Real-time Simultaneous Object Detection and Depth Estimation

Vlad-Cristian Miclea and Sergiu Nedevschi
Department of Computer Science
Technical University of Cluj-Napoca, Cluj-Napoca, Romania
Vlad.Miclea@cs.utcluj.ro, Sergiu.Nedevschi@cs.utcluj.ro

Abstract

This paper introduces a novel method to simultaneously compute 2D object bounding boxes and estimate the depth map from a single image. A Yolo-based object detection convolutional neural network is modified such that it also includes a depth estimation, both outputs being derived from the same backbone. While the detection part follows the one-stage object detection procedure, depth estimation is computed in an encoder-decoder fashion, the problem being modeled as a classification. Furthermore, since the classification can only produce depth values in a discrete limited range, a correction method is introduced. The correction uses a genetic algorithm to interpolate a sub-pixel accurate depth map from the softmax layer in the CNN. We show that the bounding boxes and depth maps can be accurately computed in real-time on a regular GPU.

1. Introduction

Environment perception is a very important part of automated driving. The surrounding scene could be understood only if sensors such as cameras, Lidars and radars can provide fast solutions with very high accuracy.

Our main research goal is to obtain a 3D understanding of the driving scene. Moving towards this goal, the first objective is to generate the depth map and 2D bounding boxes around moving objects with high accuracy (several methods for this [3], [8]). Temporally associating 3D Lidar points to objects detected by a camera is an extremely hard problem. The complexity further increases if multiple cameras are employed. Moreover, the sparsity of results [7] might not provide enough points for a clear 3D scene understanding. On the other hand, accurate stereo reconstruction [4] still suffers from a heavy computational load. Therefore, we focus our attention to MDE (monocular depth estimation) networks [2], [5], which have several advantages over their counterparts: the solution is cheaper in terms of both equipment cost and computational resources, there is no need for extra temporal alignment and it produces results with reasonable accuracy.

Our method uses deep learning techniques to compute both of the outputs, the depth map and bounding boxes for the objects. Firstly, we employ a Yolo-based CNN encoder used as a backbone for the solution to extract features at multiple scales, which we then pass to i) the Yolo head for object detection and ii) to a depth head, which acts as a classifier by assigning a depth to each pixel. Since the classification produces results in a limited range, we refine the aforementioned depth by correcting the inherent error caused by the classification. To sum-up, our main novelties are: 1) A Yolo-based CNN that simultaneously produces a coarse depth map, together with bounding boxes for the scene objects; 2) A novel sub-pixel refinement method, that relies on genetic algorithms to correct the aforementioned coarse depth map; 3) Real-time results, capable to keep-up with state of the art depth estimation and object detection approaches.

2. YOLO-based simultaneous object detection and depth estimation

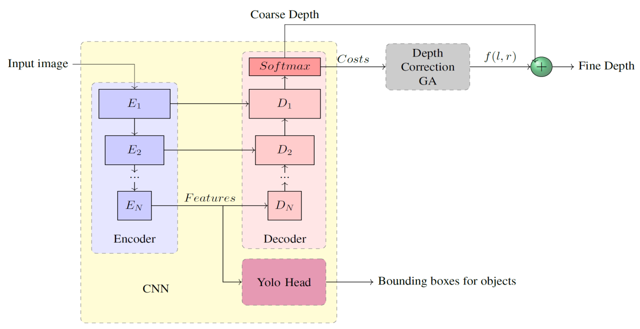


Figure 1. Architecture of the system

This section first introduces the overall architecture of our system, then it thoroughly presents the components of the neural network, its layers and the learning procedure. The system is composed by a CNN that simultaneously produces a coarse depth map and bounding boxes for object

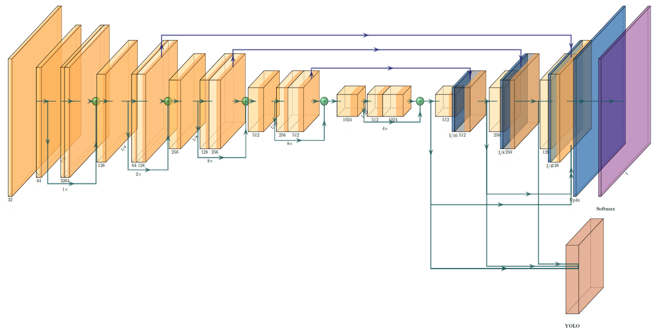


Figure 2. CNN for simultaneous Object detection and MDE

detection. The depth map is further improved by fitting a function to produce sub-pixel accurate results. The overall architecture of our system is presented in Fig. 1.

2.1. CNN architecture

As shown in Fig 2, the CNN consists of three parts: 1) a dense feature extractor; 2) a YOLO-based object detection head that outputs a set of bounding boxes for scene objects; 3) a depth estimation head, that labels each pixel inside the image with a depth value.

2.1.1 Feature extractor and YOLO

The first two parts follow the Yolo V3 model presented in [9]. The feature extraction is composed of 53 layers. Most of the layers consists of 3×3 and 1×1 convolutional layers. Skip connections are employed to preserve the image structure through all 53 layers. The resolution is decreased after each residual link, the number of features being doubled. The output of the feature encoder consists of 1024 features of size $I/32$.

In short, the feature extractor in YoloV3 makes predictions at 3 different scales, at resolution $I/32$, then upsampled by 2, then again upsampled by 2. These results are then passed to the Yolo layer, which accounts for object category and locality.

2.1.2 Depth estimation

The aforementioned feature extractor makes predictions at multiple scales, in a Feature Pyramid-like network. This is extremely useful for depth estimation, where both local information and global context must be provided for increased accuracy.

Once features are extracted, the decoder part of the architecture combines them by only using convolution and upsampling layers. The features used for detection, at $I/32$, $I/16$ and $I/8$ sizes are therefore combined with features extracted at same scales from the encoder. Finally, the output is upsampled 4 times, to match the input resolution.

The last layer generates a $im\_width \times im\_height \times depth\_classes$ 3D volume, where $depth\_classes$ corresponds to the total number of allowed depths. Finally, the last layer (softmax) assigns a probability to each possible depth value, for each pixel.

2.2. Learning

We initially train the network in the context of the object detection. The loss for object detection accounts for object type (classification), localization and confidence (the formula for L_{Yolo} can be seen in [8]). Once the object detection achieves the desired accuracy, the weights inside the encoder are ”frozen”. We then include the depth estimation part of the network, and train it without modifying the initial part According to [5], better results for depth estimation are obtained when this task is seen as a classification, rather than a regression. Therefore, instead of using mean squared error as our loss, we prefer to use the cross-entropy classification loss:

$$L_{Depth} = - \sum_{c=1}^{depth\_classes} y_{o,c} log(p_{o,c}) \tag{1}$$

where p is the probability that a pixel o belongs to class c, while y is 1 for the correct class and 0 otherwise.

Once the MDE part starts to converge, we start training the entire network, including both parts. At this point, the main task becomes to balance the two losses such that both outputs improve, benefiting one from another. The final loss becomes:

$$L_{final} = L_{Yolo} + \lambda \times L_{Depth} \tag{2}$$

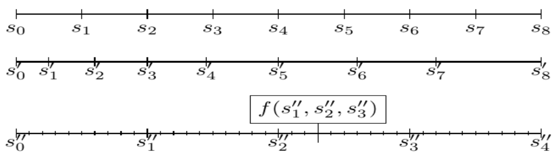
where λ is a parameter to control the influence of each particular loss. Exhaustive tests showed that the classification loss should be advantaged in this case, so setting $\lambda = 500$ provides the best balancing between the two tasks. We finally end-to-end train the network, such that it generalizes to both tasks.

3. Depth correction

3.1. Fine-discretization of the space vs interpolation

Our last layer produces only a sub-set of depth values, corresponding to the $depth\_clsses$ depths. The number of classes directly depends on the discretization of the depth interval given by the ground truths. For instance, Kitti [6] and Cityscapes [1] use 16 bits to embed their GT depth data, which results in 65536 depth possibilities. A large number of classes entails a large ”softmax probability” volume (we name it also a ”cost volume”), which affects the footprint of the GPU memory, thus the overall speed of the CNN.

There are several ways to discretize the spatial depth interval: by uniform discretization (UD – Fig. 3 up) – depth



s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8
s'_0 s'_1 s'_2 s'_3 s'_4 s'_5 s'_6 s'_7 s'_8
$f(s''_1, s''_2, s''_3)$
s''_0 s''_1 s''_2 s''_3 s''_4

Figure 3. Discretization (up and middle) vs Interpolation (bottom)

intervals are distributed equally along the entire space, or the one proposed by DORN [2] – using the log-space (Fig 3 middle). Such an approach is capable to more accurately predict small and medium depth values, which constitute the vast majority of points inside the dataset.

Although DORN method performs extremely well on benchmarks such as Kitti [6], the results it produces are still quite inaccurate especially for objects at large distances. Therefore, rather than focusing on smart ways to discretize the depth interval, we adopt a different approach. We initially compute a depth map using only a small number of classes. We then interpolate the correct value: the "fine" corrected depth should slightly drift from the chosen depth class.

3.2. Using neighbor depth values

As previously mentioned, the softmax layer in the CNN produces a $width \times height \times depth\_classes$ cost volume, assigning a probability for each position to belong to a specific depth class. Thus:

$$c_{softmax}(i,j,d) = \frac{e^{C_{conv}(i,j,d)}}{\sum_{k=1}^{n} e^{C_{conv}(i,j,k)}} \quad (3)$$

where $C_{softmax}$ is the volume generated by softmax, C_{conv} is the volume generated by the last convolution in the network and $depth\_classes$ is the total number of classes. During the evaluation phase, the chosen depth (coarse) is selected using the $argmax$ function.

For a finer depth map, the selected coarse value d must be corrected. Exhaustive tests showed that the although all possible depth classes receive a probability, the more relevant ones are the neighbor costs of the chosen class. Thus, we adapt the coarse value, such as:

$$d_{fine}(i,j) = d_{crs}(i,j) + f(c_{d-1}, c_d, c_{d+1}) \quad (4)$$

where c_d is the softmax probability at the chosen depth, while c_{d-1} and c_{d+1} represent the neighboring costs. We cleverly rewrite the function as being dependent on only one variable (by means of its symmetrical properties and other empirical observations). Thus, we rewrite (4) as:

$$d_{fine} = \begin{cases} d_{crs} - s\_f/2 + f(x) & \text{if } c_{d-1} - c_d \leq c_{d+1} - c_d \\ d_{crs} + s\_f/2 - f(1/x) & \text{otherwise} \end{cases} \quad (5)$$

where s_f is the sampling factor when choosing the $depth\_classes$ value for last layer $s\_f = \frac{max\_depth}{depth\_classes}$. Thus, the final goal is to find the optimal correction functions $f(x)$ such that the depth is properly corrected.

3.3. Function fitting

Our main goal here is to find the proper function that correctly redistributes the points to the desired GT class. The easiest way to create a valuable function is by learning. For this, we will represent the function as a binary expression tree (nodes as operators, leaves as variables), and use a stochastic optimization strategy based on genetic programming to evolve the optimal function.

We start with a random set of functions and optimize it by means of selection (keeping only the best functions from a generation), crossover (interchange branches between the best expression trees) and mutation (randomly changing nodes inside the tree). We use the following fitness:

$$f_{fitness} = \sum_{c=1}^{class} |median_{i=1}^{func\_no} f(i,c) - GT(c)| \quad (6)$$

where $func\_no$ is the number of functions in a population and $class$ corresponds to the number of classes when discretizing interval $[-s\_f/2...s\_f/2]$. We train on a randomly chosen subset of points obtained from the entire dataset, ensuring that the number of points inside each $class$ is constant. The best function we obtain by using this methodology is the following:

$$f(x) = sin(arctan(x \times \pi/2) \times \pi/2) \quad (7)$$

The function is bijective, taking values from the interval $[0..s\_f/2]$, giving values in the same interval and thus distributing the points to the correct (sub-pixel) depth.

4. Evaluation

4.1. Dataset generation

The main prerequisites for our training set are: 1) RGB images – obtained in driving scenarios; 2) Depth images acquired either from stereo (need left and right images) or LiDAR points projected to the image; 3) Ground Truth labels for all objects, including the coordinates to generate the bounding boxes. Cityscapes is the most adequate choice for us. It contains both left and right images (and a stereo disparity) and we generated the GT object labels. A different option would have been the Kitti dataset [6]. Although Kitti provides GT for both of the aforementioned tasks, we could not find common images to train both tasks simultaneously.

We used 1024x512 downsampled training images from Cityscapes (in order for the model to fit on the GPU). For training we used input images at multiple scales, additional augmentation being generated, the validation being done on

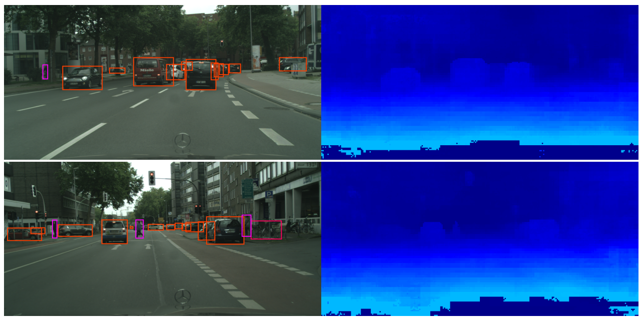


Figure 4. Results obtained with our approach: Input Image with bounding boxes for objects and the depth estimation

Table 1. Performance of various depth generation methods

Method	Mis.px.	absErr	Speed
SGM [4]	5.23%	9.78%	570 ms
DORN [2]	4.89%	7.84%	500 ms
Proposed (+c)	5.11%	8.05%	171 ms
Proposed (-c)	5.83%	8.95%	170 ms

the 500 images. The anchor boxes for Yolo are set according to the original article [8]. The CNN is trained by using the Darknet framework, on a Nvidia GTX 1080 Ti GPU.

4.2. Results

We visually show the detection results in left part of Fig. 4, while the right presents the inverse depth map (disparity) for several scenes form Cityscapes dataset.

In order to properly test the efficiency of our CNN-based depth computation, we test its accuracy and speed. Since our method computes depth as a stereo disparity (inverse depth), we test its results wrt a classic SGM [4] stereo method and a MSE-based CNN [2] (for which we used our own implementation), by computing its inverse depth. We include two variants of results obtained with our method, with and without the correction, for the case in which the sampling factor $s\_f = 1$.

As metric, we evaluate the speed, the number of mismatched pixels with an error threshold of 3 pixels $T = 3$ (classic stereo error), the relative absolute error, one of the usual MDE metrics and the speed. We show the results in table 1. Our approach produces results close to state of the art in MDE, the correction having a significant impact.

Since the architecture of the Yolo head was not modified, for detection we are only interested to see if any change in mAP (mean average precision) for YOLO was caused by our additional training. Therefore, in this case we only compare the mAP we obtain after we introduced the depth head, with the original one (Table 2). We show the results in the case of a minimum overlap of 0.5. Our method here uses a depth sampling factor of 2.

Table 2. mAP for Yolo versions

Sampling factor	Accuracy	Speed
Without depth	60.4%	47 ms
With depth	62.2%	81 ms

5. CONCLUSIONS

We have presented here a method to simultaneously compute a depth map and generate bounding boxes for moving objects inside an image. The method uses a common backbone for both tasks, over which two separate heads are applied. In case of object detection a Yolo-based layer generates the bounding boxes, while the other head assigns a depth value to each pixel through a classification method. Moreover, since the classification can only produce values in a limited range, we present a method the interpolates a sub-pixel value, correcting the initial depth. We performed multiple tests with best positive results for the proposed approach.

References

[1] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[2] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 1, 3, 4

[3] Ross B. Girshick. Fast R-CNN. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. 1

[4] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *PAMI, IEEE Transactions on*, 30(2):328–341, Feb 2008. 1, 4

[5] Ruibo Li, Ke Xian, Chunhua S., Zhiguo Cao, Hao Lu, and Lingxiao H. Deep attention-based classification network for robust depth prediction. *CoRR*, abs/1807.03959, 2018. 1, 2

[6] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2, 3

[7] V. Miclea and S. Nedevschi. Real-time semantic segmentation-based depth upsampling using deep learning. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 300–306, June 2018. 1

[8] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. 1, 2, 4

[9] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. 2