding up the computations can be useful for TSD/TSR. New generations of NVIDIA GPUs, like Volta and Turing include tensor cores that work with mixed-precision arithmetic. Mixed-precision allows us to utilize the 16-bit floating-point format, thus reducing memory usage and speeding up the computation where precision is not crucial. In this experiment, NVIDIA GeForce RTX 2080 Ti card with tensor cores is used to train the YOLOv4 network and run the proposed movable ROI-based approach for traffic sign detection and recognition. The goal is to examine whether the training and detection based on mixed-precision calculations affect accuracy and speed.

B. RESULTS

We display the results using fixed ROIs in Tab. 3. By examining and comparing the previous results, given in Tab. 1, we can notice the following. TSD/TSR with fixed ROIs improves mAP^{50} by 1.45 p.p. when YOLOv3 architecture is used on FHD images. If three GPUs are available, TSD/TSR using fixed ROIs achieves at least 29 FPS on FHD images; when not using the region-based approach, we get only 8 FPS since we cannot utilize multiple GPUs. Although region-based detection with TinyYOLO architecture achieves 1.26 p.p. lower mAP^{50} in comparison to TinyYOLO on FHD images, it is faster by 6 FPS if only one GPU is available and by 98 FPS if we utilize three GPUs. TinyYOLO-ROI performs reasonably good: it achieves a 4 p.p. lower mAP^{50} score than YOLOv3-ROI with a much simpler architecture yielding around five times more FPS. The proposed region-based method with the YOLOv4 backbone improved

TABLE 3. Comparison of processing time and mAP^{50} for different YOLO architectures with fixed ROIs and with different number of GPUs. Labels include network architecture type (TinyYOLO/YOLOv3/YOLOv4), method abbreviation (fixed-ROI), and region size after resizing (608/640/672/704). The table shows FPS when using one (FPS1) or three (FPS3) GPUs in parallel.

architecture	size	mAP^{50}	FPS1	FPS3
TinyYOLO-ROI	608 × 608	85.61	56	169
TinyYOLO-ROI	640 × 640	86.66	52	157
TinyYOLO-ROI	672 × 672	86.89	49	147
TinyYOLO-ROI	704 × 704	87.44	46	138
YOLOv3-ROI	608 × 608	87.20	11	34
YOLOv3-ROI	640 × 640	90.15	10	32
YOLOv3-ROI	672 × 672	90.54	10	32
YOLOv3-ROI	704 × 704	91.87	9	29
YOLOv4-ROI	608 × 608	92.87	12	36
YOLOv4-ROI	640 × 640	93.03	11	34
YOLOv4-ROI	672 × 672	93.15	10	32
YOLOv4-ROI	704 × 704	93.44	9	29

mAP^{50} by 0.74 p.p. compared to YOLOv4 trained on 960×540 traffic images. In this experiment, all three architectures (simple TinyYOLO, intermediate YOLOv3, and complex YOLOv4) benefited from the ROI approach since many background features are discarded. The best performance was achieved for the region size of 704 × 704 for all three different architectures. In this experiment, all networks were fine-tuned on derived training datasets (as explained in Sec. VI), which cover all traffic sign instances from the original dataset. Thus, the only reason for the weaker performance of methods with smaller region sizes is inadequate coverage on test images. We decided to use a fixed region size of 704 × 704 pixels in further experiments based on the results so far.

We show the results obtained with movable ROIs in Tab. 4. As we can see from the comparison to results in Tab. 3, using movable ROIs increases mAP^{50} by at least 0.4 p.p., depending on the method. Overall, the best result is achieved by YOLOv4-mROI with a mAP^{50} of 93.87%. The additional processing time needed for adjusting the ROI positions is approximately 45 ms for the original FHD images (1920 × 1054) and only 23 ms for the downsampled images (960×540). Additional experiments indicated that there is no

TABLE 4. Comparison of processing time and mAP^{50} for different YOLO architectures with movable ROIs and with different number of GPUs. Labels include network architecture type (TinyYOLO/YOLOv3/YOLOv4), method abbreviation (movable-ROI), and region size after resizing (704 × 704). The table shows FPS when using one (FPS1) or three (FPS3) GPUs in parallel.

architecture	size	mAP^{50}	FPS1	FPS3
TinyYOLO-mROI	704 × 704	88.02	22	33
YOLOv3-mROI	704 × 704	92.56	7	17
YOLOv4-mROI	704 × 704	93.87	8	17

difference in ROI positions in these two cases. Thus, we opted for the faster variant. In effect, every image needs an additional 23 ms of processing time before the three regions can propagate through the network. If three GPUs are available, TSD/TSR using YOLOv3 or YOLOv4 with movable ROIs of size 704 × 704 achieves 17 FPS. Thus, using movable ROIs reduces the frame rate by 12 FPS compared to the fixed ROI approach. If using only one GPU (and sequential region processing), the difference in FPS is negligible (1-2 FPS). When using TinyYOLO, the improvement in mAP^{50} is similar to YOLOv3 or YOLOv4. However, the FPS improvement is considerable – the FPS drops by a factor of 3 in the worst case. Overall, all three architectures obtained better results with movable regions compared to the case when we used fixed regions. Movable regions helped to emphasize important features further since fine tuning their position reduces the number of partially covered traffic sign instances.

The results of training on extended dataset and using movable ROIs are in Tab. 5. We can quickly notice further improvement of mAP^{50} for all three architectures. The expanded training set has a positive effect on the learning of the neural network, with no additional frame rate penalties. The highest score of 94.43% was achieved by YOLOv4, although YOLOv3 architecture performed very well and achieved 93.71%, while Tiny YOLO stays behind. On the other side, TinyYOLO is at least 2-3 times faster compared to YOLOv3 and YOLOv4.

TABLE 5. Comparison of processing time and mAP^{50} for different YOLO architectures with movable ROIs and with different number of GPU, when trained with the extended dataset. Labels include network architecture type (TinyYOLO/YOLOv3/YOLOv4), method abbreviation (movable-ROI-extended), and region size after resizing (704 × 704). The table shows FPS when using one (FPS1) or three (FPS3) GPUs in parallel.

architecture	size	mAP^{50}	FPS1	FPS3
TinyYOLO-mROIe	704 × 704	88.75	22	33
YOLOv3-mROIe	704 × 704	93.71	7	17
YOLOv4-mROIe	704 × 704	94.43	8	17

As we can see from Tab. 6, mixed-precision based detection speeds up the proposed fixed ROI-based approach by a factor of approximately 1.5. When we use movable regions, we can not avoid the additional preprocessing time; thus, the speed-up factor is approximately 1.2. The difference in FPS between RTX 2080 Ti and GTX 1080 Ti is negligible when using only single-precision. In both cases, the achieved mAP^{50} is about the same as for single-precision detection.

TABLE 6. The effect of using mixed-precision on processing time and mAP^{50} for the proposed fixed ROI and movable ROI algorithms based on YOLOv4 architecture, when trained with the extended dataset. The table shows FPS when using one (FPS1) or three (FPS3) GPUs in parallel.

architecture	size	mAP^{50}	FPS1	FPS3
YOLOv4-ROIe	704 × 704	93.83	14	43
YOLOv4-mROIe	704 × 704	94.45	10	21

An overview of the best results is given in Table 7 and includes the results of the TSD/TSR without using ROIs and the YOLO networks trained on the extended training dataset. For comparison, we added the modified Mask R-CNN results on the same dataset [20].

C. DISCUSSION

Considering the results, a step-by-step improvement in performance measured by mAP^{50} is notable throughout the experiments. By processing only the regions of interest, the key features of traffic sign instances are emphasized, and the overall performance is improved. Our results show that TSD/TSR based on YOLO architecture using movable ROIs can compete with state-of-the-art solutions based on Mask R-CNN. Our method based on YOLOv4 achieves ∼1 p.p. lower mAP^{50} than the modified Mask R-CNN algorithm but is 4 or 8.5 times faster when using one or three GPUs, respectively. In Section V, we established that the target frame rate to achieve real-time performance for TSD/TSR is around 18 FPS. We showed that this is possible using our approach. If a higher frame rate is needed, efficient processing can be achieved either with fixed ROIs or simpler network architecture (TinyYOLO) for the cost of reduced detection performance.

Overall, we would like to emphasize the most notable findings:

(**a**) retaining the features contained in high-definition images can be beneficial for accurate traffic sign detection and recognition,
(**b**) feature selection through the implementation of a region-of-interest approach retains the important features from and around instances while speeding up the inference,
(**c**) estimating the region positions through image preprocessing improves the detection accuracy but requires additional time,
(**d**) using YOLO architecture makes real-time large-scale traffic sign detection and recognition in HD images feasible,
(**e**) the YOLOv4 architecture incorporates cutting-edge machine learning algorithms, achieving better performance with comparable or faster speed compared to predecessor architecture YOLOv3, but for FHD images YOLOv4 has memory consumption that is hard to satisfy even with the latest GPUs,
(**f**) the proposed region-based TSD/TSR approach is a compromise between speed and accuracy and can be applied to different network architectures to improve performance on FHD images,
(**g**) if sufficient computational power is available (multiple GPUs), a real-time TSD/TSR with state-of-the-art accuracy can be achieved,
(**h**) hardware-based computational simplification (i.e., using single-precision computation where computational accuracy is not critical) can further speed

TABLE 7. The performance of different approaches for TSD/TSR using YOLO architectures in terms of mAP^{50} and frame rate. For comparison, we display the results achieved by Modified Mask R-CNN in the lower part of the table.

	YOLOv4		YOLOv3		TinyYOLO	
	mAP^{50}	FPS1/FPS3	mAP^{50}	FPS1/FPS3	mAP^{50}	FPS1/FPS3
FHD images (basic dataset)	-	-	90.44	8 / -	88.70	40 / -
fixed-ROIs (basic dataset)	93.44	9/29	91.89	9 / 29	87.44	46 / 138
movable-ROIs (basic dataset)	93.87	8/17	92.56	7 / 17	88.02	22 / 33
movable-ROIs (extended dataset)	**94.43**	8/17	**93.71**	7 / 17	**88.75**	22 / 32

	Modified Mask R-CNN [20]	
	mAP^{50}	FPS1
FHD images (basic dataset)	93.00	2
FHD images (extended dataset)	**95.50**	2

up ROI-based TSD/TSR without affecting the performance,

(i) an image preprocessing can be applied to improve TSD/TSR performance further, but with the time delay costs.

Depending on the practical TSD/TSR application requirements, it is possible to choose an optimal architecture. If fast traffic image processing is required, with a modest detection/recognition accuracy, TinyYOLO architecture should be used. It is low memory- and computational-demanding and fast architecture. TinyYOLO can achieve excellent detection results on the most commonly used traffic signs. If there are a bit more relaxed time- and memory-consumption limits and high detection/recognition accuracy is recommended, then YOLOv4 is the best choice. It achieves state-of-the-art accuracy on large-scale traffic sign dataset with reasonable processing time. However, it requires significant hardware support to be used on FHD images. Somewhere between is YOLOv3, which has lower memory demands compared to YOLOv4 and comparable performance. Practical implementation of the proposed traffic sign detection and recognition algorithm requires significant hardware capabilities installed in vehicles, which means at least one GPU-like processing unit.

VIII. CONCLUSION

In this article, we examined the possibilities of speeding up the processing of high-definition traffic images to achieve real-time performance for automotive applications, including driving assistance, detection of near-road objects, autonomous driving, and automatic traffic sign inventory maintenance. We proposed a region-of-interest-based approach combined with YOLO architecture to achieve a favorable trade-off between detection accuracy and processing time. Employing regions of interest increases overall performance since less data is propagated through the network. We can achieve real-time processing with the support of appropriate hardware, e.g., a graphics processing unit. Moreover, several regions can be processed in parallel if multiple graphic cards are available. We used *a priori* knowledge on traffic sign instances position and image preprocessing for salience detection to focus on image areas where traffic sign instances most probably are. Throughout the series of experiments, we show a gradual improvement in detection performance. The propagation time per image is reasonably low, and in the domain of real-time. The detection accuracy of traffic signs is comparable with state-of-the-art algorithms when evaluated on DFG large-scale dataset while achieving at least three times lower processing time.

Our work can be further improved by considering the fact that the regions often do not contain any traffic sign instances. It would be beneficial to assess the probability that no traffic sign instances are present in a region; empty regions can be discarded, thus saving time and hardware resources. Furthermore, there is still room for improvement in fine-tuning the neural network to achieve better detection accuracy. It would also be interesting to adapt our method for video sequences of traffic imagery, where a traffic sign instance appears in multiple consecutive frames. We could exploit this temporal coherence to improve detection accuracy. Finally, our future work will also focus on replacing the current algorithm for ROI placement with a shallow neural network for region proposal.

ACKNOWLEDGMENT

(Aleksej Avramović and Davor Sluga contributed equally to this work.)

REFERENCES

[1] K. C. P. Wang, Z. Hou, and W. Gong, "Automated road sign inventory system based on stereo vision and tracking," *Comput.-Aided Civil Infrastruct. Eng.*, vol. 25, no. 6, pp. 468–477, Feb. 2010.

[2] S. Segvic, K. Brkic, Z. Kalafatic, V. Stanisavljevic, M. Sevrovic, D. Budimir, and I. Dadic, "A computer vision assisted geoinformation inventory for traffic infrastructure," in *Proc. 13th Int. IEEE Conf. Intell. Transp. Syst.*, Sep. 2010, pp. 66–73.

[3] V. Balali, A. Ashouri Rad, and M. Golparvar-Fard, "Detection, classification, and mapping of U.S. Traffic signs using Google street view images for roadway inventory management," *Vis. Eng.*, vol. 3, no. 1, p. 15, Nov. 2015.

[4] J. Greenhalgh and M. Mirmehdi, "Traffic sign recognition using MSER and random forests," in *Proc. 20th Eur. Signal Process. Conf. (EUSIPCO)*, Aug. 2012, pp. 1935–1939.

[5] J. M. Lillo-Castellano, I. Mora-Jiménez, C. Figuera-Pozuelo, and J. L. Rojo-Álvarez, "Traffic sign segmentation and classification using statistical learning methods," *Neurocomputing*, vol. 153, pp. 286–299, Apr. 2015.

[6] M. Haloi, "A novel pLSA based traffic signs classification system," 2015, *arXiv:1503.06643*. [Online]. Available: http://arxiv.org/abs/1503.06643

[7] A. Ellahyani, M. El, I. El, and S. Charfi, "Traffic sign detection and recognition using features combination and random forests," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 1, pp. 686–693, 2016.

[8] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. Computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Netw.*, vol. 32, pp. 323–332, Aug. 2012.

[9] F. Zaklouta and B. Stanciulescu, "Real-time traffic sign recognition in three stages," *Robot. Auto. Syst.*, vol. 62, no. 1, pp. 16–24, Jan. 2014.

[10] D. Nandi, A. S. Saif, P. Paul, K. M. Zubair, and S. A. Shubho, "Traffic sign detection based on color segmentation of obscure image candidates: A comprehensive study," *Int. J. Mod. Educ. Comput. Sci.*, vol. 10, no. 6, pp. 35–46, Jun. 2018.

[11] S. Aziz, E. A. Mohamed, and F. Youssef, "Traffic sign recognition based on multi-feature fusion and ELM classifier," *Procedia Comput. Sci.*, vol. 127, pp. 146–153, Jan. 2018.

[12] A. Mogelmose, M. M. Trivedi, and T. B. Moeslund, "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 4, pp. 1484–1497, Dec. 2012.

[13] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale convolutional networks," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2011, pp. 2809–2813.

[14] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," 2012, *arXiv:1202.2745*. [Online]. Available: http://arxiv.org/abs/1202.2745

[15] J. Jin, K. Fu, and C. Zhang, "Traffic sign recognition with hinge loss trained convolutional neural networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 5, pp. 1991–2000, Oct. 2014.

[16] L. Abdi and A. Meddeb, "Deep learning traffic sign detection, recognition and augmentation," in *Proc. Symp. Appl. Comput. (SAC)*, New York, NY, USA, 2017, pp. 131–136.

[17] A. Shustanov and P. Yakimov, "CNN design for real-time traffic sign recognition," *Procedia Eng.*, vol. 201, pp. 718–725, Jan. 2017.

[18] Y. Zhu, C. Zhang, D. Zhou, X. Wang, X. Bai, and W. Liu, "Traffic sign detection and recognition using fully convolutional network guided proposals," *Neurocomputing*, vol. 214, pp. 758–766, Nov. 2016.

[19] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu, "Traffic-sign detection and classification in the wild," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2110–2118.

[20] D. Tabernik and D. Skocaj, "Deep learning for large-scale traffic-sign detection and recognition," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 4, pp. 1427–1440, Apr. 2020.

[21] J. Zhang, M. Huang, X. Jin, and X. Li, "A real-time chinese traffic sign detection algorithm based on modified YOLOv2," *Algorithms*, vol. 10, no. 4, p. 127, Nov. 2017.

[22] A. Tran, D. Dien, H. Huynh, N. V. Long, and N. Tran, "A model for real-time traffic signs recognition based on the YOLO algorithm—A case study using vietnamese traffic signs," in *Future Data and Security Engineering* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2019.

[23] T. T. Le, S. T. Tran, S. Mita, and T. D. Nguyen, "Real time traffic sign detection using color and shape-based features," in *Intelligent Information and Database Systems*. Berlin, Germany: Springer, 2010, pp. 268–278.

[24] K. Kaplan, C. Kurtul, and H. Levent Akin, "Real-time traffic sign detection and classification method for intelligent vehicles," in *Proc. IEEE Int. Conf. Veh. Electron. Saf. (ICVES)*, Jul. 2012, pp. 448–453.

[25] Y. Yang, H. Luo, H. Xu, and F. Wu, "Towards real-time traffic sign detection and classification," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 7, pp. 2022–2031, Jul. 2016.

[26] F. Shao, X. Wang, F. Meng, T. Rui, D. Wang, and J. Tang, "Real-time traffic sign detection and recognition method based on simplified Gabor wavelets and CNNs," *Sensors*, vol. 18, no. 10, p. 3192, Sep. 2018.

[27] NVIDIA. (Aug. 2019). *DRIVE Labs: Classifying Traffic Signs and Traffic Lights With SignNet and LightNet DNNs*. [Online]. Available: https://news.developer.nvidia.com/drive-labs-signnet-and-lighnet-dnns/

[28] R. Timofte, K. Zimmermann, and L. Van Gool, "Multi-view traffic sign detection, recognition, and 3D localisation," *Mach. Vis. Appl.*, vol. 25, no. 3, pp. 633–647, Apr. 2014.

[29] S. Salti, A. Petrelli, F. Tombari, N. Fioraio, and L. Di Stefano, "Traffic sign detection via interest region extraction," *Pattern Recognit.*, vol. 48, no. 4, pp. 1039–1049, Apr. 2015.

[30] D. Temel, J. Lee, and G. Alregib, "CURE-OR: Challenging unreal and real environments for object recognition," in *Proc. 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2018, pp. 137–144.

[31] D. Temel, M.-H. Chen, and G. AlRegib, "Traffic sign detection under challenging conditions: A deeper look into performance variations and spectral characteristics," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 9, pp. 3663–3673, Sep. 2020.

[32] D. Temel, T. Alshawi, M.-H. Chen, and G. AlRegib, "Challenging environments for traffic sign detection: Reliability assessment under inclement conditions," 2019, *arXiv:1902.06857*. [Online]. Available: http://arxiv.org/abs/1902.06857

[33] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, Jan. 2017, pp. 2961–2969.

[34] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 936–944.

[35] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Computer Vision—ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 21–37.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[37] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7263–7271.

[38] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*. [Online]. Available: http://arxiv.org/abs/1804.02767

[39] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*. [Online]. Available: http://arxiv.org/abs/2004.10934

[40] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10781–10790.

[41] Z. Yang, S. Liu, H. Hu, L. Wang, and S. Lin, "RepPoints: Point set representation for object detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9656–9665.

[42] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "CenterNet: Keypoint triplets for object detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 6568–6577.

[43] H. Law, Y. Teng, O. Russakovsky, and J. Deng, "CornerNet-lite: Efficient keypoint based object detection," 2019, *arXiv:1904.08900*. [Online]. Available: http://arxiv.org/abs/1904.08900

[44] Z. Tian, C. Shen, H. Chen, and T. He, "FCOS: Fully convolutional one-stage object detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9626–9635.

[45] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End object detection with transformers," 2020, *arXiv:2005.12872*. [Online]. Available: http://arxiv.org/abs/2005.12872

[46] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.

[47] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

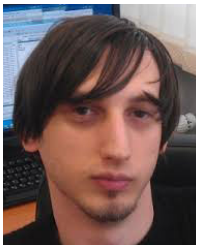
[48] K. Zuiderveld, *Contrast Limited Adaptive Histogram Equalization*. New York, NY, USA: Academic, 1994, pp. 474–485.

[49] P. Soille, *Morphological Image Analysis: Principles and Applications*, 2nd ed. New York, NY, USA: Springer-Verlag, 2003.

[50] G. Bradski, "The OpenCV library," *Dr. Dobb's J. Softw. Tools*, vol. 25, pp. 120–126, Nov. 2000.

ALEKSEJ AVRAMOVIĆ received the Ph.D. degree in electrical engineering and computer sciences from the University of Belgrade, Serbia, in 2016. He is currently an Associate Professor with the Faculty of Electrical Engineering, University of Banja Luka, Bosnia and Herzegovina. He is the Head of the Laboratory of Applied Electrical Engineering. His main research interests include machine learning, pattern recognition, circuit theory, and measurements.

DAVOR SLUGA received the B.Sc. and Ph.D. degrees in computer science from the University of Ljubljana, Slovenia, in 2010 and 2017, respectively. He currently holds the position of a Teaching Assistant at the Faculty of Computer and Information Science, University of Ljubljana. His research interests include high-performance computing, general-purpose GPU computing, and data mining.



DOMEN TABERNIK received the B.Sc. degree in computer and information science, in 2010. Since 2010, he has been working as a Computer Vision Researcher with the Visual Cognitive Systems Laboratory, Faculty of Computer and Information Science, University of Ljubljana. Since 2015, he has been enrolled with the Doctoral Program of the Faculty of Computer and Information Science, University of Ljubljana, where he is also working on the topics of compositional hierarchies and deep learning.

DANIJEL SKOČAJ (Member, IEEE) is currently an Associate Professor and the Head of the Visual Cognitive Systems Laboratory, Faculty of Computer and Information Science, University of Ljubljana. His main research interests include computer vision, pattern recognition, machine learning, and cognitive robotics. He has led a number of research projects from these research areas and facilitated the transfer of research findings into practical applications.

VLADAN STOJNIĆ (Member, IEEE) received the B.Sc. degree in electrical engineering from the Faculty of Electrical Engineering, University of Banja Luka, Bosnia and Herzegovina, in 2017, where he is currently pursuing the master's degree. His research interests include machine learning, computer vision, and self-supervised learning.

NEJC ILC (Member, IEEE) received the B.Sc. and Ph.D. degrees in computer science from the University of Ljubljana, Slovenia, in 2009 and 2016, respectively. He currently holds the position of a Teaching Assistant at the Faculty of Computer and Information Science, University of Ljubljana. His research interests include machine learning and data mining.