Neural Computing and Applications (2022) 34:8275–8290
https://doi.org/10.1007/s00521-021-05978-9

ORIGINAL ARTICLE

Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3

Petr Hurtik[1] · Vojtech Molek[1] · Jan Hula[1] · Marek Vajgl[1] · Pavel Vlasanek[1] · Tomas Nejezchleba[2]

Received: 12 November 2020 / Accepted: 25 March 2021 / Published online: 19 February 2022

Abstract
We present a new version of YOLO with better performance and extended with instance segmentation called Poly-YOLO. Poly-YOLO builds on the original ideas of YOLOv3 and removes two of its weaknesses: a large amount of rewritten labels and an inefficient distribution of anchors. Poly-YOLO reduces the issues by aggregating features from a light SE-Darknet-53 backbone with a hypercolumn technique, using stairstep upsampling, and produces a single scale output with high resolution. In comparison with YOLOv3, Poly-YOLO has only 60% of its trainable parameters but improves the mean average precision by a relative 40%. We also present Poly-YOLO lite with fewer parameters and a lower output resolution. It has the same precision as YOLOv3, but it is three times smaller and twice as fast, thus suitable for embedded devices. Finally, Poly-YOLO performs instance segmentation by bounding polygons. The network is trained to detect size-independent polygons defined on a polar grid. Vertices of each polygon are being predicted with their confidence, and therefore, Poly-YOLO produces polygons with a varying number of vertices. Source code is available at https://gitlab.com/irafm-ai/poly-yolo.

Keywords Object detection · Instance segmentation · YOLOv3 · Bounding box · Bounding polygon · Real-time detection

1 Problem statement

Object detection is a process where all important areas containing objects of interest are bounded, while the background is ignored. Usually, the object is bounded by a box that is expressed in terms of the spatial coordinates of its top-left corner and its width and height. The disadvantage of this approach is that for the objects of complex shapes, the bounding box also includes background data, which can occupy a significant part of the area as the bounding box does not wrap the object tightly. Such behavior can decrease the performance of a classifier applied over the bounding box [1] or may not fulfill the requirements of precise detection [2]. To avoid the problem, classical detectors such as Faster R-CNN [3] or RetinaNet [4] were modified into a version of Mask R-CNN [5] or RetinaMask [6]. These methods also infer the instance segmentation, i.e., each pixel in the bounding box is classified into object/background classes. The limitation of the methods is their computation speed, where they are unable to reach real-time performance on non-high-tier hardware. The problem we focus on is to create a precise detector with instance segmentation and the ability of real-time processing on mid-tier graphics cards. By "mid-tier,"' we mean NVIDIA 2O and 3O graphics cards, while the high-tier is presented by V100/A100, Titans, or TPUs.

In this study, we start with YOLOv3 [7], which excels in processing speed, and therefore, it is a good candidate for real-time applications running on computers [8] or mobile devices [9]. On the other hand, the precision of YOLOv3 lags behind detectors such as RetinaNet [4], EfficientDet [10], or CornerNet [11]. We analyze YOLO's performance and identify its two drawbacks. The first drawback is the low precision of the detection of big boxes [7] caused by inappropriate handling of anchors in the output layers. The

✉ Petr Hurtik
petr.hurtik@osu.cz

1 Centre of Excellence IT4Innovations, Institute for Research and Applications of Fuzzy Modeling, University of Ostrava, 30. dubna 22, Ostrava, Czech Republic

2 Varroc Lighting Systems, Suvorovova 195, Šenov u Nového Jičína, Czech Republic

second one is rewriting of labels by each other due to the coarse resolution. To solve these issues, we design a new approach, dubbed Poly-YOLO, that significantly pushes forward the original YOLOv3 capabilities. To tackle the problem of instance segmentation, we propose a way to detect tight polygon-based contours; see (Figs. 1, 2, 3) illustrating the output of Poly-YOLO. Our contributions and benefits of our approach are as follows:

- We propose Poly-YOLO that increases the detection accuracy of the previous version, YOLOv3. Poly-YOLO has a brand-new feature decoder with a single output tensor that goes to a head with a higher resolution that solves two principal YOLO's issues: rewriting of labels and incorrect distribution of anchors.
- We produce a single output tensor by a hypercolumn composition of multi-resolution feature maps produced by a feature extractor. To unify the resolution of the feature maps, we utilize stairstep upscaling, which allows us to obtain a slightly lower loss in comparison with direct upscaling while the computation speed is preserved.
- We design an extension that realizes instance segmentation using bounding polygon representation. The number of maximal polygon vertices can be adjusted according to the requirement of precision. The system accepts labels with an arbitrary number of vertices, even over the defined maximum.
- The bounding polygon is detected within a polar grid with relative coordinates that allow the network to learn general, size-independent shapes. The network produces a dynamic number of vertices per bounding polygon up to the maximum defined vertices.

2 Current state and related work

2.1 Object detection

Models for object detection can be divided into two groups: two-stage and one-stage detectors. Two-stage detectors split the process as follows. In the first phase, regions of interest (RoI) are proposed, and in the subsequent stage, bounding box regression and classification is being done inside these proposed regions. One-stage detectors predict the bounding boxes and their classes at once. Two-stage detectors are usually more precise in terms of localization and classification accuracy, but in terms of processing are slower than one-stage detectors. Both of these types contain a backbone network for feature extraction and head networks for classification and regression. Typically, the backbone is some SOTA network such as ResNet [5] or ResNext [12], pre-trained on ImageNet or OpenImages. Even, some approaches [13, 14] also experiment with training from scratch.

2.1.1 Two-stage detectors

The prototypical example of two-stage architecture is Faster R-CNN [3], which is an improvement of its predecessor Fast R-CNN [15]. The main improvement lies in the use of region proposal network (RPN), which replaced a much slower selective search of RoIs. It also introduced the usage of multi-scale anchors to detect objects of different sizes. Faster R-CNN is, in a way, a meta-algorithm that can have many different incarnations depending on a type of the backbone and its heads. One of the frequently used backbones, called feature pyramid network (FPN) [16], allows to predict RoIs from multiple feature maps, each with a different resolution. This is beneficial for the recognition of objects at different scales.

2.1.2 One-stage detectors

Two best-known examples of one-stage detectors are YOLO [7] and SSD [17]. The architecture of YOLO will be thoroughly described in Sect. 3. Usually, one-stage detectors divide the image into a grid and predict bounding boxes and their classes inside them, all at once. Most of them also use the concept of anchors, which are predefined typical dimensions of bounding boxes that serve as a priori knowledge. One of the major improvements in the area of one-stage detectors was a novel loss function call Focal Loss [4]. Because of the fact that two-stage detectors produce a sparse set of region proposals in the first step, most of the negative locations are filtered out for the second stage. One-stage detectors, on the other hand, produce a dense set of region proposals which they need to classify as containing objects or not. This creates a problem with the non-proportional frequency of negative examples. Focal loss solves this problem by adjusting the importance of negative and positive examples within the loss function. Another interesting idea was proposed in an architecture called RefineDet [18], which performs a two-step regression of the bounding boxes. The second step refines the

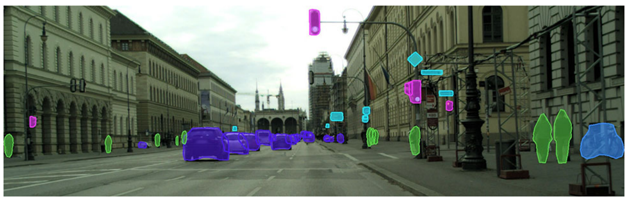


Fig. 1 The figure shows instance segmentation performance of the proposed Poly-YOLO algorithm applied on Cityscapes dataset and running 22FPS on a mid-tier graphic card. Image was cropped due to visibility

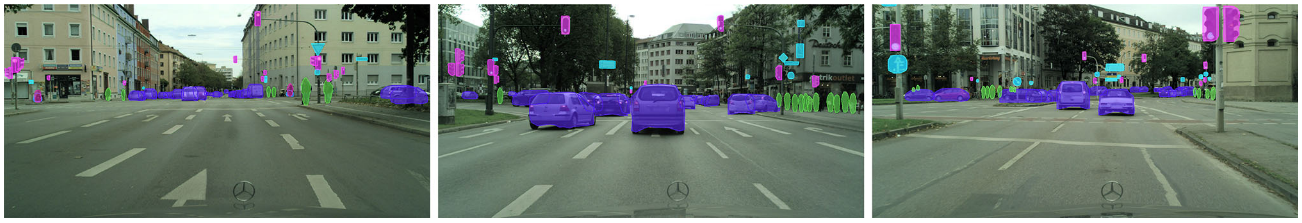


Fig. 2 Examples of Poly-YOLO inference on the Cityscapes testing dataset

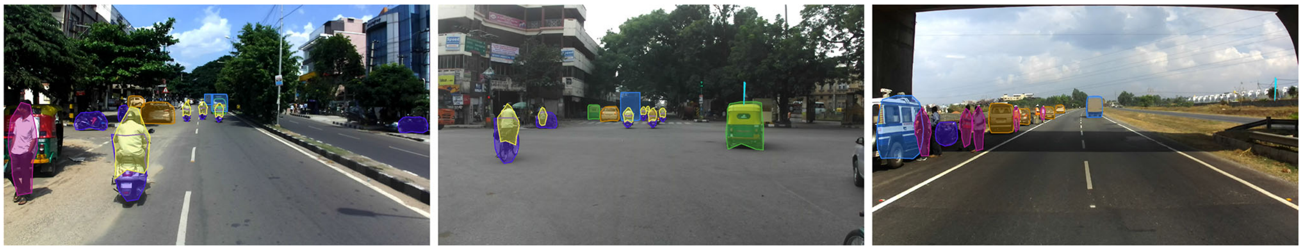


Fig. 3 Examples of Poly-YOLO inference on the India driving testing dataset

bounding boxes proposed in the first step, which produces more accurate detection, especially for small objects. Recently, there has been a surge of interest in approaches that do not use anchor boxes. The main representative of this trend is the FCOS framework [19], which works by predicting four coordinates of a bounding box for every foreground pixel. These four coordinates represent a distance to the four boundary edges of a bounding box in which the pixel is enclosed in. The predicted bounding boxes of every pixel are subsequently filtered by NMS. Similar anchor-free approach was proposed in CornerNet [11], where the objects are detected as a pair of top-left and bottom-right corners of a bounding box.

2.2 Instance segmentation

In many applications, a boundary given by a rectangle may be too crude, and we may instead require a boundary framing the object tightly. In the literature, this task is called instance segmentation, and the main approaches also fit into the one-stage/two-stage taxonomy. The prototypical example of a two-stage method is an architecture called Mask R-CNN [5], which extended Faster R-CNN by adding a separate fully convolutional head that predicts masks of objects. Note the same principle is also applied to RetinaNet, and the improved net is called RetinaMask [6]. One of Mask R-CNN innovations is a novel way for extracting features from RoIs using the RoIAlign layer, which avoids the problem of misalignments of the RoI due to its quantization to the grid of the feature map. One-stage methods, for instance, segmentation can be further divided into top-down methods, bottom-up methods and direct methods. Top-down methods [20, 21] work by first detecting an object and then segmenting this object within a bounding box. Prediction of bounding boxes either uses anchors or is anchor free following the FCOS framework [19]. Bottom-up methods [22, 23], on the other hand, work by first embedding each pixel into a metric space in which are these pixels subsequently clustered. As the name suggests, direct methods work by directly predicting the segmentation mask without bounding boxes or pixel embedding [24]. We also mention that independently of our instance segmentation, PolarMask [25] introduces instance segmentation using polygons, which are also predicted in polar coordinates. In comparison with PolarMask, Poly-YOLO learns itself in general size-independent shapes due to the use of the relative size of a bounding polygon according to the particular bounding box. The second difference is that Poly-YOLO produces a dynamic number of vertices per polygon, according to the shape-complexity of various objects.

3 Fast and precise object detection with Poly-YOLO

Here, we firstly recall YOLOv3 fundamental ideas, describe issues that block reaching higher performance and propose our solution that removes them.

3.1 YOLO history

First version of YOLO (You Only Look Once) was introduced in 2016 [26]. The motivation behind YOLO is to create a fast object detector with an emphasis on speed. The detector is made of two essential parts: the convolutional neural network (CNN) and a specially designed loss function. The CNN backbone is inspired by GoogleNet [27] and has 24 convolutional layers followed by 2 fully connected layers. The network output is reshaped into a two-dimensional *grid* with the shape of $G^h \times G^w$, where G^h is the number of cells in the vertical side and G^w in the horizontal side. Each grid cell occupies a part of the image, as depicted in Fig. 4. Every object in the image has its center in one of the cells, and that particular cell is responsible for detecting and classifying the said object. More precisely, the responsible cell outputs N^B bounding boxes. Each box is given as a tuple (x, y, w, h) and a confidence measure. Here, (x, y) is the center of the predicted box relative to the cell boundary and (w, h) is the width and height of the bounding box relative to the image size. The confidence measures how much is the cell confident that it contains an object. Finally, each cell outputs N^c conditional class probabilities, i.e., the probabilities that the detected object belongs to a certain class(es). In other words, cell confidence tells us that there is object in the predicted box and the conditional class probabilities tell us that the box contains, e.g., a vehicle—car. The final output of the model is a tensor with dimensions $G^h \times G^w \times (5N^B + N^c)$, where constant five is used because of (x, y, w, h) and a confidence.

YOLOv2 [28] brought a couple of improvements. Firstly, the architecture of the convolutional neural network was updated to Darknet-19—a fully convolutional network with 19 convolutional layers containing batch normalization and five max-pooling layers. The cells are no longer predicting the plain (x, y, w, h) directly, but also scales and translates *anchor boxes*. The parameters (a^w, a^h), i.e., the width and height of an anchor box for all anchor boxes, are extracted from a training dataset with the usage of k-means algorithm. The clustering criterion is IoU. Lastly, YOLOv2 uses skip connections to concatenate features from different parts of the CNN to create a final tensor of feature maps, including features across different scales and levels of abstraction.

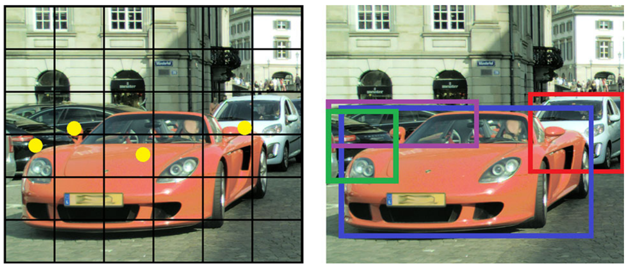


Fig. 4 The left image illustrates the YOLO grid over the input image, and yellow dots represent centers of detected objects. The right image illustrates detections

The most recent version of YOLO [7] (YOLOv3) introduces mainly three output scales and a deeper architecture—Darknet-53. Each of the scales has its own set of anchors—three per output scale. Compared with v2, YOLOv3 reaches higher accuracy, but due to the heavier backbone, its inference speed is decreased.

3.2 YOLOv3 issues blocking better performance

YOLOv3, as it is designed, suffers from two issues that we discovered and that are not described in the original YOLOv2 and YOLOv3 papers: rewriting of labels and imbalanced distribution of anchors across output scales. Solving these issues is crucial for the improvement of the YOLO performance.

3.2.1 Label rewriting problem

Here, we discuss the situation, when a bounding box given by its label from a ground truth dataset can be rewritten by other box and therefore the network is not trained to detect it. For the sake of simplicity and explanation, we avoid the usage of the anchor's notation in the text below. Let us suppose there is an input image with a resolution of $r \times r$ pixels. Furthermore, let s_k be the scale ratio of the k-th output to the input, where YOLOv3 uses the following ratios: $s_1 = 1/8, s_2 = 1/16, s_3 = 1/32$. These scales are given by the YOLOv3 architecture, namely, by strided convolution. Finally, let $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ be a set of boxes presented in an image. Each box $\mathbf{b}_i$ is represented as a tuple $(b_i^{x^1}, b_i^{y^1}, b_i^{x^2}, b_i^{y^2})$ that defines its top-left and bottom-right corners. For simplicity, we also derive the centers $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_n\}$ where $\mathbf{c}_i = (c_i^x, c_i^y)$ is defined as $c_i^x = 0.5(b_i^{x^1} + b_i^{x^2})$ and the same for c_i^y. With this notation, a label is rewritten, if the following holds:

$$\exists(\mathbf{c}_i, \mathbf{c}_j \in C) : \xi(c_i^x, c_j^x, s_k) + \xi(c_i^y, c_j^y, s_k) = 2, \tag{1}$$

where

$$\xi(x, y, z) = \begin{cases} 1, \lfloor xz \rfloor = \lfloor yz \rfloor \\ 0, \text{else} \end{cases}, \tag{2}$$

and $\lfloor \cdot \rfloor$ denotes the lowest integer of the term. The purpose of the function ξ is to check if both boxes are assigned to the same cell of a grid on the scale s_k. In simple words, if two boxes with the same scale are assigned to the same cell, then one of them will be rewritten. Introducing anchors, both must belong to the same anchor. As a consequence, the network is trained to ignore some objects, which leads to a low number of positive detections. According to Eqs. (1) and (2), there is a crucial role of s_k that directly affects the number and resolution of cells. Considering the standard resolution of YOLO $r = 416$, then, for s_3 (the coarsest scale) we obtain a grid of 13×13 cells with the size of 32×32 pixels each. Moreover, the absolute size of boxes does not affect the label rewriting problem; the important indicator is the box center. The practical illustration for such a setting and its consequences for the labels is shown in Fig. 5. The ratio of rewritten labels in the datasets used in the benchmark is shown in Table 1.

3.2.2 Anchors distribution problem

The second YOLO issue comes from the fact that YOLO is anchor-based (i.e., it needs prototypical anchor boxes for training/detection), and the anchors are distributed among output scales. Namely, YOLOv3 uses nine anchors, three per output scale. A particular ground truth box is matched with the best matching anchor that assigns it to a certain output layer scale. Here, let us suppose a set of box sizes $M = \{\mathbf{m}_1, \ldots, \mathbf{m}_n\}$, where $\mathbf{m}_i = (m_i^w, m_i^h)$ is given by $m_i^w = b_i^{x^2} - b_i^{x^1}$ for width and analogously for height. The k-means algorithm [29] is applied to M to determine the centroids in 2D space, which then represent the nine anchors. The anchors are split into triplets and connected with small, medium, and large boxes detected in the output layers. Unfortunately, such a principle of splitting anchors according to three sizes is generally reasonable if

$$M \sim \mathcal{U}(0, r)$$

holds. By $\mathcal{U}(0, r)$ we notate a uniform distribution between the bounds given by 0 and r. However, such a condition cannot be guaranteed for various applications in general. Note $M \sim \mathcal{N}(0.5r, r)$, where $\mathcal{N}(0.5r, r)$ is a normal distribution with mean $\mu = 0.5r$ and standard deviation $\sigma^2 = r$ is a more realistic case, which causes that most of the boxes will be captured by the middle output layer (for the medium size) and the two other layers will be underused.

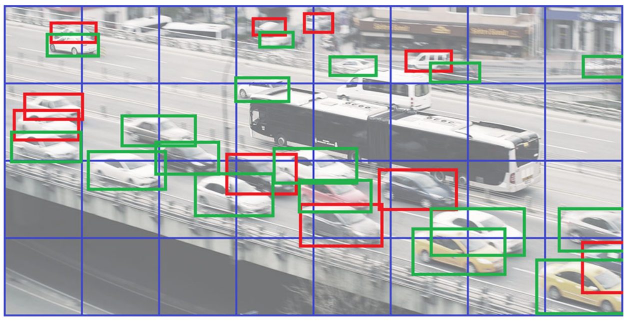


Fig. 5 The image illustrates the label rewriting problem for the detection of cars. A label is rewritten by other if centers of two boxes (with the same anchor box) belong to the same cell. In this illustrative example, blue denotes grid, red rewritten label, and green preserved label. Here, 10 labels out of 27 are rewritten, and the detector is not trained to detect them

Table 1 Amount of rewritten labels for various datasets

Dataset	Resolution	Rewritten labels [%]		
		YOLOv3	Poly YOLO	Poly YOLO lite
Simulator	416×416	16.36	0.22	2.31
Simulator	608×800	12.55	0.00	0.61
Cityscapes	416×416	9.51	2.79	9.50
Cityscapes	608×832	3.92	0.97	2.75
Cityscapes	640×1280	2.56	0.59	1.44
India Driving	416×416	23.07	5.80	13.78
India Driving	448×800	13.54	1.92	4.96
India Driving	704×1280	9.16	1.12	2.44

To illustrate the problem, let us suppose two sets of box sizes, M_1 and M_2; the former connected with the task of car plate detection from a camera placed over the highway and the latter connected with a person detection from a camera placed in front of the door. For such tasks, we can obtain roughly $M_1 \sim \mathcal{N}(0.3r, 0.2r)$ because the plates will cover a small area and $M_2 \sim \mathcal{N}(0.7r, 0.2r)$ because the people will cover large areas. For both sets, the anchors are computed separately. The first case leads to the problem that the output scales for medium and large will also include small anchors because the dataset does not include big objects. Here, the problem of label rewriting will escalate because small objects will need to be detected in a coarse grid. The second case works vice versa. Large objects will be detected in the small and medium output layers. Here, the detection will not be precise because the small and medium output layers have limited receptive fields. The receptive field for the three scales is $\{85 \times 85,\ 181 \times 181,\ 365 \times 365\}$. The practical impact of the two cases is the same: the performance will be suboptimal. In the paper that introduced YOLOv3 [7], the author says "*YOLOv3 has relatively high AP $_{small}$ performance. However, it has comparatively worse performance on medium and larger size objects. More investigation is needed to get to the bottom of this.*" We believe that the reason why YOLOv3 has these problems is explained in the paragraph above. Let us note that generic datasets for object detection have a distribution of sizes from Gaussian's, Poisson's, or binomial distribution rather than from uniform, which is almost unreal. To verify the claim, see Fig. 5 in [30] where is shown the distribution of sizes in COCO, Pascal VOC, Sun and ImageNet. In all cases, we can see the distributions are far from the uniform one.

3.3 Poly-YOLO architecture

Before we describe the architecture itself, let us mention the motivation and the justification for it. As we described in the previous section, YOLO's performance suffers from the problem of label rewriting and the problematic distribution of anchors among output scales.

The first issue can be suppressed by high values of s_k, i.e., a scale multiplicator that expresses the ratio of the output resolution with respect to the input resolution r. The ideal case would happen when $r = rs_k$, i.e., $s_k = 1$, which means that the output and input resolutions are equal. In this case, no label rewriting may occur. Such a condition generally holds in many encoder–decoder-based segmentation NNs such as U-Net [31]. As we are focusing on the computational speed, we have to omit such a scheme to find a solution where $s_k < 1$ will be a reasonable trade-off. Let us recall that YOLOv3 uses $s_1 = 1/8$, $s_2 = 1/16$, $s_3 = 1/32$.

The second issue can be solved in one of two ways. The first way is to define the receptive fields for the three output scales and define two thresholds that will split them. Then, k-means will compute centroid triplets (used as anchors) according to these thresholds. This would change the data-driven anchors to problem-driven (receptive field) anchors. For example, data $M \sim \mathcal{N}(r/5, r/10)$ would be detected only on the scale detecting small objects and not on all scales as it is currently realized in YOLOv3. The drawback of such a way is that we will not use the full capacity of the network. The second way is to create an architecture with a single output that will aggregate information from various scales. Such an aggregated output will also handle all anchors at once. Thus, in contrast to the first way, the estimation of anchor sizes will be again data-driven.

We propose to use a single output layer with a high $\mathbf{s_1}$ scale ratio connected to all anchors, which solves both issues mentioned above. Namely, we use $s_1 = 1/4$. An illustration of a comparison between the original and the new architecture is shown in Fig. 6. For the composition of the single output scale from multiple partial scales, we use the hypercolumn technique [32]. Formally, let O be a feature map, $u(\cdot, \omega)$ a function upscaling an input by a factor ω, and $m(\cdot)$ be a function transforming a feature map with dimension $a \times b \times c \times \cdot$ into a feature map with dimension $a \times b \times c \times \delta$, where δ is a constant. Furthermore, we consider $g(O_1, \ldots, O_n)$ to be an n-nary composition/aggregation function of feature maps $O_1, \ldots, O_n$. For that, the output feature map using the hypercolumn technique is given as

$$O = g\left(m(O_1), u\left(m(O_2), 2^1\right), \ldots, u\left(m(O_n), 2^{n-1}\right)\right).$$

Selecting addition as an aggregation function, the formula can be rewritten as

$$O = \sum_{i=1}^{n} u\left(m(O_i), 2^{i-1}\right).$$

As it is evident from the formula, there is a high imbalance—a single value of O_1 projects into O just a single value, while a single value of O_n is projected into $2_{n-1} \times 2_{n-1}$ values directly. To break the imbalance, we propose to use the staircase approach known from the computer graphic, see Fig. 7. The stairstep interpolation increases (or decreases for downscale) an image resolution by 10% at maximum until the desired resolution is reached. In comparison with a direct upscale, the output is more smooth but does not include, e.g., step artifacts as a direct upsampling does. Here, we will use the lowest available upscale factor, two. Formally, a stairstep output feature map O' is defined as

$$O' = \ldots u(u(m(O_n), 2) + m(O_{n-1}), 2) \ldots + m(O_1).$$

If we consider the nearest neighbor upsampling, $O = O'$ holds. For bilinear interpolation (and others), $O \neq O'$ is reached for non-homogenous inputs. The critical fact is that the computational complexity is equal for both direct upscaling and stairstep upscaling. Although the stairstep approach realizes more adding, they are computed over feature maps with a lower resolution, so the number of added elements is identical.

For understanding the practical impact, we initiated the following experiment. We trained Poly-YOLO for 200 training and 100 validation images from Cityscapes dataset [33] for the version with direct upscaling and stairstep upscaling used in the hypercolumn. We ran the training process five times for each of the versions and plotted the training progress in the graph in Fig. 8. The graph shows that the difference is tiny, but it is evident that stairstep interpolation in hypercolumn yields slightly lower training and validation losses. The improvement is obtained for the identical computation time.

The last way how we propose to modify YOLO's architecture is the usage of squeeze-and-excitation (SE) blocks [34] in the backbone. Darknet-53, like many other neural networks, uses repetitive blocks, where each block consists of coupled convolutions with a residual connection. The squeeze-and-excitation blocks allow the usage of both spatial and channel-wise information, which leads to accuracy improvement. By the addition of squeeze-and-excitation blocks and by working with higher output resolution, the computation speed is decreased. Because speed is the main advantage of YOLO, we reduced the number of

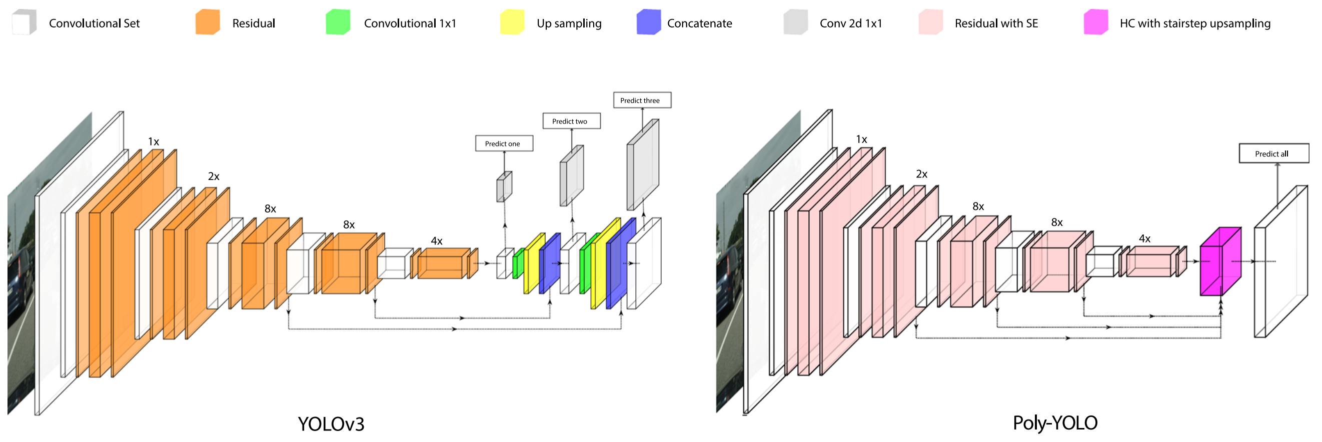


Fig. 6 A comparison of YOLOv3 and Poly-YOLO architecture. Poly-YOLO uses less convolutional filters per layer in the feature extractor part and extends it by squeeze-and-excitation blocks. The heavy neck is replaced by a lightweight block with hypercolumn that utilizes a stairstep for upsampling. The head now uses single instead of three outputs and has a higher resolution. In summary, Poly-YOLO has 40% less parameters than YOLOv3 while producing more precise predictions

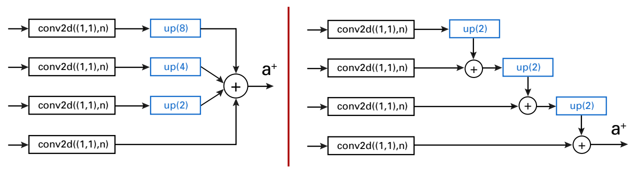


Fig. 7 Illustration of HC scheme (left) and HC with stairstep (right)

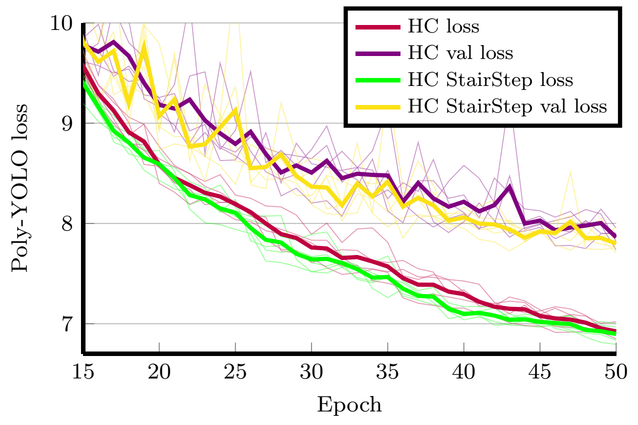


Fig. 8 The graph shows a difference between the usage of the standard hypercolumn technique and the hypercolumn with stairstep in the term of the loss. The thin lines denote particular learning runs, and the thick lines are mean of the runs. The training loss is computed as a loss over the epoch through the samples used for training. The validation loss is computed after each epoch through the samples from the validation dataset, which is used only for validation purposes and not for training. The difference between the training and validation loss expresses the overfitting of the model

convolutional filters in the feature extraction phase. Namely, it is set to 75% of the original number. In addition, the neck and head are lighter, together having 37.1M parameters, which is significantly less than what YOLOv3 has (61.5M). Still, Poly-YOLO achieves higher precision than YOLOv3—see Sect. 5.3. We also propose Poly-YOLO lite, which is aimed at higher processing speed. In the feature extractor and the head, this version has only 66% of the filters of Poly-YOLO. Finally, s_1 is reduced to 1/8. The number of parameters of Poly-YOLO lite is 16.5M.

We want to highlight that for feature extraction, an arbitrary SOTA backbone such as (SE)ResNeXt [12] or EfficientNet [10] can be used, which would probably increase the overall accuracy. Such an approach can also be seen in the paper YOLOv4 [35], where the authors use a different backbone and several other tricks (that can also be applied in our approach), but the head of the original YOLOv3 is left unchanged. The issues we described and removed in Poly-YOLO actually arise from the design of the head of YOLOv3, and a simple swap of a backbone will not solve them. The model would still suffer from label rewriting and improper anchor distribution. In our work, we have focused on the performance improvement achieved by conceptual changes and not brute force. Such improvements are then widely applicable, and a modern backbone can be easily integrated.

4 Instance segmentation with Poly-YOLO

The last sentence in YOLOv3 paper [7] says "*Boxes are stupid anyway though, I'm probably a true believer in masks except I can't get YOLO to learn them.*" Here, we show how to extend YOLO with masking functionality (instance segmentation) without a big negative impact on its speed. In our previous work [1], we were focusing on more precise detection of YOLO by means of irregular quadrangular detection. We proved that the extension for quadrangular detection converges faster. We also demonstrated that classification using the quadrangular approximation yields higher accuracy than using the rectangular approximation. The limitation of that approach lies in the fixed number of detected vertices, namely, four. Here, we introduce a polygon representation that is able to detect objects with a varying number of vertices without the usage of a recurrent neural network that would slow down the processing speed. To see a practical difference between the quality of bounding-box detection and polygon-based detection, see Fig. 10, where we show results from Poly-YOLO trained to detect various geometric primitives including random polygons.

4.1 The principle of bounding polygons

YOLOv3 uses a perpendicular grid consisting of cells where each cell can detect a bounding box, or bounding boxes in the case of multiple anchors. We extend each cell with an additional polar subgrid, see Fig. 9. Let us recall, we can describe a box as $\mathbf{b}_i = (b_i^{x^1}, b_i^{y^1}, b_i^{x^2}, b_i^{y^2})$, i.e., as a tuple of its top-left and bottom-right coordinates. We propose to extend the tuple as $\mathbf{b}_i = (b_i^{x^1}, b_i^{y^1}, b_i^{x^2}, b_i^{y^2}, V_i)$, where $V_i = \{\mathbf{v}_{i,0}, \ldots, \mathbf{v}_{i,n}\}$ is a set of polygon vertices of a given object with n polar cells. Furthermore, $\mathbf{v}_{i,j} = (\alpha_{i,j}, \beta_{i,j}, \gamma_{i,j})$, where α and β are the coordinates of a polygon vertex in a polar coordinate system and γ is its confidence. If no vertex is present in a polar cell, the confidence should be ideally equal to zero; otherwise, it should be equal to one. The purpose of the confidence is to indicate whenever a given cell contains an object vertex or not. For example, Fig. 9 contains a star, and some polar cells do not have a vertex in them. Such cells will have confidence 0, and their predictions will be ignored. In the case of polar cells with vertices, confidence 1. In practice, Poly-YOLO produces confidences in [0,1]. The threshold for the absence/presence of a vertex in a cell is set to 0.5.

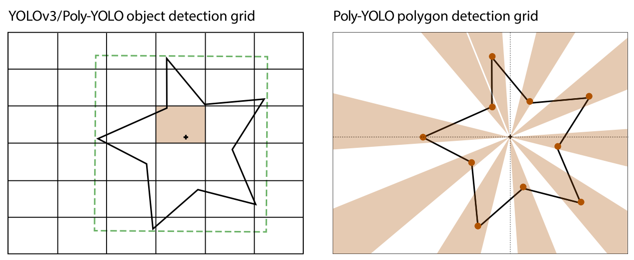

YOLOv3/Poly-YOLO object detection grid
Poly-YOLO polygon detection grid

Fig. 9 The image illustrates grids used in Poly-YOLO. Left: the rectangular grid, which is taken from YOLOv3. A cell where an object's bounding box has its center predicts its bounding box coordinates. Right: the grid based on circular sectors used in Poly-YOLO for the detection of vertices of the polygon. The center of the grid coincides with the center of the object's bounding box. Each circular sector is then responsible for detecting polar coordinates of the particular vertex. Sectors, where no vertex is present, should yield confidence equal to zero

In a common dataset, many objects are masked with a similar shape because they are captured from a similar viewpoint; the difference is only in the object size. For example, instances of car plates, hand gestures, humans, or cars have almost identical shapes. The general shape can be easily described using polar coordinates, which is the motivation why we use the polar coordinate system instead of the Cartesian one for bounding polygons. Here, $\alpha_{i,j}$ stands for the distance of a vertex from the origin and $\beta_{i,j}$ for an oriented angle. The center of a bounding box is used as the origin. Furthermore, we divide the vertex distance from the origin by the length of a diagonal of the bounding box to obtain $\alpha_{i,j} \in [0, 1]$. Then, during the inference, when the bounding box with a bounding polygon is detected, the absolute distance from the origin is obtained by multiplying $\alpha_{i,j}$ with the diagonal of the detected box. The principle allows the network to learn general, size-independent shapes, not particular instances with sizes. For example, let us suppose two images of the same car that are placed at two distinct distances from a camera so that in each image, its size will be different. The model will be trained to detect confidence, angles, and relative distances from the bounding box center for every vertex. These values will be the same for both images. When the predictions are realized, the distances are multiplied by the box diagonal, and the particular values of the two differently sized cars will be obtained. In comparison with PolarMask [25] which has to predict distinct distances for different object sizes, this sharing of values should make the learning easier.

Still, additional improvement is possible. For the oriented angle, it holds that $\beta_{i,j} \in [0, 360]$, which can be changed to $\beta_{i,j} \in [0, 1]$ by a linear transformation. Because our polar system is split into polar cells, it would be beneficial to focus the inside of a cell to a particular part of the angle interval, which is covered by the cell. When a certain polar cell fires with high confidence, the vertex has to be inside the cell. Therefore, we propose to take $\beta_{i,j} \in [\beta^1, \beta^2]$, where $\beta^1_{i,j}$ and $\beta^2_{i,j}$ are the minimum and the maximum angle captured by the polar cell in which the vertex lies. Then, we make a linear transformation of $\beta_{i,j}$, where $\beta^1_{i,j} = 0$ and $\beta^2_{i,j} = 1$ hold. In other words, when a certain polar cell has high confidence, we know that it contains a vertex. By a distance from the origin and the location of the polar cell, we know its approximate position, and by the angle inside the cell, we refine the position precisely.

4.2 Integration with Poly-YOLO

The idea of detecting bounding polygons is universal and can be easily integrated into an arbitrary neural network. In general, three parts have to be modified: the way how the data are prepared, an architecture and a loss function. For the extraction of bounding polygons from semantic segmentation labels, see Sect. 5.1. The extracted bounding polygons have to be augmented in the same way as the bounding boxes.

The architecture has to be modified to produce the intended values. In the case of Poly-YOLO, the number of convolutional filters in the output layer has to be updated. When we detect only bounding boxes, the last layer is represented by $n = n^a(n^c + 5)$ convolutional filters with a kernel of dimension (1, 1), where n^a is the number of anchors (nine, in our case) and n^c stands for the number of classes. After integrating the extension for polygon-based object detection, we obtain $n = n^a(n^c + 5 + 3n^v)$, where n^v is the maximal number of detected vertices per polygon. We can observe that n^v has a high impact on the number of convolutional filters. For example, when we have nine anchors, twenty classes and thirty vertices, the output layer detecting bounding boxes and polygons will have $4.6\times$ more filters than when detecting bounding boxes only. On the other hand, the increase happens only in the last layer; all remaining YOLO layers have the same number of parameters. From that point of view, the total number of the NN parameters is increased by a negligible 0.83%, and the processing speed is not affected. The weak point lies in the fact that the increase is in the last layer, which processes high-resolution feature maps. This causes an increased demand for VRAM for a symbolic tensor when the network is trained, which may lead to a decrease of the maximum possible batch size used during the learning phase.

For explaining how a loss function has to be modified, we describe the multi-part loss function ℓ used in Poly-YOLO as follows:

$$\ell = \sum_{i=0}^{G^w G^h} \sum_{j=0}^{n^a} q_{i,j}[\ell_1(i,j) + \ell_2(i,j) + \ell_3(i,j) + \ell_5(i,j)] + \ell_4(i,j),$$

where $\ell_1(i,j)$ is a loss for a prediction of a center of a bounding box, $\ell_2(i,j)$ is a loss for the dimensions of a box, $\ell_3(i,j)$ is the confidence loss, $\ell_4(i,j)$ is the class prediction loss and $\ell_5(i,j)$ is a loss for a bounding polygon made of distance, angle and vertex confidence prediction. Finally, $q_{i,j} \in \{0, 1\}$ is a constant indicating if the i-th cell and the j-th anchor contains a label or not. The loss iterates over $G^w G^h$ grid cells and n^a anchors. The parts $\ell_1, \ldots, \ell_4$ are taken from YOLOv3 and modified into a form working with a single output layer. Part ℓ_5 is new and extends Poly-YOLO with the functionality of polygon detection. In the following formulas, we use $\hat{\cdot}$ to denote predictions of the network. The parts of the loss function are defined as follows:

$$\ell_1(i,j) = z_{i,j}\left[H(c^x_{i,j}, \hat{c}^x_{i,j}) + H(c^y_{i,j}, \hat{c}^y_{i,j})\right],$$

where $c^x_{i,j}$ and $c^y_{i,j}$ are coordinates of the center of a box, $H(\cdot,\cdot)$ is the binary cross-entropy, $z_{i,j} = 2 - w_{i,j}h_{i,j}$ serves for a relative weighting of (i, j)-th box size according to its width $w_{i,j}$ and height $h_{i,j}$.

$$\ell_2(i,j) = 0.5 z_{i,j}\left[\left(\log\left(\frac{w_{i,j}}{a^w_j}\right) - \hat{w}_{i,j}\right)^2 + \left(\log\left(\frac{h_{i,j}}{a^h_j}\right) - \hat{h}_{i,j}\right)^2\right],$$

where a^w_j and a^h_j are the width and height of the j-th anchor.

$$\ell_3(i,j) = q_{i,j}H(q_{i,j}, \hat{q}_{i,j}) + (1 - q_{i,j})H(q_{i,j}, \hat{q}_{i,j})I_{i,j},$$

where $\hat{q}_{i,j}$ is the predicted confidence and $I_{i,j}$ is a mask which excludes the part of a loss for the i-th cell if $q_{i,j} = 0$ but its prediction has $\mathrm{IoU} > 0.5$.

$$\ell_4(i,j) = \sum_{k=0}^{c} H(C_{i,j,k} - \hat{C}_{i,j,k}),$$

where $C^{i,j,k}$ is k-th class probability in i-th cell. Finally,

$$\ell_5(i,j) = 0.2 \sum_{l=0}^{v} z_{i,j}\left[\gamma_{i,j,k}\left(\log\left(\frac{\alpha_{i,j,k}}{a^d_j}\right) - \hat{\alpha}_{i,j,k}\right)^2 + \gamma_{i,j,k}H(\beta_{i,j,k}, \hat{\beta}_{i,j,k}) + H(\gamma_{i,j,k}, \hat{\gamma}_{i,j,k})\right],$$

where a^d_j is the diagonal of the j-th anchor. Note that the last equation is our polygon representation loss, one of our main contributions.

The described scheme of integration results in the simultaneous detection of both bounding boxes and bounding polygons. Such a combination may be beneficial due to the synergy—convolutional neural networks detect edges in its bottom, then combine them into more complex shapes in the middle and propose highly descriptive abstract features in a head [36]. Because the polygon vertices always lie in the bounding box and because the vertices delimit the same object as the bounding box, the intuition is the bounding polygon part will find features useful for the bounding box and vice versa. The assumption is that the training of YOLO with polygon shape detection extension will be more efficient and converge faster. The principle is well known and described in the literature as *Auxiliary task learning* [37]. For completeness, let us suppose a special case when an object is a perpendicular box. In such a case, the contour of the bounding box will coincide with the contour of the bounding polygon, and the left-top vertex will be detected by both the bounding box and polygon. Still, the two detections will be synergistic, and the training will require a shorter time than the training of vanilla bounding box detection. For the verification of the claims, see results of Poly-YOLO detection with/without bounding polygon detection in Sect. 5.3.

5 Benchmarks

Here, we describe the experiments and results which we realized. We divide them into two scenarios, algorithms for pure bounding box detection and algorithms for instance segmentation. Each scenario includes three datasets, namely Simulator, Cityscapes and IDD. For the training, we use three computers with RTX2080Ti, RTX2060, or GTX1080 graphics cards. The inference time is always measured on the computer with the 2080 card. In the experiments, we set the maximum number of vertices for Poly-YOLO to 24.

5.1 Preparing data for Poly-YOLO

As was stated before, one of the features of Poly-YOLO is an object detection with the ability to estimate a polygon tightly wrapping around the object. For that purpose, quite specific data must be fetched in. Commonly, publicly available datasets focus on pixel-precise segmentation masks. The mask assigns each pixel to one of the many predefined classes. Unfortunately, for Poly-YOLO, we need a polygonal representation and not pixel-wise representation. Because of that, some pre-processing became inevitable.

As an input to the extraction of a bounding polygon, we suppose a blob of pixel coordinates of a given object. That notation is standard for general pixel segmentation tasks. Then, we find the most distant points from an object center. It means that if an object is folded, we do not extract the inner boundary points, just the most distant ones. For example, an object with a shape of a Swiss roll will have a contour similar to the circle. Finally, we erase points that lie in a straight line between two other points. Note, for a single object, we always extract a single bounding polygon. If we take, e.g., a car which is partly overlapped by a tree, the bounding polygon will bound the whole car, including the part covered behind the tree.

Another way to obtain training data is to generate them synthetically. For that, we use two of our tools. The first one serves for a generation of complex and realistic scenes, as is shown in Fig. 11. The second one can generate an infinite number of images, where the following parameters can be configured: the resolution of images, the number of geometric primitives per image, the type of geometric primitives and the range of their size. It is also possible to add a random background. For the illustration, see Fig. 10. The tool is available at our GitLab repository.[1]

5.2 Datasets

In the benchmark, we use three datasets: Simulator, Cityscapes [33] and India Driving [38].

Simulator is our own synthetic dataset available online,[2] consisting of 700 training, 90 validation and 100 test images with a resolution of 600×800 px. The dataset is useful for fast prototyping, hyperparameter searching, or as a starting point for transfer learning because the low number of images allows fast training, and the captured scenes are trivial. It includes only a single class (a car), where its particular instances are rendered using a single 3D model. On the other hand, the scene is illuminated by physically precise lights. An illustrative image with detection by Poly-YOLO is visualized in Fig. 11.

Cityscapes is a dataset captured from a car driven through various German cities. The images were captured during the day or evening, and all of them have the same resolution of 2048×1024 px. For Cityscapes, both bounding boxes and pixel-level labels are available. The process, how we extracted polygons from pixel-level annotations is described in Sect. 5.1. Notwithstanding, the pixel-level labels include objects such as sky, building and tree, we use only objects connected with traffic such as a car, pedestrian, or bike. In total, we predict 12 classes. Because the testing dataset does not contain labels, we moved several images from the training to the testing dataset. Finally, our train/valid/test datasets consist of 2474/500/500 images. The information about the particular distribution of the images can be found in our repository.

IDD is a dataset similar to Cityscapes that focuses on unstructured traffic on India's roads. In contrast to Cityscapes, the road borders captured in the images are fuzzy, traffic is heavier and messy, and the images have various resolutions. The distribution of images is as follows—14010 for training, 977 for validation and 1049 for testing.

[1] https://gitlab.com/irafm-ai/poly-yolo/-/tree/master/synthetic_dataset.

[2] https://gitlab.com/irafm-ai/poly-yolo/-/tree/master/simulator_dataset.

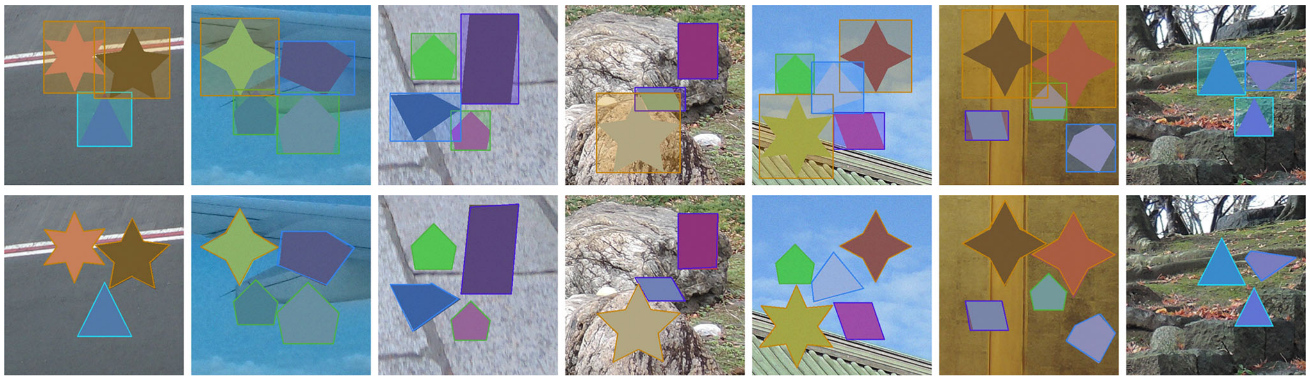


Fig. 10 Comparison of Poly-YOLO bounding box detection (top) and Poly-YOLO bounding polygon detection (bottom)

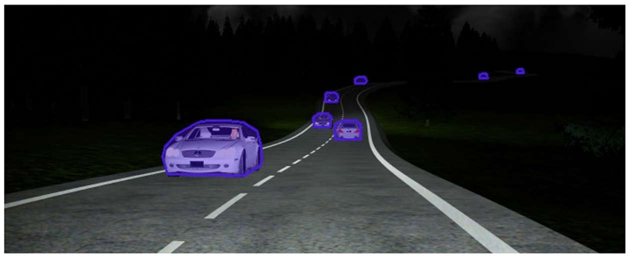


Fig. 11 The figure shows an illustrative image from the Simulator dataset. The blue color shows prediction realized by Poly-YOLO lite at 52FPS. The image was slightly cropped to increase visibility

5.3 Results

We present all measured results for the three datasets in Table 2. Poly-YOLO were trained separately for the version of pure bounding box detection and with bounding polygon detection. The original vanilla version[3] of YOLOv3, which we modified into the Poly-YOLO version, is included as well. For the comparison with SOTA, we have trained RetinaNet[4] as a representative of the bounding box detection algorithm and Mask R-CNN[5] as a representative of the instance segmentation algorithms. The SOTA algorithms were trained using transfer learning. Poly-YOLO was trained from scratch due to the fact that no pre-trained model is available. The models are trained until early stopping is reached. To see the detailed setting of the training procedure, please check our repository. For the evaluation of the results, we use the mAP (mean average precision) coefficient from the official COCO repository.[6]

Overall, we can observe that Poly-YOLO significantly increases YOLOv3 detection accuracy (the relative average increase is 40%), although the inference speed is slightly faster. On the other hand, Poly-YOLO lite slightly outperforms the detection accuracy of YOLOv3, but it is twice faster. The important fact is also that Poly-YOLO with bounding polygon preserves the bounding box detection accuracy. Moreover, the accuracy of bounding boxes is increased in four out of six cases, so we can say the features used for bounding polygons are suitable for bounding boxes too. If we compare Poly-YOLO with RetinaNet, we have to report that RetinaNet yields higher precision, but it is slower in two of the three cases.

Let us also emphasize that the used framework and operating system have a significant impact. According to the original RetinaNet paper [4], it should run approximately 10FPS. However, the authors have rewritten the original RetinaNet implementation with the usage of the PyTorch framework, have optimized it massively, and made it available only for Linux, which clearly leads to a significant speed-up. According to the official documentation of Detectron2[7] library from which we used RetinaNet implementation, the reimplementation increased the computation speed three times. Therefore, the comparison of the computation speed is slightly unfair for us as we are using Tensorflow and Windows. Thus, the open direction and future work is to rewrite our Poly-YOLO to PyTorch by a coding expert to reach even higher computation speed. Therefore, we mark RetinaNet inference speed in the table with * symbol. Let us note YOLOv3 and Mask R-CNN were performed using the same framework and OS as Poly-YOLO, i.e., Tensorflow and Windows.

In the case of the simulator dataset where RetinaNet has the same resolution as Poly-YOLO, Poly-YOLO has higher AP_{50}, but it is less accurate for AP_{75}, which can be given by the fact that RetinaNet utilizes more anchor boxes, which can improve the precise detection. For the two other datasets, RetinaNet has been trained for a higher resolution (selected automatically), and it is better even for AP_{50}. When we analyzed the outputs, we observed that Poly-

[3] https://github.com/qqwweee/keras-yolo3.

[4] https://github.com/facebookresearch/detectron2.

[5] https://github.com/matterport/Mask_RCNN.

[6] https://github.com/cocodataset/cocoapi/tree/master/PythonAPI.

[7] https://detectron2.readthedocs.io/notes/benchmarks.html.

Table 2 The results of the involved algorithms for bounding box detection and instance segmentation on the three datasets

Method	Backbone	Resolution	Instance	Box			Mask			FPS
			segment	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	
Performance on the simulator dataset										
RetinaNet	ResNet-50 FPN	608×800	✗	0.475	0.714	0.487	–	–	–	25.0*
YOLOv3	Darknet-53	608×800	✗	0.305	0.699	0.220	–	–	–	21.2
Poly-YOLO	SE-Darknet-53	608×800	✗	0.413	0.735	0.408	–	–	–	22.0
Poly-YOLO	SE-Darknet-53 lite	416×576	✗	0.322	0.661	0.258	–	–	–	58.6
Mask R-CNN	ResNet-50	448×448	✔	0.389	0.664	0.414	0.203	0.452	0.157	15.8
Poly-YOLO	SE-Darknet-53	608×800	✔	0.435	0.745	0.445	0.345	0.731	0.272	19.6
Poly-YOLO	SE-Darknet-53 lite	416×576	✔	0.377	0.694	0.348	0.298	0.675	0.270	52.7
Performance on the cityscapes dataset										
RetinaNet	Resnet-50 FPN	608×1216	✗	0.224	0.379	0.231	–	–	–	21.0*
YOLOv3	Darknet-53	416×832	✗	0.106	0.266	0.061	–	–	–	26.3
Poly-YOLO	SE-Darknet-53	416×832	✗	0.168	0.344	0.141	–	–	–	26.5
Poly-YOLO	SE-Darknet-53 lite	320×608	✗	0.104	0.231	0.080	–	–	–	46.8
Mask R-CNN	Resnet-50	1024×1024	✔	0.164	0.318	0.151	0.069	0.202	0.031	6.2
Poly-YOLO	SE-Darknet-53	416×832	✔	0.129	0.273	0.105	0.087	0.240	0.046	21.9
Poly-YOLO	SE-Darknet-53 lite	320×608	✔	0.114	0.253	0.091	0.078	0.217	0.044	37.2
Performance on the India driving dataset										
RetinaNet	Resnet-50 FPN	608×1080	✗	0.221	0.357	0.230	–	–	–	19.8*
YOLOv3	Darknet-53	448×800	✗	0.117	0.267	0.089	–	–	–	23.9
Poly-YOLO	SE-Darknet-53	448×800	✗	0.152	0.304	0.137	–	–	–	25.5
Poly-YOLO	SE-Darknet-53 lite	352×608	✗	0.125	0.260	0.105	–	–	–	46.7
Mask R-CNN	Renset-50	1024×1024	✔	0.175	0.300	0.177	0.098	0.217	0.077	7.5
Poly-YOLO	SE-Darknet-53	448×800	✔	0.145	0.288	0.134	0.115	0.267	0.083	20.6
Poly-YOLO	SE-Darknet-53 lite	352×608	✔	0.131	0.263	0.119	0.101	0.239	0.074	37.1

YOLO has a less precise class classification. That may be given by the fact that it uses a categorical cross-entropy for the classification loss function, while RetinaNet uses focal loss that works significantly better for imbalanced datasets. Here, it may be beneficial to integrate the focal loss into Poly-YOLO in future work. From the last comparison with Mask R-CNN, we can report that Mask R-CNN has slightly better box detection accuracy, but it is less accurate in masking. Furthermore, its processing speed is lower than the Poly-YOLO processing speed. That makes it complicated for real-time image/video processing.

Note the value of the precision numbers for the neural networks is lower than the state-of-the-art results. That is, by the fact that our graphics cards are unable to process colossal batch sizes, and because of the training time, we do not realize the enormous amount of iterations with a fixed decrease of a learning rate, but we control the learning rate dynamically and utilize early stopping.

6 Discussion

Here, we present the impact of hyperparameter setting on Poly-YOLO performance, discussing additional improvements and current limitations.

6.1 Hyperparameters

In Poly-YOLO, three aspects should be examined: the way, how squeeze-and-excitation blocks are integrated into the architecture, a dependency on a number of vertices and a dependency on a number of anchors.

According to the original paper [34], the best result is achieved when the squeeze-and-excitation block is placed in the first position in the residual block, before convolution. However, Darknet-53 uses a special tuple of convolution where the expansion layer follows the bottleneck layer. Therefore, we realized a simple experiment whose results are shown in Fig. 13. From this graph, we can see that SE-Standard, i.e., placing a squeeze-and-excitation block after the convolution gives the best result. On the basis of this experiment, we use the setting of SE-Standard in our SE-Darknet-53.

The second aspect is the maximal number of possible vertices in the polygon, or in other words, the resolution of the polar grid. Let us note, if an object is quadrangular, there is no difference if the maximal number is four or twenty. On the other hand, if an object is complicated and defined by, e.g., 80 vertices, with the maximal number set to 10, we will lose information due to quantization. In Fig. 12, we examine the impact of the maximal number of vertices on the loss and a mean average precision. According to the definition of the loss function given in Sect. 4.2, the error is summed over all vertices. Therefore, a bigger maximal number of vertices should produce a higher loss if the objects consist of enough vertices. That assumption is reflected in the graph. It is interesting that the dependency is sub-linear. It means that increasing the maximal number of vertices does not produce a significantly more complicated task. On the contrary, increasing this number may lead to smaller quantization and deliver more information useful in the training of a network. The disadvantage is that as the number goes up, it increases the number of parameters in the high-resolution output tensor. That may force us to use smaller batch sizes and, therefore, to an increase in the training time. Thus, the proper selection is up to a user and available hardware.

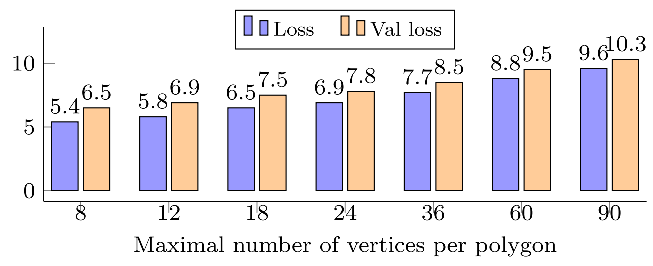
The last aspect to investigate how the precision depends on the number of anchors used. Let us recall, YOLOv2 uses a single output layer and five anchors, YOLOv3 uses three output layers with three anchors per layer, nine in total. When the labels are pre-processed, each label is assigned to an anchor, for which the IOU is maximized. Then, such an anchor is used to detect a box for that particular label the network is trained. From that, it is evident that the higher number of anchors makes the task more complicated. On the other hand, a higher number of anchors may be helpful to partially solve the label rewriting problem mentioned in Sect. 3.2. In Fig. 12, we show the results of the experiment where a network is trained for a various number of anchors. According to the loss, the optimal value is between six and nine anchors; a higher or lower number increases both the training loss and the validation loss. We have selected the same number as YOLOv3, i.e., nine (Fig. 13).

6.2 Emphasizing parts of detections

As we mentioned in Sect. 1, a practical application where quick instance segmentation may be helpful is the implementation of an intelligent car headlamp, where various objects in front of a car can be lightened/dimmed individually. The precise object detection based on the polygonal principle, which we propose, uses polar coordinates that allow improving the functionality even further. Let us consider classes such as car, biker, pedestrian, or van. Objects of such a class should be lightened more (to increase their visibility), but it is also necessary to avoid their dazzling. The solution is not to illuminate parts that include a front glass of a car, the head of a pedestrian, etc. To detect such parts, additional extensive data labeling would be required in the standard case. In our case, it is enough to manually define an interval in the polar coordinate system that should not be dazzled. The benefits are that such a manual definition is fast, easily controlled, it does not affect training/inference speed, it is explainable, and what is essential—it is independent on the size of an object or its aspect ratio. Finally, it is not necessary to define additional labels. The illustration of such inference with this additional extension is shown in Fig. 14.

6.3 Limitations

The major limitation discovered during our research came from the scheme used for polygon vertices definition and the scheme of how the labels are created, as is described in Sect. 5.1. If two vertices belong to the same polar cell, the vertex with a bigger distance from the bounding box center is taken. That may lead to a situation when a new part to a strongly non-convex object is added, as it is shown in



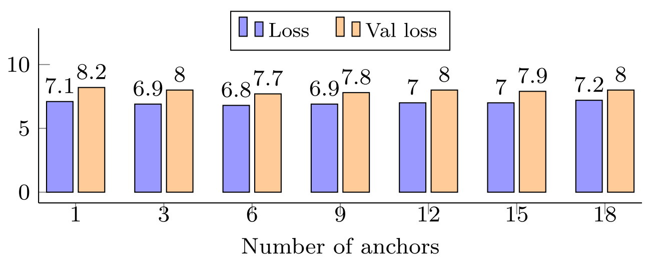


Fig. 12 Left: dependence of number of vertices on loss for 9 used anchors. Right: dependence of number of anchors on loss for 24 used vertices. The training loss is computed as a loss over the epoch through the samples used for training. The validation loss is computed after each epoch through the samples from the validation dataset, which is used only for validation purposes and not for training. The difference between the training and validation loss expresses the overfitting of the model

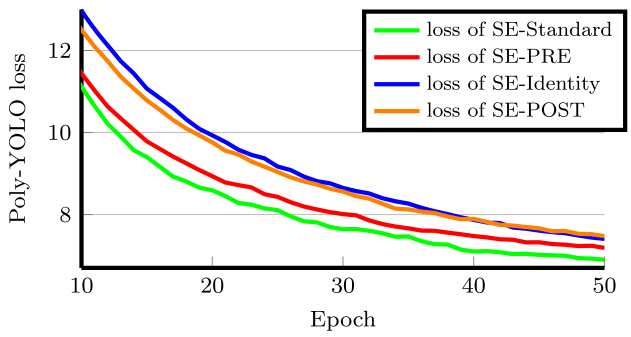


Fig. 13 Progress of training loss for various SE block placing variants. The marked lines are computed as the mean from five runs

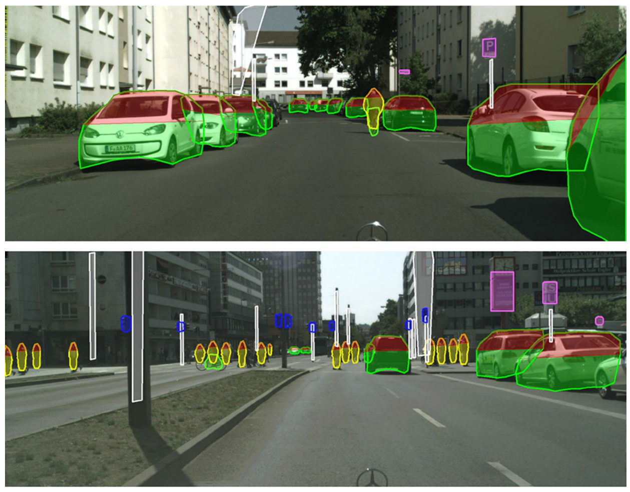


Fig. 14 The two images illustrate the case when there are manually defined intervals for angles of vertices (for cars and for pedestrians). Object area defined by these vertices should be dimmed, while the rest of the object should be emphasized by car headlamps

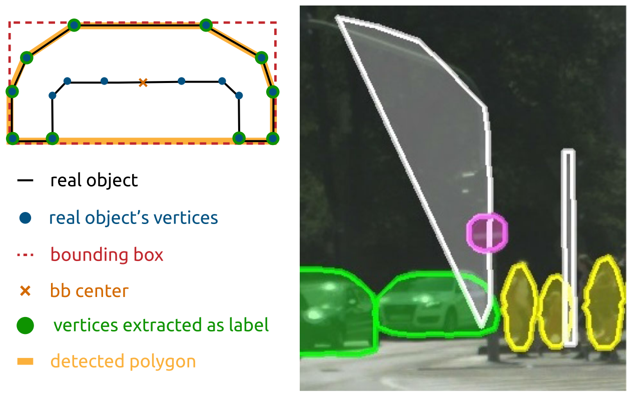


Fig. 15 Left: a scheme of label creation for a problematic object, where the limitation appear. Right: impact on a real predictions, see the lamp object

Fig. 15. The figure also shows the practical impact of this limitation. Let us note it is not a problem of training or inference. It is a problem of the creation of labels; the network itself is trained correctly and makes predictions based on the (imprecise) training labels. After we connect individual vertices, we connect the first vertex with the last one, and this can cause problems for strongly non-convex objects. For completeness, this behavior does not happen for all non-convex objects. If two vertices lie in two distinct polar cells, even non-convex objects will be handled correctly, as can be seen in Fig. 10, where Poly-YOLO works nicely even for non-convex stars.

6.4 Future work

The Poly-YOLO takes as the labels vertices, together with other pieces of information. The vertices are constructed as those points in polar cells, where the distance from the center is maximalized. Such a process is easy to implement but does not guarantee a maximized IoU between such the established polygon and the original, "uncompressed," object's area. Future work should address the optimal selection of the vertices. That can be realized by, e.g., a genetic algorithm that will take the original object's contour as an input and produce a nearly optimal representation given by the vertices. There is a hypothesis that the more precise will be the labels used for training Poly-YOLO, the more precise will be the predictions.

7 Summary

We have presented Poly-YOLO, which improves YOLOv3 in three aspects. It is more precise, faster and able to realize instance segmentation. The precision is improved due to the analysis of issues in YOLO (rewriting of labels and incorrect distribution of anchors) and their removal by a newly proposed neck and head. The neck consists of the hypercolumn technique improved by the stairstep approach, and the head processes a single output tensor with high resolution. The new neck and head reach higher precision, which allows us to decrease the number of parameters in the original feature extractors, while still preserving a significantly higher precision. Poly-YOLO has only 60% of the parameters of YOLOv3 but improves the accuracy by relatively 40%.

For the task of instance segmentation, we have designed an extension that detects bounding polygons with a dynamic number of vertices per detected object. The proposed bounding polygon detection learns itself size-independent shapes, which simplifies the task. Poly-YOLO is able to run real-time on mid-tier graphics cards.

The approach based on Poly-YOLO reached the second place in a worldwide Signate competition on object detection and tracking. For details, see the repository.[8]

[8] https://gitlab.com/irafm-ai/signate_3rd_ai_edge_competition.

Acknowledgements The work is supported by ERDF/ESF "Centre for the development of Artificial Intelligence Methods for the Automotive Industry of the region" (No. CZ.02.1.01/0.0/0.0/17049/0008414).

Declarations

Conflict of interest Authors declare that they have no conflict of interest.

Human and animals participants This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. Hurtik P, Molek V, Vlasanek P (2020) YOLO-ASC: you only look once and see contours, accepted. In: Proceedings of IEEE-WCCI conference
2. Kirillov A, He K, Girshick R, Rother C, Dollár P (2019) Panoptic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 9404–9413
3. Ren S, He K, Girshick R, Sun J (2015) Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in neural information processing systems, pp 91–99
4. Lin TY, Goyal P, Girshick R, He K, Dollár P (2017) Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision, pp 2980–2988
5. He K, Gkioxari G, Dollár P, Girshick R (2020) Mask R-CNN. IEEE Trans Pattern Anal Mach Intell 42(2):386–397
6. Fu CY, Shvets M, Berg AC (2019) RetinaMask: learning to predict masks improves state-of-the-art single-shot detection for free. arXiv preprint, arXiv:1901.03353
7. Redmon J, Farhadi A (2018) Yolov3: an incremental improvement. arXiv preprint, arXiv:1804.02767
8. Huang R, Pedoeem J, Chen C (2018) YOLO-LITE: a real-time object detection algorithm optimized for non-GPU computers. In: 2018 IEEE international conference on big data (big data). IEEE, pp 2503–2510
9. Oltean G, Florea C, Orghidan R, Oltean V (2019) Towards real time vehicle counting using yolo-tiny and fast motion estimation. In: 2019 IEEE 25th international symposium for design and technology in electronic packaging (SIITME). IEEE, pp 240–243
10. Tan M, Pang R, Le QV (2019) Efficientdet: scalable and efficient object detection. arXiv preprint, arXiv:1911.09070
11. Law H, Deng J (2018) Cornernet: detecting objects as paired keypoints. In: Proceedings of the European conference on computer vision (ECCV), pp 734–750
12. Xie S, Girshick R, Dollár P, Tu Z, He K (2017) Aggregated residual transformations for deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1492–1500
13. Shen Zhiqiang, Liu Zhuang, Li Jianguo, Jiang Yu-Gang, Chen Yurong, Xue Xiangyang (2019) Object detection from scratch with deep supervision. IEEE Trans Pattern Anal Mach Intell 42(2):398–412
14. Zhu R, Zhang S, Wang X, Wen L, Shi H, Bo L, Mei T (2019) ScratchDet: training single-shot object detectors from scratch. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2268–2277
15. Girshick R (2015) Fast R-CNN. In: Proceedings of the IEEE international conference on computer vision, pp 1440–1448
16. Dollár Piotr, Appel Ron, Belongie Serge, Perona Pietro (2014) Fast feature pyramids for object detection. IEEE Trans Pattern Anal Mach Intell 36(8):1532–1545
17. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg AC (2016) SSD: single shot multibox detector. In: European conference on computer vision. Springer, pp 21–37
18. Zhang S, Wen L, Bian X, Lei Z, Li SZ (2018) Single-shot refinement neural network for object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4203–4212
19. Tian Z, Shen C, Chen H, He T (2019) FCOS: fully convolutional one-stage object detection. In: Proceedings of the IEEE international conference on computer vision, pp 9627–9636
20. Chen X, Girshick R, He K, Dollár P (2019) TensorMask: a foundation for dense object segmentation. In: Proceedings of the IEEE international conference on computer vision, pp 2061–2069
21. Bolya D, Zhou C, Xiao F, Lee YJ (2019) YOLACT++: better real-time instance segmentation. arXiv preprint, arXiv:1912.06218
22. Newell A, Huang Z, Deng J (2017) Associative embedding: end-to-end learning for joint detection and grouping. In: Advances in neural information processing systems, pp 2277–2287
23. Liu S, Jia J, Fidler S, Urtasun R (2017) SGN: sequential grouping networks for instance segmentation. In: Proceedings of the IEEE International conference on computer vision, pp 3496–3504
24. Wang X, Zhang R, Kong T, Li L, Shen C (2020) SOLOv2: dynamic, faster and stronger. arXiv preprint, arXiv:2003.10152
25. Xie E, Sun P, Song X, Wang W, Liu X, Liang D, Shen C, Luo P (2019) PolarMask: single shot instance segmentation with polar representation. arXiv preprint, arXiv:1909.13226
26. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 779–788
27. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1–9
28. Redmon J, Farhadi A (2017) YOLO9000: better, faster, stronger. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7263–7271
29. MacQueen J et al (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol 1. Oakland, CA, USA, pp 281–297
30. Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollár P, Zitnick CL (2014) Microsoft coco: common objects in context. In: European conference on computer vision. Springer, pp 740–755
31. Ronneberger O, Fischer P, Brox T (2015) U-Net: convolutional networks for biomedical image segmentation. In: International conference on medical image computing and computer-assisted intervention. Springer, pp 234–241
32. Hariharan Bharath, Arbelaez Pablo, Girshick Ross, Malik Jitendra (2016) Object instance segmentation and fine-grained localization using hypercolumns. IEEE Trans Pattern Anal Mach Intell 39(4):627–639
33. Cordts M, Omran M, Ramos S, Rehfeld T, Enzweiler M, Benenson R, Franke U, Roth S, Schiele B (2016) The cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 3213–3223
34. Hu J, Shen L, Sun G (2018) Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 7132–7141

35. Bochkovskiy A, Wang CY, Liao HY (2020) YOLOv4: optimal speed and accuracy of object detection. arXiv preprint, arXiv:2004.10934
36. Lee Honglak, Grosse Roger, Ranganath Rajesh, Ng Andrew Y (2011) Unsupervised learning of hierarchical representations with convolutional deep belief networks. Commun ACM 54(10):95–103
37. Ruder S (2017) An overview of multi-task learning in deep neural networks. arXiv preprint, arXiv:1706.05098
38. Varma G, Subramanian A, Namboodiri A, Chandraker M, Jawahar CV (2019) IDD: a dataset for exploring problems of autonomous navigation in unconstrained environments. In: 2019 IEEE winter conference on applications of computer vision (WACV). IEEE, pp 1743–1751