YOEO - You Only Encode Once: A CNN for Embedded Object Detection and Semantic Segmentation

Conference Paper · December 2021

DOI: 10.1109/ROBIO54168.2021.9739597

4 authors:

Florian Vahl
University of Hamburg
9 PUBLICATIONS **46** CITATIONS
SEE PROFILE

Jan Gutsche
University of Hamburg
9 PUBLICATIONS **33** CITATIONS
SEE PROFILE

Marc Bestmann
German Aerospace Center (DLR)
28 PUBLICATIONS **216** CITATIONS
SEE PROFILE

Jianwei Zhang
University of Hamburg
433 PUBLICATIONS **6,283** CITATIONS
SEE PROFILE

YOEO – You Only Encode Once: A CNN for Embedded Object Detection and Semantic Segmentation

Florian Vahl[1] and Jan Gutsche[1] and Marc Bestmann[1] and Jianwei Zhang[1]

***Abstract*— Fast and accurate visual perception utilizing a robot's limited hardware resources is necessary for many mobile robot applications. We are presenting YOEO, a novel hybrid CNN which unifies previous object detection and semantic segmentation approaches using one shared encoder backbone to increase performance and accuracy. We show that it outperforms previous approaches on the TORSO-21 and Cityscapes datasets.**

***Index Terms*— Computer vision, Robotics, Machine learning, Object recognition, Segmentation**

I. Introduction

Computer vision is an important tool in the field of robotics, as it produces highly detailed observations of the robot's environment. Especially in the context of the *RoboCup Humanoid Soccer League*, where the autonomous robots are restricted to only use sensors equivalent to human senses [1], the detection of objects and features does not only need to be robust and accurate but also be done in near real-time on the robot itself. Furthermore, the robots need to be able to make accurate detections even with changing natural light conditions in a dynamic environment.

Many *things*, such as the ball, other robots, or the goalposts, can sufficiently be described by bounding boxes. These include a localization on the image with a rough shape and a specification of the class. But such a representation yields problems for *stuff* classes like the floor or the field lines where a pixel-wise semantic segmentation describing the shape is more important than the instance abstraction of the bounding box. In many other domains that involve mobile robots, for example automated delivery systems, a similar need to perform these two types of detection in near real-time using limited hardware resources can be expected. CNN-based approaches such as YOLOv4 [2] for object detection and U-NET [3] for semantic segmentation dominate the field of computer vision in recent years. Using separate approaches for both the object detection and the semantic segmentation leads to a computational overhead since both networks feature their own encoder and similar features are encoded twice.

Therefore, it is reasonable to share one encoder backbone between multiple decoders generating outputs for different purposes. Additionally, this helps with generalization because the features are needed in different contexts. With this in mind, we designed a novel neural network architecture, YOEO (You Only Encode Once, pronounced ['jojo]), that provides both, bounding box instance detections and pixel-precise segmentations while minimizing computational overhead by using a shared encoder backbone. The model was optimized using the *OpenVino*™ toolkit and deployed on an *Intel® Neural Compute Stick 2* (NCS2) *vision processing unit* (VPU) used in our Wolfgang-OP robot [4].

II. Related Work

Starting with AlexNet [5] in 2012, approaches using CNNs in the context of computer vision grew in popularity. Outside of simple classification tasks, CNN architectures like YOLO [6] and SSD [7] gained in influence. Both predict bounding boxes for the detected objects. This allows for easy differentiation and classification for multiple objects at the same time.

The field of semantic segmentation is also dominated by CNNs nowadays. Networks like FCN [8] or U-NET [3] use this approach to generate promising results for pixel-precise classifications.

Using a shared encoder backbone for both object detection and semantic segmentation has been proposed by Teichmann et al. in 2018 [9]. Their approach features bounding box object detection, image classification, and semantic segmentation. They mainly focus on the field of autonomous driving,

This research was partially funded by the German Research Foundation (DFG) and the National Science Foundation of China (NSFC) in project Crossmodal Learning, TRR-169.

[1]All the authors are with the Department of Informatics, University of Hamburg, 22527 Hamburg, Germany [florian.vahl, jan.gutsche, marc.bestmann, jianwei.zhang]@uni-hamburg.de

generating predictions for vehicle detection, street scenery classification, and road segmentation at the same time. The findings are promising, but the overall model size is too large for a real-time inference on an embedded device and the image classification part is not needed for our use case. Furthermore, the object detection decoder architecture is quite different from the YOLO object detection which is state of the art in the RoboCup humanoid soccer domain.

A combined approach that is based on an older version of the YOLO architecture was developed by the automotive supplier *Valeo* and is described in *Real-time Joint Object Detection and Semantic Segmentation Network for Automated Driving* [10]. It uses a combination of a ResNet10-like encoder with a YOLOv2- and FCN8-like decoder but is not very detailed when it comes to the implementation and deployment details.

The field of panoptic segmentation features models like Panoptic FPN [11] to predict both, instance and semantic segmentation. This allows for pixel-precise instance segmentation, by removing the abstraction of the bounding box. While seeming very promising it also requires an expensive pixel-precise annotation for all classes including ones where the abstraction of a bounding box is sufficient. A pixel-precise prediction for these classes would also waste resources during the inference.

III. Approaches

We trained and evaluated multiple architectures to construct a model that fits our needs. All of the presented architectures feature two YOLO detection heads at different resolutions and a U-NET-like decoder which share a common encoder backbone. They are also fully convolutional, meaning they do not use any fully connected layers and are easily scalable to different resolutions. The encoder and YOLO heads are taken from the YOLOv3 or YOLOv4 architectures. The decoder consists of a U-NET-like topology meaning feature maps of the encoder are upsampled multiple times using nearest-neighbor interpolation. Each time the upsampled feature map is concatenated with the corresponding feature map in the encoder branch using a skip connection and a convolutional layer. This layer is also used to reduce the number of channels and convert the features to a more spatial topology.

We used the *TORSO-21 Dataset* [12] to train and evaluate different variants of the network. The dataset is used because it resembles the intended deployment domain of the architecture. It features many images with bounding boxes as well as segmentation classes. The images were downscaled to an input resolution of 416 by 416 pixels which is also the output resolution of the segmentation head. Data augmentation in the form of random changes to the sharpness, brightness, and hue of the image as well as random vertical mirroring, translation (max 10% of the image size), and scaling (80% to 150%) has been applied.

The network parameters were optimized using the ADAM optimizer [13]. The loss function consists of the sum of a YOLOv4 loss for object detection and a cross-entropy loss for semantic segmentation. Both are weighted equally.

Our open source reference implementation[1] is a fork of PyTorch-YOLOv3[2] which served as the starting point of our software architecture.

While trying to find the optimal architecture for our use case, we investigated the following architecture layouts:

- A YOLOv3-tiny for the encoder and bounding box prediction. A U-NET decoder is connected with all four feature maps down to a resolution of 52x52 and includes three convolutional blocks (3x3 convolution, batch normalization, and ReLU) before the output. The model served as a baseline for further experiments. (*YOEO-rev-0*)
- A YOLOv4-tiny for the encoder and bounding box prediction. YOLOv4-tiny uses a larger stride instead of maxpooling in the first layers so the output of the first convolution is already downsampled by a factor of two. So we tested if we should
  - add a convolution with a stride of one in front of it and start the native resolution skip connection here. (*YOEO-rev-1*)
  - use the downsampled feature map as the first skip connection. (*YOEO-rev-2*)
- A segmentation decoder head with one or two convolutions after the last upsampling. (*YOEO-rev-3*, *YOEO-rev-4* respectively)
- A segmentation decoder head with a 1x1 convolution instead of the 3x3 after the last upsampling. (*YOEO-rev-5*)
- A segmentation decoder which starts with the deepest feature map in the encoder. (*YOEO-rev-6*)

[1] `github.com/bit-bots/YOEO`
[2] `github.com/eriklindernoren/PyTorch-YOLOv3`

- A variant of YOEO-rev-2 which starts with the deepest feature map in the encoder. (*YOEO-rev-7*)
- A segmentation decoder head with residual skip connections after the last upsampling. (*YOEO-rev-8*)
- A segmentation decoder that starts with the deepest feature map in the encoder and has only one high-resolution skip connection to gain a performance benefit. (*YOEO-rev-9*)

The complete model architectures are also provided in the repository of the reference implementation[1].

In the end, we settled for a YOLOv4-tiny with a segmentation decoder that uses encoder feature maps from all resolutions except the native resolution (YOEO-rev-7). This architecture is shown in figure 1. In the selection, we considered the object detection and segmentation performance as well as the overall runtime of the network. The detailed evaluation of the different architectures can be seen in section IV.

This final architecture was converted to the ONNX format. The model was then optimized by the model optimizer of the OpenVINO toolkit and converted to the OpenVINO toolkit's intermediate representation format, which itself was used to deploy the network on an Intel NCS2 VPU on the Wolfgang-OP robot. This allows for efficient inference of the network which is crucial in our power, weight, and performance-limited use case.

IV. Evaluation

In the following, we will compare the different revisions of our YOEO model as well as YOLOv4-tiny and U-NET regarding their precision and runtime on two different datasets and domains.

A. Metrics

To quantify the precision of the segmentation images, we have selected the common *Jaccard index* also known as *Intersection over Union* (IoU) metric. For the evaluation of the bounding box object detection, the *mean average precision* with a minimum of 50% IoU (mAP50) was used. The inference speed of the models was evaluated using the *frames per second* (FPS) measure, running either on an NVIDIA® GeForce® RTX 2080 Ti or with model optimizations on the Intel NCS2 VPU ready for deployment.

B. Datasets

During our evaluation phase, we have used two datasets for training and evaluation. As it perfectly

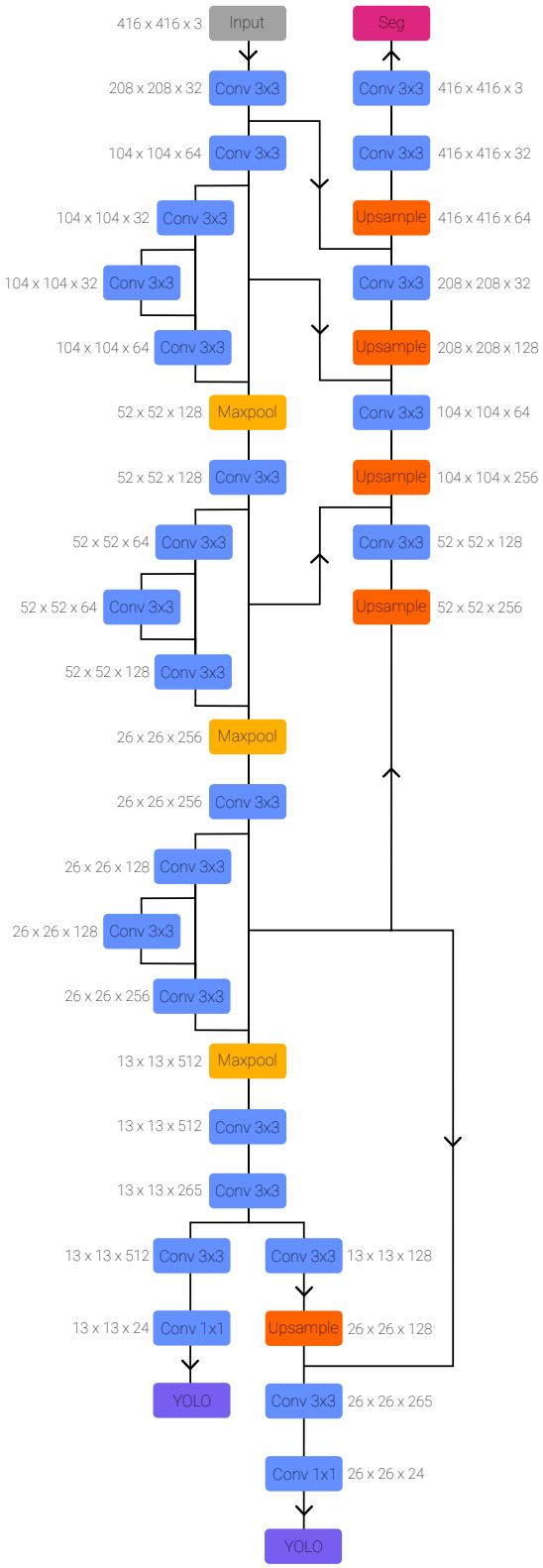

Fig. 1. Layout of the final YOEO architecture (YOEO-rev-7). The shape of the output feature map is noted next to the layers. Maxpooling layers are marked in orange while nearest-neighbor upsampling is red. The convolutional layers (blue) also include batch normalization and a leaky ReLU activation function. The number of filters before the output layers (purple, pink) depend on the number of predicted classes, this figure shows the layout for the TORSO-21 dataset with three bounding box and three segmentation classes.

matches our use case, we have used the *TORSO-21 Dataset* [12] from the RoboCup soccer domain to train, test, and select from the various approaches mentioned in section III. Additionally, we have used the *Cityscapes Dataset* [14] to evaluate our final model architecture and to quantitatively compare it with other architectures.

The TORSO-21 dataset consists of two image collections, one contains images and labels of a simulated soccer environment, the other is a diverse dataset of 8894 train and 1570 test images respectively. The images were recorded by five different camera types from twelve locations containing six ball types. Classes included in this collection are ball, robot, and L-, T-, and X-intersections of the soccer field lines as bounding box labels, and goalpost labels as four-point polygons. Additionally, it provides segmentation masks for *stuff* classes like the field area and lines. We converted the goalpost polygons to bounding boxes as required by the YOLO architecture. We chose to not make use of the line intersection labels, as we want to use the semantically segmented lines, which contain more information. As the images do not share a common resolution or aspect ratio, they get scaled to match 416 pixels in the largest axis and padded with black on the other axis. This results in a 416 by 416 pixel square image as described above.

The Cityscapes dataset is commonly used as a benchmark for vision algorithms, especially in the autonomous driving domain. It contains several collections from stereo video recordings of street scenes from 50 different cities. We have used the *gtFine* collection consisting of 5,000 images with pixel-level instance and semantic segmentation. 30 classes are labeled that can be combined into eight groups. As our initial problem space does not require as many classes, we have decided to train and evaluate our model architecture only based on the groups for the semantic segmentation. For object detection, we used all classes that supported instance segmentations. As there are only eight of them, no grouping was applied. The instance segmentations were converted to bounding boxes which are compatible with the predictions of our YOLO decoder. For this conversion, we have utilized a tool[3] by Till Beemelmanns. Instead of applying padding to the input images, we scaled the rather wide but consistent resolution of 1024 x 2048 pixels directly to a 416 by 416 pixel square.

[3] `github.com/TillBeemelmanns/cityscapes-to-coco-conversion`

TABLE I
PERFORMANCE OF THE EVALUATED ARCHITECTURES ON THE TORSO-21 TEST DATASET.

Architecture	mAP50	IoU	FPS
YOEO-rev-0	78.87%	82.62%	175.45
YOEO-rev-1	**84.10%**	83.37%	133.96
YOEO-rev-2	83.87%	83.49%	138.76
YOEO-rev-3	77.91%	82.22%	**185.78**
YOEO-rev-4	77.97%	82.21%	180.64
YOEO-rev-5	78.71%	82.27%	179.59
YOEO-rev-6	78.95%	**85.28%**	154.36
YOEO-rev-7	83.36%	85.02%	137.14
YOEO-rev-8	78.62%	82.53%	173.24
YOEO-rev-9	79.12%	84.23%	163.40
YOLOv4-tiny	83.56%	-	174.67
U-NET	-	81.58%	152.07

C. Architecture Comparison

The different proposed architecture layouts have been trained for up to 60 epochs on the TORSO-21 dataset. The results are presented in table I. The performance of the object detection is measured using the mAP50 metric, while the IoU is used for semantic segmentation. The frames per second (FPS) are measured on an NVIDIA® GeForce® RTX 2080 Ti and do not include the inference optimizations applied to the deployed model. Standard deviations for the FPS measurements are negligibly small. The reference U-NET model uses a full YOLOv4-tiny encoder with skip connections on every level (similar to the segmentation decoder of YOEO-rev-7).

It can be seen that the YOLOv4-tiny based models have a clear advantage over the YOLOv3-tiny ones when it comes to object detection performance. We, therefore, strongly prefer a YOLOv4-tiny network. Also, it is evident that a deeper starting segmentation decoder results in a higher IoU. The change from normal (YOEO-rev-0) to deeper (YOEO-rev-6) increases the IoU from 82.62% to 85.28%. While this change is relatively small, it is still significant for classes like the field where the majority is easy to detect but the interesting edge cases are only solved by the deeper model. This includes cases where other intentionally unlabeled soccer fields are visible in the background, images with natural light (areas might be over- or underexposed), or cases where there are obstructions (people, robots, cables, laptops, ...) on the field that are part of the field class due to the simplified annotation method for field annotations described in the TORSO-21 paper [12]. Deeper encoder feature maps help in this regard, because they are less spatially specific, allowing to focus on a larger context while

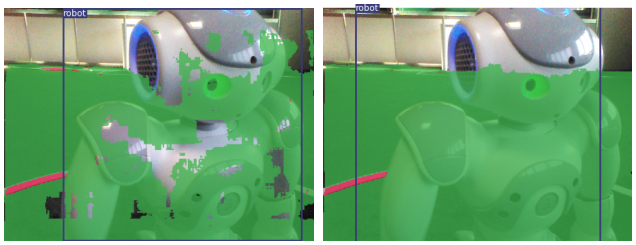

Fig. 2. This figure shows an exemplary difference between the segmentation of the YOEO-rev-0 (left) and YOEO-rev-6 (right) segmentation decoder. It shows how the deeper decoder of YOEO-rev-6 handles large occlusions of the *field* (**green**) better. It also correctly classifies the bright area lit by the sun in the upper left of the image as field instead of *line* (**red**).

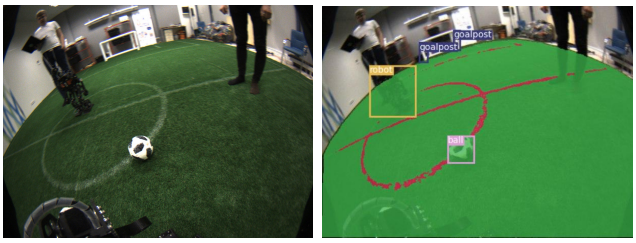

Fig. 3. Visualization of the predictions (right) of the final model on a test image (left) from the TORSO-21 dataset. The prediction includes the successful detection of another *robot* (**yellow**), the *goalposts* (**blue**), and the *ball* (**pink**) while ignoring the robot itself and the humans on the field as intended. Further, an accurate segmentation of *field* (**green**) and *lines* (**red**) can be seen.

also having more non-linearities which allow for an approximation of more complex data distributions. This is demonstrated in figure 2.

The experiments also show that the difference between one and two convolutional blocks after the last upscaling in the segmentation decoder head is mainly a longer runtime than any performance benefit for models with a full resolution skip connection. Using a 1x1 convolution in the last layer has nearly no effect despite impacting the performance (see YOEO-rev-5).

The residual connections in the segmentation decoder head (YOEO-rev-8) made no notable difference in the accuracy while slightly slowing down the model during inference. The improved gradient flow resulted in faster training times, but it was not significant enough to justify the slower inference.

Using the encoder layout from YOEO-rev-2 which does not include a full-resolution skip connection over YOEO-rev-1 seems reasonable because the accuracy trade-off is negligible and the lower resolution skip-connection results in a faster inference with a smaller memory footprint. It also enables the usage of pre-trained YOLOv4-tiny encoder weights that are available for many datasets.

The combined YOEO network outperforms the standalone versions of both the YOLOv4-tiny and U-NET even while sharing the encoder. This seems to be the case because the different domains of both decoders help the encoder to generalize better during training. The combined encoder also results in an inference speed benefit of 68.70% when we compare the combined runtime of the individual networks with the YOEO-rev-7 architecture as seen in table I.

Therefore, it is reasonable to settle for a YOLOv4-tiny based architecture that utilizes the full encoder for the semantic segmentation without using a full resolution skip connection or residuals in the decoder (YOEO-rev-7). It is also the best out of the YOLOv4-tiny models when ranking them using the geometric mean of the evaluated attributes including the runtime.

This final model has been deployed as described in section III. The model runs at 6.7 FPS with less than 2 Watt power usage on the embedded hardware of the robot.

D. Performance on Cityscapes

We also tested our architecture on the Cityscapes dataset. The results can be seen in table II. The dataset is slightly out of domain for this architecture and it can be seen that the YOLOv4-tiny model struggles with the high number of small objects in the dataset. This is typical for this family of object detectors [15]. With these limitations in mind, our approach still outperforms the similar YOLO-based approach from Valeo [10] in all tested categories on the dataset while being comparable in size and speed (see table II. A complete runtime comparison was not possible due to a lack of information regarding their implementation and deployment. An exemplary prediction is shown in figure 4.

Fig. 4. Predictions from YOEO on the Cityscapes dataset. *Stuff* segmentation classes: *flat* (**red**), *construction* (**green**), *object* (**yellow**), *nature* (**blue**), *sky* (**orange**), and *void* (**transparent**). Visible *things*: *cars* (**brown**), *trucks* (**pink**), and *persons* (**blue**).

TABLE II
PERFORMANCE OF THE FINAL ARCHITECTURE ON THE CITYSCAPES TEST DATASET.

Metric	Class	YOEO-rev-7	MTL Valeo [10]
AP	Person	**26.87%**	19.31%
AP	Rider	29.97%	-
AP	Bicycle	**27.51%**	18.98%
AP	Car	**51.05%**	41.10%
AP	Bus	29.88%	-
AP	Motorcycle	03.29%	-
AP	Truck	14.52%	-
AP	Train	08.11%	-
IoU	Flat (e.g. Road)	**91.76%**	59.66%
IoU	Construction	**81.43%**	63.63%
IoU	Object (e.g. Poles)	40.99%	-
IoU	Nature	**85.45%**	69.49%
IoU	Sky	**86.08%**	62.28%

V. CONCLUSION & FUTURE WORK

With YOEO we present a hybrid CNN approach based on YOLOv4-tiny for object detections and U-NET for semantic segmentation utilizing only one encoder backbone. Our architecture outperforms both standalone versions in precision indicating better generalization of the encoder while simultaneously gaining a 68.70% speed benefit over running both standalone architectures sequentially. Many robotics computer vision tasks could benefit from this hybrid approach.

Training the YOEO architecture requires a diverse dataset containing *stuff* and *things* classes, which is more expensive to create than a dataset just containing bounding boxes. On the other hand it is less expensive to label than a fully annotated panoptic segmentation dataset. It is also possible to convert datasets containing panoptic segmentations as shown with the Cityscapes dataset.

The YOEO architecture could be improved further by using automated parameter optimization for the hyperparameters. Additionally, a classification decoder head could be appended to the shared encoder to provide classifications of e.g. weather, game states, or perturbations such as glare or staining of the camera. Combining other segmentation architectures such as *BiSeNet V2* [16] could also improve the results.

ACKNOWLEDGMENT

Thanks to Erik Linder-Norén for the original *PyTorch-YOLOv3* implementation. Also thanks to the members of the Hamburg Bit-Bots for helping to develop this architecture.

REFERENCES

[1] "RoboCup Soccer Humanoid League Laws of the Game 2019/2020," humanoid.robocup.org/wp-content/uploads/RCHL-2020-Rules-Dec23.pdf, (accessed November 21, 2020).

[2] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *preprint arXiv:2004.10934*, 2020.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015.

[4] M. Bestmann, J. Güldenstein, F. Vahl, and J. Zhang, "Wolfgang-OP: A Robust Humanoid Robot Platform for Research and Competitions," in *IEEE Humanoids 2021*, 07 2021.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems*, vol. 25, 2012.

[6] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *preprint arXiv:1506.02640*, 2015.

[7] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot Multibox Detector," in *European conference on computer vision*. Springer, 2016.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[9] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun, "Multinet: Real-Time Joint Semantic Reasoning for Autonomous Driving," in *Intelligent Vehicles Symposium (IV)*. IEEE, 2018.

[10] G. Sistu, I. Leang, and S. Yogamani, "Real-Time Joint Object Detection and Semantic Segmentation Network for Automated Driving," *preprint arXiv:1901.03912*, 2019.

[11] A. Kirillov, R. Girshick, K. He, and P. Dollár, "Panoptic Feature Pyramid Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6399–6408.

[12] M. Bestmann, T. Engelke, N. Fiedler, J. Güldenstein, J. Gutsche, J. Hagge, and F. Vahl, "TORSO-21 Dataset: Typical Objects in RoboCup Soccer 2021," in *RoboCup 2021*, 2021.

[13] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR*, 2015.

[14] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] N.-D. Nguyen, T. Do, T. D. Ngo, and D.-D. Le, "An Evaluation of Deep Learning Methods for Small Object Detection," *Journal of Electrical and Computer Engineering*, 2020.

[16] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, "Bisenet v2: Bilateral Network With Guided Aggregation for Real-Time Semantic Segmentation," *preprint arXiv:2004.02147*, 2020.