Combining Multiscale Features in Convolutional Neural Networks for Region Segmentation and Border Detection

Felipe A. L. Reis, Raquel Almeida, Ewa Kijak, Simon Malinowski, Silvio Jamil F. Guimarães and Zenilton K. G. do Patrocínio Jr.

***Abstract*—Boundary detection and region segmentation have been extensively studied for over 50 years, with several approaches. Latterly, machine learning, specially convolutional neural networks (CNN), techniques have proven quite effective in solving these problems. A recent improvement is to combine features generated in multiple network layers. Due to their architecture, CNNs produce different information along their layers, on a multiple scale, which, when combined, contribute to the final result with their own characteristics. Some works, aiming to increase performance, decided to train individual convolutional blocks to force this behavior, producing similar maps in muliple scales. This method, however, increases the cost and time of training, once it does not take advantage of information previously generated by correlated problems. This work seeks to propose and evaluated trivial techniques to combine features resulting from different layers of CNNs, without generate similar multiple outputs, to produce boundary detection and region segmentation. It evaluates the influence of the number of intermediate results extracted from the network (side-outputs) and what trivial operations, such as average, maximum and sum, can be used in those tasks. Also evaluated how to combine multiple multiple ground truths annotations, presented in some data sets, into a single reference map, to increase convergence. The creation of simple or even trivial methods favors the use in different scenarios, becoming a generalist method. The networks developed here were tested for region segmentation and edge detection tasks, with performance comparable to the literature, despite its simplicity. In the edge detection task, our best results reached 0.780 ODS on the well-known BSDS500 data set, at 44.8 FPS.**

***Index Terms*—Edge detection, border detection, boundary detection, region segmentation, image segmentation, convolutional neural networks, multi-scale learning.**

I. Introduction

Segmentation subdivides an image into its constituent regions or objects, while edge detection techniques identify the boundaries between objects [1]. An edge, according to [1], is defined as a set of connected pixels that border two regions.

In addition to [1]'s definition, [2] subdivide edge detection into boundary detection and the classical edge detection. For them, edge is "an abrupt change in some low-level image feature such as brightness or color" while boundary "is a contour in the image plane that represents a change in pixel ownership from one object or surface to another" [2].

[3] defines contours as a link between edges and a representation of region boundaries. Closed contours represents region boundaries and open contours are parts of region boundaries [3]. Contours can be combined into a UCM representation, an indexed hierarchy of regions as a soft boundaries [4].

Region segmentation, object and boundary detection are correlated tasks. Object limits with its high-level information can be used to define boundaries location. Also, boundaries can limit pixels into regions [2]. This correlation allows one problem to be modeled on another and effective techniques can be shared between them.

Traditional segmentation methods use basic image properties to find discontinuities that can indicate objects, regions or borders. However, these techniques face practical problems, such as similar color patterns, noise, and multiple thresholds for groups of images. To solve this, nowadays, its common to use deep learning supervised methods (especially CNNs), in which image characteristics are learned by algorithms [2] [5].

Due to their architectures, a common CNN produces different information along its layers. Each layer contributes specifically to the final result, with its own characteristics, that indicates how weights are adjusted by the network. Furthermore, the network produces information in different sizes along the layers, due convolutional and pooling operations [6] [7] [8] [9].

In order to combine these information at scale, some works such as HED, RCF and BDCN trained convolutional blocks individually, with specific loss functions for each of them. This procedure aims to produce macro and detailed information, which are later combined into a single map [10] [11] [12].

This work seeks to simplify this idea, proposing and evaluating some simple network architectures, based in some recent papers. Also, it will be evaluate the capacity of obtain good results without forcing layers to produce similar information.

To evaluate the methodology, our work will show results obtained in the tasks of region segmentation and edge detection. The task of region segmentation was addressed in our previous work [13] and, at this stage, we will focus mainly, but not only, in edge detection.

In edge detection we will refine the results of our previous work in a more difficult task, that allow us to check the best method. Also, this paper will address how to increase training speed and performance with simple transformation in the ground truth of edging detection.

II. Related Work

Edge detection has been researched over the last 50 years, with several proposed algorithms [14]. It was used in the first

segmentation works, which later also began to identify regions. In parallel to the edge detection works, specific algorithms for region detection and pixel classification were developed [15].

The first works of edge detection, in the 1960s and 1970s, used color and shades intensity variation to identify horizontal or vertical differences, corresponding to the gradients. In the following decade, works used Laplacian of Gaussian operator for smoothing and edge detection. In the 1990s and 2000s, techniques were developed using manually created features, who developed methods based on gradients, color intensity and textures [15]

In the current decade, works with manually developed features were also proposed, such as the Sketch Tokens [16]. However, new approaches have emerged using machine learning, as *Structured Edges*, which uses decision forests [17]. At the same time, some studies have been carried out using deep convolutional networks, following the success of neural networks for object detection [5].

DeepContour [18] and HED [10] networks, are one of the pioneers in the use of deep learning to produce edge detection. DeepContour uses its own network, in a traditional way, with convolutional layers in sequence. HED developed a new approach, adding side-outputs in the well known VGG16 [6], to produce intermediary edge maps.

HED removed the VGGNet's classification layers and added side-outputs on the last convolutional layer of each stage. Also forced all side-outputs to produce similar predictions, using loss functions for each of them. Then, these outputs have been combined to generate a final edge map output [10]. The changes become the state of art in BSDS500 data set [19] and was adopted by other authors.

Following HED, [20] defined an architecture 3 times bigger to combine outputs in three different scales, merged into a single map. [21] captured relaxed labels from simple detectors that were merged to the ground truth, and use it to supervise the training and remove some hard-to-classify false positives (relaxed labels). [22] used Semantic Segmentation and Domain Transform (DT) to improve predictions.

Using ResNet [23] as its based architecture, COB [24] generated 8 distinct side-outputs in each layer, corresponding to different direction angles of the convolution operation. It also generates two outputs with images at different scales, to produce fine and coarse detections. All them are combined in a fusion layer, producing a single Ultrametric Contour Map.

Contrary to the tendency of side-outputs, [25] used a simple convolutional network followed by a morphological thin operation to increase accuracy while [26] also used a simple 3-layers convolutional architecture to predict edges, followed by a contour refinement post processing, that transform coarse contours into fine-scale ones. Both works did not reach the results of other side-output based methods.

Following the side-output approach, it was observed by [27] that HED predictions produces some blurry edge maps. To avoid "crispness" of the boundaries (precisely localizing edge pixels), it was developed a new architecture, called Crisp Edge Detector that create a backward-refining pathway, which increases the resolution of intermediary maps progressively [27]. The border crispness was also observed by [28], that proposed refinement modules to reduce them.

Some level of refinement was used also by [29]. It was proposed the usage of Attention-Gated Conditional Random Fields (AG-CRFs) to refine and fuse the representations learned at multiple scales. Attention mechanisms were integrated, into a two-level hierarchical CNN model, to produce multi-scale learning, in a custom network for contour detection [29].

[30] proposed a network to create a multi-scale feature graph. It used an architecture based on U-Net [31] and ResNet [23], with skip connections in order to avoid gradient vanishing from low level features. The work build an end-to-end Deep Symmetrical Metric Learning network (also know as Encoder and Decoder Network) and a Metric Learning to combine side-outputs from each network layer [30].

An Encoder and Decoder Network was also proposed by [32]. In this approach, it was developed a Generative Adversarial Network (GAN) for edge detection. This model is composed by a encoder-decoder model to extract edge information and a discriminator, a classification network that distinguishes the generated contours from the ground truth [32].

A more traditional work is RCF [11], an HED-based project that developed side-output network versions of VGGNet and ResNet. Its architecture contains side-outputs after each convolution. At each stage, the outputs are combined into a single output with the sum of predictions. To improve the results, it is also made a process of scaling the images that enter the network in 3 different levels of detail. The results are then combined to generate a single map of borders.

Assessing the tendency in previous work to sum isolated side-outputs, with uneven qualities, [33] proposed C-Net Cumulative Network, that aims to learns cumulatively based on visual features and low-level side-outputs and gradually remove detailed and sharp boundaries. The authors indicate that this procedure allows more accurate edge detection, since superfluous results are progressively abandoned while the network learns [33].

Similar as [33]'s work, [34] proposed an edge detector based on feature re-extraction (FRE). It contains side-outputs in different stages, which are combined in feature re-extraction blocks. Each block contains three convolutional layers, a batch normalization and a residual operation, that are combined to produce a final stage output. The stage outputs are combined using a convolution 1×1 fusion module [34].

The Hybrid Convolutional Features (HCF) network [35] defined a "CNN-based pipeline" that combine multi-level features, producing a probability map. To do it, it creates auxiliary branches, new convolution, normalization, up-sampling and concatenation layerts to increase performance. Its proposed architecture is called HybridNet, and combine VGGNet and ResNet architectures [35].

Bi-Directional Cascade Network (BDCN) network, proposed by [12], explore multi-scale features for edge detection. The model differ from previous works due it custom supervision of labeled edges at its specific scales, instead of using the same supervision in all CNN outputs. Also, it uses

dilated convolution to generate multi-scale features, named Scale Enhancement Module (SEM), rather of explicitly fuse multi-scale edge maps [12]. This work reached the state of the art in the BSDS500 data set [36].

The analysis of related works provides important information regarding some trends in neural networks for edge detection. One trend is that side-outputs have been widely used in edge detection works in recent years. In addition, the results indicate that at least the 5 best ODS values in the BSDS500 were achieved by methods that combine intermediary features.

Another important information is related to the number of papers that developed their own loss functions. For training their network architecture, 11 of 18 works developed its own custom loss. This is observed due to the characteristics of the problem, where there is an imbalance between edge and background pixels. According to [34], approximately 83% of the pixels in the original BSDS500 testing images are non-edge pixels.

Traditional loss functions does not help methods to identify edges, as they tend to learn only background pixels, producing highly accurate but unwanted results [10]. Then some papers started to create its own functions, as Class-balanced Cross-entropy, proposed by [10] and also used in [33] and [28] papers, that aims to balance edges and non-edges pixels.

Some works, like [20], [24], [11] and [34], adapted class-balanced cross-entropy to create their own versions, exploring parameters improvement or addind some extra features. Others, like [35] and [12], used it with other auxiliary losses to help to extract high-level features. These auxiliary function aims to increase performance in side branches of the network.

These new loss functions have the additional objective of reducing the number of training iterations. Some methods needs a lot of iterations to converge properly and achieve its best performance. Some methods with good results, like [11], [32] and [12] (state-of-art) needs near 40k iterations, while others, like [30] needed 110k.

III. Method

In this paper we will evaluate and create some different network architectures, inspired by HED [10] and RCF [14] networks. All architectures uses VGG16 [6] neural network with multiple side-outputs, in order to obtain multi-level features.

A. Neural Network Architectures

Once our work is based on VGG16, we will address its architecture. VGG16 is made up of 13 convolutional layers, divided into 5 stages, with 2 or 3 convolutional layers. It uses a rectified linear unit $ReLU(\cdot) = max(0, \cdot)$ as an activation function and a max-pooling layer. At the end of the network, it has a fully connected layer, a ReLU layer and a final softmax activation function layer, totaling 16 weight layers [6].

To build our architecture, the last activation layer and all the fully connected layers of the VGG16 network were removed. Also it was added side-outputs in different quantities, for evaluation. Details regarding these architectures, influence on training and accuracy will be discussed in Sections III-B and V-A. An overview of the architecture can be seen in Figure 3.

VGG16 was chosen due its overall performance and its relatively simplicity to create, train and study the influence of side-outputs, being suitable for our experiments. Creating, training and analyzing the influence of outputs is a simpler task than in residual networks such as ResNet [23], despite its higher performance.

B. Side-outputs

Following a current trend, our work used side-outputs, like several other existing approaches in the literature, as described in Section II. To evaluate the most appropriate number of side-outputs, it was analyzed HED and RCF architectures. HED creates outputs after the last convolutional layer of each stage (5 outputs), while RCF adds side-outputs after each of the convolutions (13 outputs) [10] [14].

The first strategy, named *Stage Layer Outputs* (**SLO**) and inspired by the HED model, creates a side-output $\mathcal{H}_i$ for each stage S of the network. Its graphical representation is available in Figure 1. The second strategy, named *All Layers Outputs* (**ALO**) and inspired by the RCF architecture, creates a side-output $\mathcal{H}_i$ for each convolutional layer L, as detailed in Figure 2.

In SLO, the number of side-outputs corresponds to the number of network stages $|S|$. In turn, in the ALO method, the amount of output equals the number of convolutional layers $|L|$. Formally, the set $\mathcal{H}$ of m side-outputs maps in each strategy is defined as:

$$\mathcal{H}_{SLO} = \{\mathcal{H}_1, ..., \mathcal{H}_m \mid m \leq |S|\} \quad (1)$$

$$\mathcal{H}_{ALO} = \{\mathcal{H}_1, ..., \mathcal{H}_m \mid m \leq |L|\} \quad (2)$$

Each side-output $\mathcal{H}_1, ...\mathcal{H}_m$ is composed of a 1×1 convolution, followed by a transposed convolution of variable size. Due to the network architecture, with *pooling* layers, the images are reduced along the network. Thus, for the images to be rescaled to a single size, it is necessary that the transposed convolutions have a variable size.

C. Side-outputs Merging Techniques

Working with side-outputs requires the usage of methods to combine intermediate results. These are generated at different scales and may represent different concepts due to the position of the output on the network. The goal, then, is to produce a single proposition $\hat{Y}$ to be evaluated in the task, keeping useful information contained in different layers.

In this work, a new simple strategy is developed to overcome the challenge of combining the outputs by exploring the knowledge of the learning process. Operations are performed element-wise at each side-output. To do this, the initial step is to resize the images generated at different stages of the network, so that they all have the same size. Then, different trivial merge functions were studied and compared, once each of these has an expected behavior, as described below:

- *ADD*: sums intermediate results, aiming to balance negative and positive weights;

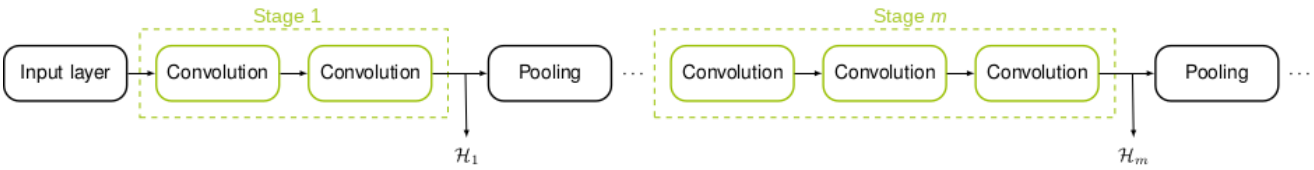


Fig. 1: SLO Neural Network Architecture.

Fig. 2: ALO Neural Network Architecture.

- *AVG*: calculates the average value of the outputs to create a proposal that represents learning at all levels;
- *MAX*: returns the maximum value between the results, aiming to value the most confident outputs.

Formally, the combined output $\mathcal{Z}$ in each strategy can be defined as:

$$\mathcal{Z}_{ADD} = \sum_{i=1}^{m}(\mathcal{H}_i) \tag{3}$$

$$\mathcal{Z}_{AVG} = \frac{\sum_{i=1}^{m}(\mathcal{H}_i)}{m} \tag{4}$$

$$\mathcal{Z}_{MAX} = \max_{1\leq i\leq m}(\mathcal{H}_i) \tag{5}$$

D. Merged Output

After combining the side-outputs as detailed in Section III-C, a single final prediction $\hat{Y}$ is produced. The value is obtained by executing a convolutional operation of 1×1 followed by a ReLU activation function. The method overview is illustrated in Figure 3.

In region segmentation tasks, $\hat{Y}$ seeks to provide partitions that represent significant areas of an image, while in the edge detection tasks, it seeks to represent the boundary between regions.

E. Class Balancing Methods

Edge detection seeks to identify abrupt change in an image aspect, like color or brightness [2]. In this problem, the number of edge pixels is considerably less than objects and background pixels. Due to class imbalance, machine learning methods have difficult to identify edges. Without proper treatment, the classification returns only backgound pixels, with high confidence.

To help to solve this problem, we decided to use Focal Loss, proposed by [37], aims helps to solve problems with high imbalance between classes. It is a modification in Cross Entropy Loss Function for binary classification, that seeks to devalue pixels that are easy to identify and to value pixels that are hard to classify [37].

IV. Organization of Experiments

A. Data sets

The methods present here were evaluated in the following data sets, widely used and cited in the literature:

- KITTI Road/Lane: part of the Karlsruhe Institute of Technology and Toyota Technological Institute project, it provides a set of images, and their respective human-annotated ground truths of the roads and lanes [38];
- BSDS500: the *Berkeley Segmentation Data Set and Benchmarks 500* contains a set of natural images and their human-annotated ground truths for image segmentation and edge detection research. [19];

B. Evaluation Tools and Benchmarking

The quality evaluation of the results were made using benchmarks of the own data sets. The following benchmarks were used:

- KITTI Road/Lane Evaluation: benchmark of the KITTI suite, estimates roads and tracks based on birds-eye-view space. It returns *f-measure*, precision and recall values, false positive and false negative rates. [38];
- BSDS500 Benchmark: BSDS500 benchmark for segmentation quality assessment. It uses precision and recall methods to classify results of segmentation and boundaries [19];

C. Experimental Setup

Our networks were built using using Keras [39] with Tensorflow [40]. We used a pre-trained VGG16 model to initialize the weights. Training experiments were performed in GeForce GTX 1080 8GB GPU[1].

V. Region Segmentation Experiments

The first set of experiments aimed to analyze the feasibility of the method described in Section III. For an initial analysis, the region segmentation task was considered more appropriate, as it requires a lower level of precision than edge detection. Also, it was evaluated the influence of the number of side-outputs and the methods to combine them.

[1]Some experiment were performed using a GeForce GTX 1660 6GB GPU.

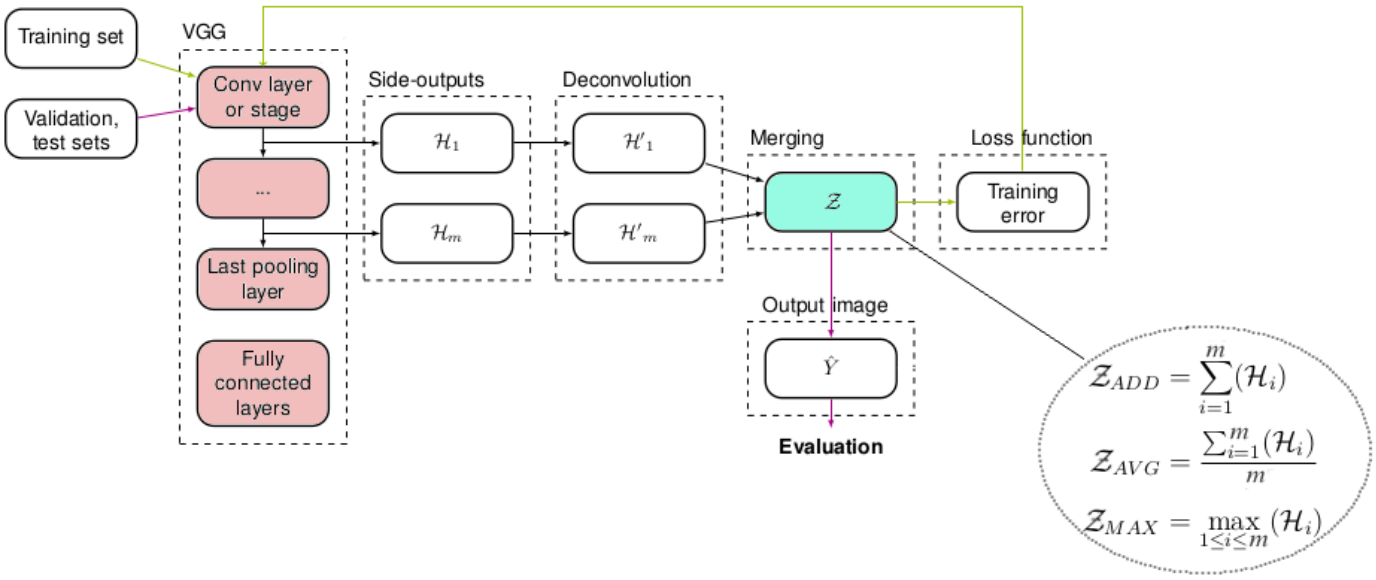


Fig. 3: Overview of our method.

The experiments were conducted using the KITTI Road/Lane data set. This data set is composed of 289 training and 290 test images with size of 1242×375 pixels. Ground truth is manually annotated for two different pixel types: (i) **road** is the area of the road that makes up all lanes; and (ii) **lane** is the ego-lane where the vehicle is currently in [38].

KITTI data set contains only ground truth for training data. The test set is evaluated using the KITTI Server Evaluation. In this work, only road markings were considered and images for lane segmentation were ignored. The ground truth of roads is divided into three categories: (i) urban unmarked (**uu_road**), (ii) urban marked (**um_road**) and (iii) urban multiple marked lanes (**umm_road**) [38].

To increase the number of images in the training set, data-augmentation techniques were used. The following transformations have been applied: salt/pepper noise, noise shadow, horizontal inversion (mirroring), contrast and brightness change, random rain/snow addition. Data augmentation procedures that create unwanted behavior, such as vertical inversion and distortions that would change the nature of objects in the scene, such as cars and pedestrians, have been avoided. Enlargement procedures resulted in 2601 images, divided into 2080 training samples and 521 validation samples (about 20%).

The initialization of weights was based on a pre-trained VGG16 model[2]. In addition, we use Stochastic Gradient Descen (SGD) optimization with 1×10^{-3} learning rate, 5×10^{-6} learning decay and 0.95 momentum. To tune the network and speed up the process, all images have been scaled down to 624×192 pixels (about 50%). The default batch size had 16 images.

[2]VGG16 model was trained using ImageNet dataset [41].

A. Influence of merging methods and the number of side-outputs

The first test of the current experiment was designed to identify the influence of the number of side-outputs and the best merging methods. For this, it were used **ALO** and **SLO** networks, detailed in Section III-B. The networks have been trained for only 100 epochs, using each merging methods. It was also used a modified VGG16 version to predict regions, without side-outputs, as baseline, called in this work *No Side-Outputs* (**NSO**).

Figure 4 shows the loss curves during the training phase for the validation set. It can be observed that the ALO networks appear to be more stable, with a steeper loss curve than the NSO and all SLO approaches. In addition, NSO and SLO-MAX networks were highly unstable during learning phase, with abrupt changes in the loss value, resulting in graphical spikes. On the other hand, the ALO-AVG network had the best result, followed by ALO-MAX and ALO-ADD. This is possibly caused by the considerably higher number of outputs, which creates the possibility of exchange between confident values.

To improve results, a new round of testing was created with 500 training epochs. As SLO networks showed poor performance in the previous test and other tests performed with different parameters, then it was decided to evaluate only ALO networks. The network NSO was also trained as a comparison *baseline*.

Performance analysis was done by two different metrics: the well-known *Categorical Cross Entropy* and *Pixel-error* metric. Pixel-error was adopted when high precision values were observed in results where there were noticeably many errors, especially with numerous false positives. All versions of network ALO outperforms NSO network using both metrics.

The results of the merging techniques of the ALO networks have similar value, indicating the absence of a considerably

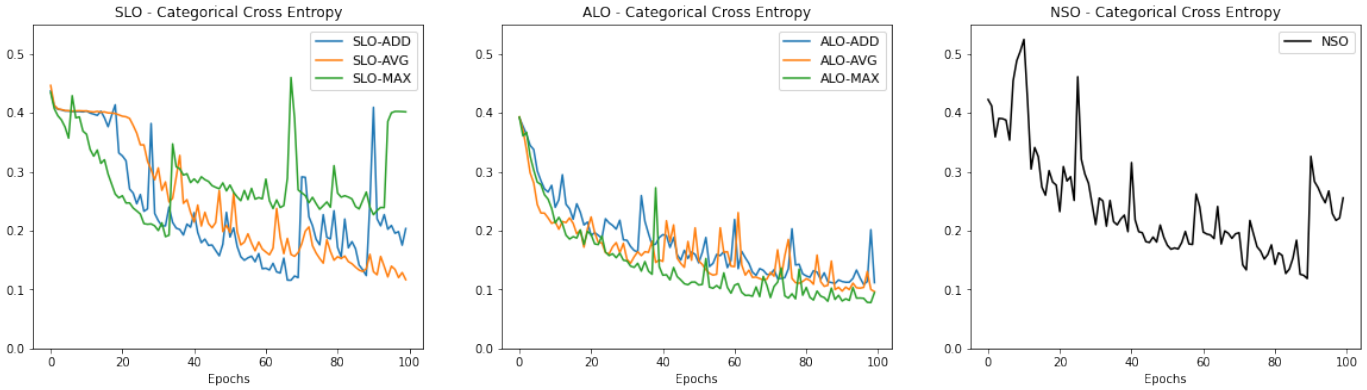


Fig. 4: Validation loss using *Categorical Cross Entropy* metric in first experiment.

Benchmark	MaxF	AP	PRE	REC	FPR	FNR
UM_ROAD	91.15%	83.82%	89.07%	93.33%	5.22%	6.67%
UMM_ROAD	94.05%	90.96%	94.82%	93.29%	5.60%	6.71%
UU_ROAD	89.45%	79.87%	85.40%	93.90%	5.23%	6.10%
URBAN_ROAD	92.03%	85.64%	90.65%	93.45%	5.31%	6.55%

TABLE I: Segmentation performance on KITTI Road Dataset.

superior method to the others. The best result for *Categorical Cross Entropy* metric slightly outperforms the worst method (0.983 for ALO-ADD and 0.9821 for ALO-AVG). Using *Pixel-error metric*, the best value is only 0.004 higher than the worst (0.0332 for ALO-AVG and 0.0372 for ALO-MAX).

Removed Side-Outputs Contribution (Exists Edge Detect)

B. Post Processing

At the end of the segmentation, a post processing step was added to reduce some noise. For this, the *morphological opening* operation was used to remove small noises created by the foreground (road) in the background. The operation was applied using a 13×13 square structuring element as a result of empirical tests.

C. Segmentation Evaluation

After the post-processing step, segmentation quality was evaluated. The tests were performed using the ALO-AVG method, the best in the training phase. Results were sent with name of **ALO-AVG-MM**[3] to the KITTI Evaluation Server, that returned the following metrics: MaxF, AP, PRE, REC, FPR and FNR, whose results for each road scene category[4], available in Table I.

Compared to the best result on the KITTI platform named PLARD [42][5], our method had an overall performance 5.0% lower. It is also necessary to remember that the model was trained for only 500 epochs. Thus, it is indicated that the techniques for combining side-outputs had good results, as desired.

To show the performance of our model, Figure 5 was created with ALO-AVG-MM predictions[6]. Pixels have been labeled as true positives (*green*), false negatives (*red*) and false positives (*blue*). The original images were created by the KITTI Evaluation Server, based on the binary map sent to the server.

VI. Border Detection Experiments

The second set of experiments aimed to evaluate the method performance in edge detection, a more complex and precise task than region segmentation. BSDS500 was chosen due its small size, relative simplicity and its wide use in the literature, which makes it suitable for comparison to the literature.

BSDS500 consists in a set of 500 images with size of 480×320 pixels[7]. The set is explicit subdivided into training, validation and test, containing 200, 100 and 200 images, respectively. Ground truths are manually annotated by five different subjects on average. It contains ground truth for region segmentation and border detection [19].

Unlike KITTI, BSDS data set does not contain pixels of interest (a.k.a. edges) distributed in a specific position of the image, making training difficult. Also, the number non-edge pixels are considerably bigger than edge pixels [34], becoming necessary to use methods that handles unbalanced classes, as discussed in the Section III-E.

BSDS500's test set is evaluated using BSDS500 Benchmark, provided by the authors [19]. In our experiments, training and validation sets were merged into one group, in order to increase generalization. The test set was only used to evaluate the performance of our results.

To increase training performance, it was used some data augmentation techniques, as salt and pepper noise, log and gamma contrast, JPEG compression, dropout, affine and perspective transformations, crop, pad, among others. Rotation,

[3] Results are available in a public leaderboard at http://www.cvlibs.net/datasets/kitti/eval_road.php

[4] *URBAN_ROAD* category corresponds to the combination of the other three categories.

[5] At the time of our first conference paper submission, [13], *PLARD* had not yet been published in the literature, being an anonymous submission on KITTI platform.

[6] More visual results are public available on KITTI Server.

[7] For boundary detection, images contains a extra pixel (481×321 or 321×481)

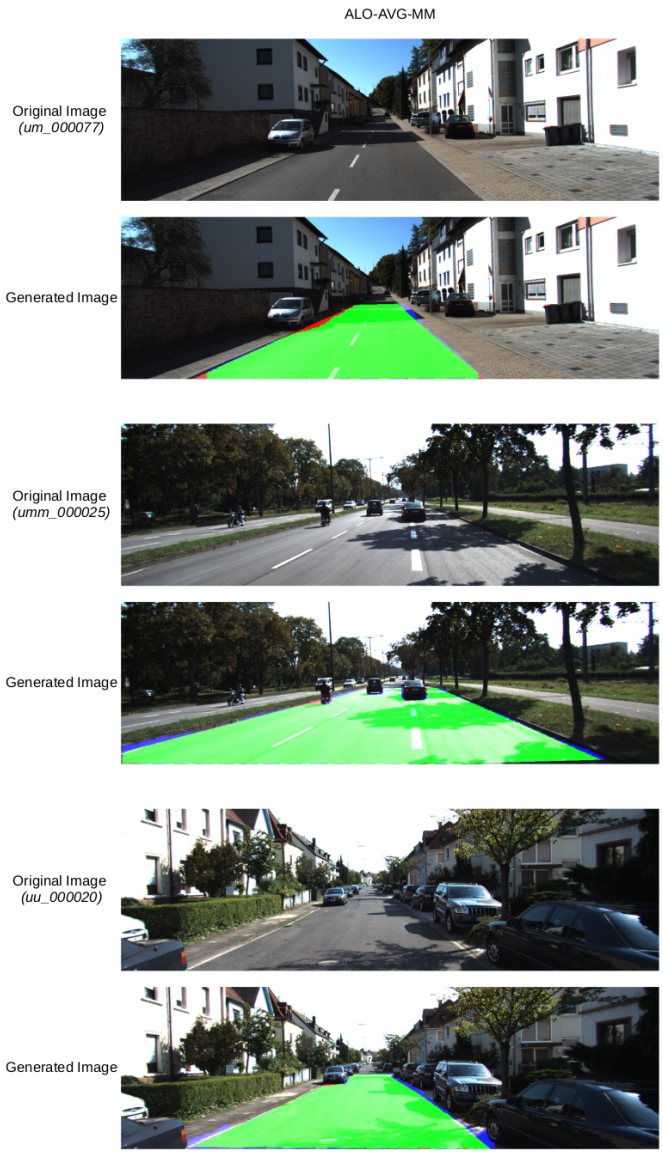


Fig. 5: Visual representation of ALO-AVG-MM predictions

flipping, mirroring and zoom (with factors of 0.5, 1.0 and 1.5) were combined with the techniques previously described. All procedures resulted in 11400 images (38 samples per image).

As previous experiments, the ground truth was separated into two binary classes, corresponding to the background and the edges. However, after training, the system was changed to maintain classes, but to generate results based on confidence, in the range of 0 to 1, and no longer in binary form. This method allows the identification of soft borders, as will be explained in the Section VI-A.

Of all the experiments carried out, only a few important ones were detailed in this next subsections. The analysis of experiments are summarized in Section VII.

A. Ground truth analysis

As briefly reported previously, BSDS500's ground truths are manually annotated by five different subjects on average [19]. Due multiples ground truths, some experiments were conducted to define how to combine them, in order to help the network to learn properly. To do that, we developed and studied 3 different methods to combine them.

The first ground truth representation, named **ALL**, added all ground truths and divided them by the maximum of ground truths' overlaps in each image. Ground truths borders were represented in a interval of 0 to 1, where the *1-value* represents borders annotated in all ground truths, while the *0-value* represents background in every annotation. Values in between, correspond to soft borders, annotations marked in one or more ground truths but not in all them. This representation is shown in Figure 6b.

This method had two major problems:

1) As the number of ground truths for an image increases, some borders has really small values, almost close to the background value;
2) Some borders are really close to others, but they do not overlap. So, the ground truth shows two different boundaries, which can make training difficult.

One possible solution to fix the first problem is to limit boundaries to the interval of 0.5 to 1.0. Background pixels are defined with value zero. Once this method limited border in the upper range of values, it was named **UPPER**. This ground truth representation method is available in Figure 6c.

Although benefits of the first approach, the second problem is not solved by them. To settle it, an morphological set of operations were proposed to join close borders. The chosen method corresponds to the dilatation of the boundaries using a kernel of 3×3, followed by a morphological thinning. Dilation helps to join borders, but produced some shadows, that are removed by morphological thinning.

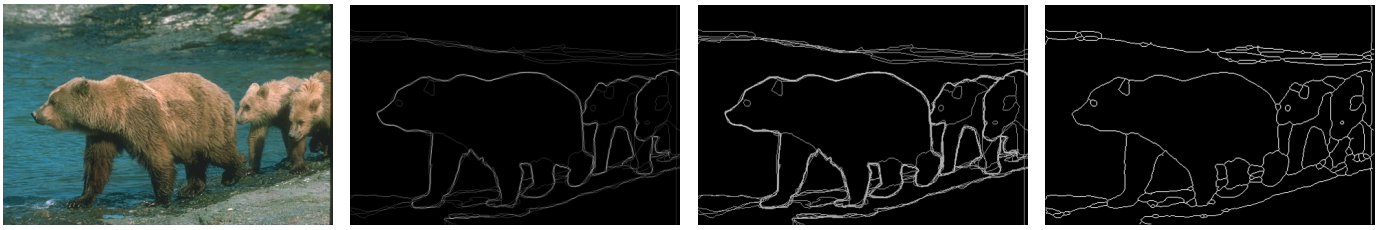
However, once morphological thinning is a binary operation, this method removes difference of values between borders, transforming soft borders into "hard borders", with values equal 1. The visual result of the morphological operations, named **MORPH**, is available in Figure 6d.

Some initial experiments were performed to evaluate the three approaches. It was observed that training the network using only ALL ground truth resulted into poor performance, due difficult of generalization. The best generalization for early stage training was produced using MORPH ground truths. However, ALL and UPPER ones, applied after a some MORPH training, produced better visual results.

B. Protocol for experiments

As described in Section VI-A, the usage of ground truth *MORPH* in the early stages of training achieved the best generalization. After it, the usage of *ALL* and *UPPER* ground truths improves achieved results. For this, we developed the following protocol, with the following phases:

1) Train only deconvolutional layers with MORPH ground truth;
2) Train the whole network with MORPH ground truth;
3) Train the whole network using ALL or UPPER ground truths;



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

Fig. 6: (a) BSDS500 image and its ground truth, using (b) ALL, (c) UPPER and (d) MORPH representations.

The first step starts with pre-trained VGG16 model. The results produced by first training phase, in terms of loss value, will be used by the following step. Likewise, the second step produces the weights for third training phase. At the end, the weights produced in the last phase are used to evaluate the network performance.

The following parameters set as standard in our experiments with BSDS500, unless otherwise specified:

- Learning method: SGD
- Activation: ReLU
- Learning rate: 1×10^{-6} (first phase) and 1×10^{-7} (others)
- Decay: 1×10^{-10} (first phase) and 1×10^{-11} (others)
- Batch size: 8 images
- Merging method: ALO
- Loss: Focal Loss (γ=2.0; α=0.25)[8]
- Initial weights: VGG-16
- Training samples: 9690 images
- Validation samples: 1710 images
- Image size: 384×288

In addition to the parameters listed previously, it was decided to use some functions to automatically reduce the learning rate and finish the training procedure after a number of epochs without any improvements. Learning rate (LR) is reduced by factor of 0.3 ($LR \times 0.3$), after 10 epochs, if the loss does not decrease, in absolute value, at least 0.5 units[9].

After loss decreasing, the code waits for 5 epochs (*cooldown*) and the procedure repeats until the learning rate achieves 1×10^{-9}, when it stops decreasing. Concomitantly with learning rate reduction, the early stopping procedure finishes training when the absolute value does not decrease at least 0.5 units after 50 epochs. If early stopping is not reached, the code will be completed after 1000 training epochs.

C. Experiment 2.1 - Basic experiment

To assess the network's performance, training was carried out with each version of the ALO network. The training presented here follows the protocol and parameterization described in Section VI-B.

The current experiment will be described into four subsections. The first three ones will describe each network behavior in phases using MORPH ground truth. The last section will compare final training results.

[8] Parameters recomended by [37] paper.

[9] This technique is also called as Reduce Learning Rate on Plateau.

1) ALO-ADD: The network quickly converged from pre-trained weights and after about 130 epochs, it has stabilized, with small performance gains. After no gain for 50 epochs, the protocol ended its first phase.

In second phase of the protocol, the network had a sharp decay in the training loss. Contrarily, validation loss increased (this condition will be carefully analysed in Section VI-D). Close to the 500th epoch, the network showed again poor training loss decrease and it was finished in the 567th epoch.

2) ALO-AVG: The network promptly converged, with a behavior similar to ALO-ADD. After near 140 epochs, the network has stabilized, with small performance gains. The protocol has ended its first phase (only deconvolutions training) at the epoch 219.

In the second phase, the training loss began to decrease slowly, but the validation loss increased, also slowly, for most of the process. This increase is similar to the training of ALO-ADD, even if it is less than that presented in the previous experiment. It is important to note that this behavior occurred even with small learning rates (less than 1×10^{-9}).

3) ALO-MAX: Contrarily to other networks, ALO-MAX had more difficult to converge properly in preliminary tests. It had the lowest loss (722.39), 24.06% more when compared to ALO-ADD (582.29) and 20.29% more when compared with ALO-AVG (600.54).

The validation loss was really unstable during both phases, with many ups and downs, without any convergence in the process. The training was faster than the previous ones, with 94 epochs in the first phase and only 69 epochs in second one. This behaviour indicates that the network did not improved its results and the training of the deconvolution layers was enough for the network to reach the best possible value.

D. Evaluation and Predictions

Due to the low capacity to evaluate the previous experiments using the accuracy metric, it was decided to evaluate the model with BSDS500 Benchmark. The results are presented in Figure 7. To evaluate the gain of UPPER and ALL ground truths usage, it was decided to evaluate MORPH performance also. As explained in Section VI-B, the network was trained until it does not improve for 50 epochs, making MORPH almost unable to perform better.

The overall results of ALO-ADD and ALO-AVG were really similar, considerably better than ALO-MAX. The number of training epochs to reach the best results were also similar, as

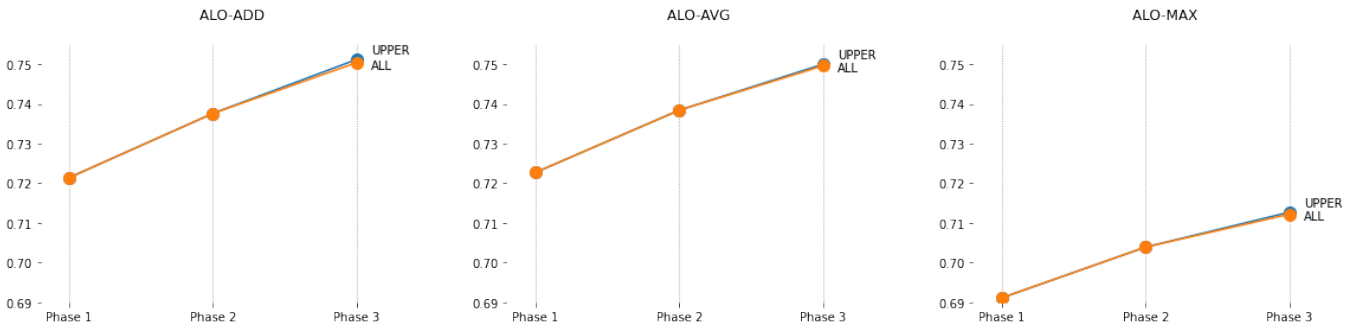


Fig. 7: Border detection ODS performance on BSDS500 for ALO-AVG, ALO-ADD and ALO-MAX in Experiment 1.

TABLE II: Number of training epochs in Experiment 2.1.

Ground truth	ALO-ADD	ALO-AVG	ALO-MAX
MORPH / ALL	666	641	372
MORPH / UPPER	697	635	361

described in Table II. These results indicates that both methods are similar in performance and number of training epochs. In contrast, ALO-MAX had the least number of training epochs. However, due to its poor performance, this information cannot be used in favor of the method.

It is also possible to notice, in the analysis of Figure 7, that all networks improved with training using ALL or UPPER ground truths. The benefit of using both ground truths is around 1,5% in comparison with the best result achieved by MORPH ground truth.

These results can be partially explained due to the deformation caused by the morphological thinning operation. UPPER and ALL ground truths maintains the original outline of the edges. Also, the differences can be seen as just an adjust of the weights, to provide hard and soft defined borders. This difference seems to influence the evaluation, once the benchmark compare borders with each human annotation in each ground truth.

After this experiment, it was decided that the ALO-MAX method will no longer be tested, due to its low performance and weak convergence. The following sections will provide and discuss experiments using only the ALO-ADD and ALO-AVG methods.

E. Experiment 2.2 - Improving Focal Loss and Hyperparameter Tuning

After training BSDS500 using Focal Loss, a new experiment was developed to try to increase the results achieved and decrease the number of training epochs. To do it, a simple change in Focal Loss is proposed in this section. The suggested change is to add the metric Pixel-Error (PE), as a factor to improve Focal Loss (FL).

The purpose of the modification is to increase the separation between the edges and the background, making the network convergence faster. It is important to clarify that this change will mainly affect the ground truths MORPH and UPPER, that contains edge values far from background values. The new

TABLE III: Border detection performance on BSDS500 for ALO-ADD and ALO-AVG in Experiment 2.2.

Network	Loss	Ground truth	TH	ODS	OIS
ALO-ADD	PEFL	ALL	0.21	0.7558	**0.7776**
ALO-ADD	PEFL	UPPER	0.30	0.7559	0.7746
ALO-AVG	PEFL	ALL	0.20	0.7546	0.7770
ALO-AVG	PEFL	UPPER	0.30	**0.7563**	0.7749

TABLE IV: Number of training epochs in Experiment 2.2.

Ground truth	ALO-ADD	ALO-AVG
MORPH / ALL	553	409
MORPH / UPPER	620	473

metric, named PEFL (*Pixel-Error Focal Loss*), can be simple defined as Equation 6:

$$PEFL = FL \times (1 + PE) \tag{6}$$

In conjunction with the new loss function, some hyperparameter tuning were made[10]. Some quick experiments were carried out and the one that showed the greatest evidence of improvement was the reduction in *gamma* parameter. Then, ALO-ADD and ALO-AVG networks were fully trained with the following changes in default parameterization, defined in Section VI-B.

Loss: PEFL (γ=1.0; α=0.25)

In the new experiment, ALO-AVG had, in all phases, a similar behavior to the network ALO-MAX in the experiment described in Section VI-C. This unusual behavior, nonetheless, did not increase the number of training epochs nor decrease the performance. Instead, it performed better than that obtained in the experiment described in Section VI-C, using the default parameterization, as can be seen in Table III.

Table IV shows that the number of training epochs has decreased compared to the results of the previous training, available in Table II. Comparing the number of epochs of both experiments, it is possible to perceive that ALO-ADD reduced 16.97% in MORPH/ALL method and 11.05% in

[10] The experiment described in Section VI-C was performed again, changing only the loss function. It produced, using ALL and UPPER ground truths, 0.7502 and 0.7505 ODS values, with 429 (36% less) and 460 (34% less) epochs, respectively.

MORPH/UPPER method. ALO-AVG performed even better, with reduction of training epochs for MORPH/ALL 36.19% and 25.51% for MORPH/UPPER ground truths. It is important to remember that both networks have increased their performance when compared to previous experiments.

F. Side-Outputs Contribution

After training the network in previous experiments, it is important to evaluate the contributions of each layer. Model interpretation can help decision making process to obtain knowledge to improve the model's performance [43] and, also, improve results trustworthiness [44].

Figure 8 combine results from side-outputs inside one distinct stage while Figure 9 combine results of all previous stages, until the current one. Figure 8 helps to understand how each stage combine to the final output while 9 helps to evaluate the gain when some stages are added to the final output. Both images can contribute to the evaluation of a possible removal of stages in the network architecture.

It can be seen, in Figures 8 and 9, that the side-outputs of the first three stages are able to better identify the edges than the outputs of the final stages. The last two stages only assist the results presented by the initial layers, and the latest one, in both networks, does not seen to have any important information for the network, only random noises.

It is also important to notice in Figure 8 that most layers exhibit a large amount of light pixels (borders) compared to dark pixels (background). This behavior can be considered unexpected, as the final result has borders and backgrounds in the correct colors and tones. However, it is possible to see that the border, in most of them, is lighter than the surrounding colors, indicating an border. Also, it was observed that the latest convolution, with operation of 1×1, described in Section III-D, helps to separate these colors and tones, making the result close to the ground truth.

Figure 9 shows a different perspective of stage outputs contribution. It can be seen that the results become more accurate as more stages are added, except for the last stage, which apparently does not improve the overall performance of the model.

G. Experiment 2.3 - Removing last stage layers

After analyzing the intermediate results, as indicated in Section VI-F, a new experiment was developed with the purpose of evaluating the impact of removing layers from the last stage. As observed in Figures 8 and 9, the layers of the fifth stage apparently does not contribute effectively to the final result.

In this experiment, the same parameterization described in Section VI-E was used, except for the loss decreasing and the minimum learning rate (MinLR). The value, once fixed into 1×10^{-9}, was replaced by Equation 7, where the value depends of the original Learning Rate. This procedure reduced it to 1×10^{-10} in the first phase and 1×10^{-11} in last two phases of the protocol.

$$MinLR = LR \times 10^{-4} \tag{7}$$

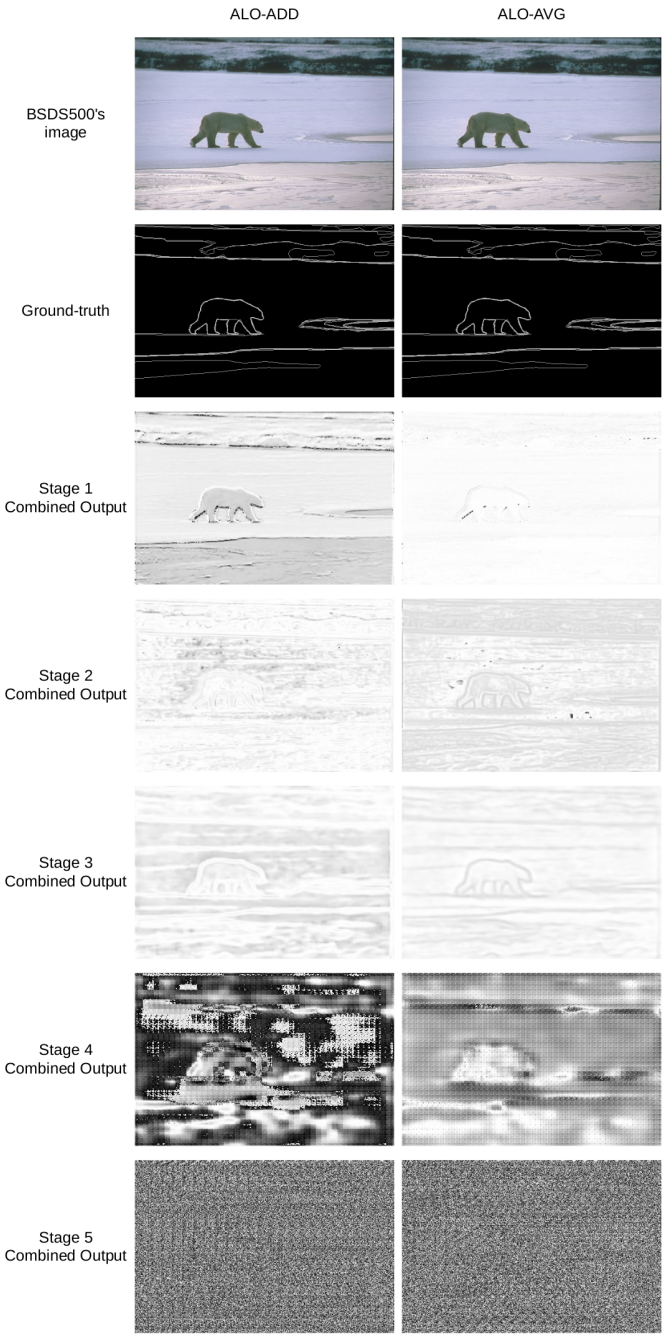


Fig. 8: Side-outputs of each stage of ALO-ADD and ALO-AVG methods, trained with BSDS 500 using MORPH/UPPER ground truth methods.

Once ALO-AVG and ALO-ADD had similar performance in the previous tests, it was decided to evaluate only one of the methods. Due to the better performance in the ODS metric, with fewer training epochs, according to Section VI-E, ALO-AVG was chosen to be used in the following experiments. Then, ALO-AVG network was completely trained using the protocol defined in Section VI-B and the parameterization recently defined.

The 4 and 5-stages ALO-AVG networks had similar values in the first phase of the protocol. In the second phase, the 4-

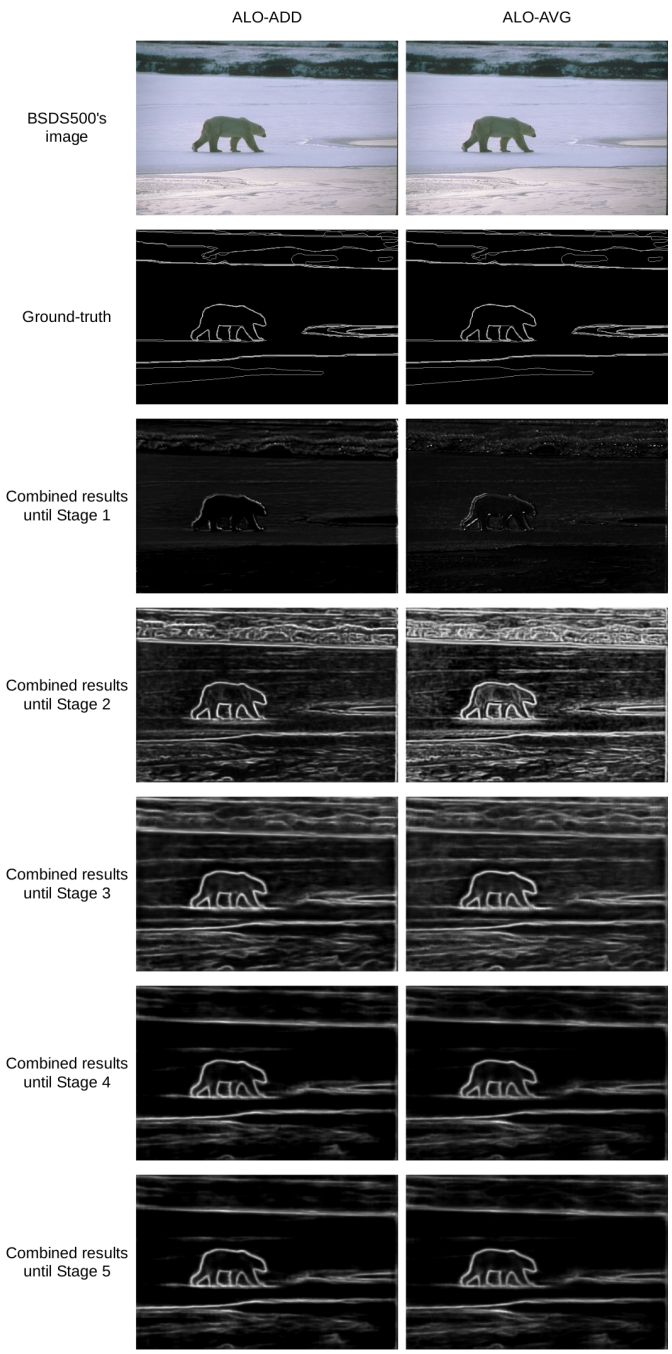


Fig. 9: Results of ALO-ADD and ALO-AVG methods with side-outputs until each stage, trained with BSDS 500 using MORPH/UPPER ground truth methods.

stage network contains considerably lower losses. This pattern is also manifest in the last phase, using ALL ground truth. As results of the lower loss, the 4-stage network produced better results, as described in Table V, despite of the bigger number of epochs to reach the best value.

The network training process of the 4-stage ALO-AVG method was carried out in 556 epochs, instead of 409 from regular 5-stages ALO-AVG. As the network with 4 layers is smaller than the traditional network, each epoch runs faster. However, even with this advantage, the total training time was about 21.80% longer than conventional 5-stage ALO-AVG.

TABLE V: Border detection performance on BSDS500 in Experiment 2.3 (using histogram equalization pre-processing).

Network	Loss	Ground truth	TH	ODS	OIS
ALO-AVG	PEFL	ALL	0.17	0.7625	0.7791

TABLE VI: Border detection performance on BSDS500 in Experiment 2.3 (using VGG16 pre-processing).

Network	Loss	Ground truth	TH	ODS	OIS
ALO-AVG	PEFL	ALL	0.18	0.7735	0.7876

1) Changing data pre processing: All previous experiments, including those with the KITTI data set, were done using histogram equalization for each image channel as a pre-processing step. The technique was chosen due to its possible adaptation to different databases. However, after all the experiments previously carried out, it was decided to evaluate the subtraction of the means of each channel, using the same parameters described in [6] paper.

Using VGGNet pre-processing weights, an increase in performance was expected, since the method would equalize the entire database evenly, unlike the previous technique, which adjusted each image separately. Such a procedure would enable faster and more effective learning.

To assess the change, a naive test was performed using the final weights from the previous experiment. The network was trained again, only in the last phase of the protocol, using images with the new pre-processing method. Surprisingly, only this step was able to raise previous results by more than 1%, to 0.7769 ODS and 0.7913 OIS values.

In order to provide a fair comparison, the network was fully trained using the same parameters described in experiment of Section VI-G. The training procedure left 490 epochs and produced the results in Table VI. The results are slightly inferior to training described in the last paragraph, but once it followed the fully protocol, it better represent the behavior of the network.

The results provided in this section indicates that the change in the pre-processing method increased the results and reduced the number of training epochs, as expected. The following experiments, in next sections, will use the same pre-processing method.

H. Experiment 2.4 - Training Side-Outputs

The current experiment aims to evaluate if making all side-outputs to produce similar predictions increase overall results. All networks used in this experiment are VGG16 versions with 4 stages and parameterization described in Section VI-G (including pre-processing of Section VI-G1).

1) Training each convolution of ALO-AVG with 4 stages: The first experiment aimed to evaluate the possible improvement in the results of the ALO-AVG network by training it with loss functions after each convolutional layer. The network was trained with 11 loss functions, one for each convolution and one for the final output, with merged output.

After training, it was observed that the network had more difficulty to learn when compared to previous architectures, resulting in a worse performance. VII. As result, the network reached an ODS value of 0.7259 and an OIS value of 0.7418 after 271 training epochs.

It was observed, in the individual evaluation of the outputs, that some intermediate results, mainly in the first stages, did not present useful information (map without any border) and that several others, specially at the end, presented many noises, generating many false positives and inaccurate edges.

2) Training each stage of SLO-AVG with 4 stages: Due to the poor results in the last experiment, it was proposed to reduce the number of side-outputs that should be trained. Following the context of this work and the previously used networks, it was decided to use the SLO-AVG net. The network was trained with 5 loss functions, one for each stage and one for the final average output.

Since early epochs, this version of the network produced smaller training and validation loss values than the previous ALO-AVG experiment. After 476 training epochs, the network produced 0.7640 ODS and 0.7795 OIS values, which is almost 6% better than the previous network. Nonetheless, these results are smaller than the results of ALO-AVG without multiple loss functions.

3) Training a fine and a coarse output: Inspired in Convolutional Oriented Boundaries network [45], this new experiment aimed to evaluate the creation of a fine and a coarse output. To produce it, a new version of ALO-AVG network was created and named ALO-PLUS.

The network contained two side-outputs, a fine one, containing the average of the initial intermediate outputs (convolutions in stages 1 to 3) and a coarse one, containing the average of the outputs of the fourth stage. The outputs were trained independently. To produce a single result, a late merging method was created, with the average of both results.

The BSDS500 evaluation of the results produced by this ALO-PLUS network resulted in 0.7573 ODS and 0.7729 OIS values (the training procedure left 534 epochs). The results can be considered intermediate when compared to values of other experiments carried out in this work.

4) Training each stage of SLO-ADD with 4 stages: Due to the good results of the SLO-AVG in the experiment described in Section VI-H2 and the similar results reached using ADD and AVG operations in experiments in Sections VI-E and VI-E, a new experiment was carried out using SLO-ADD network. The network was trained with 5 loss functions, one for each stage and one for the final output.

The training of SLO-ADD was slower to other networks, with exactly 700 epochs. Despite the longer training time, SLO-ADD produced poorly 0.7117 ODS and 0.7231 OIS values, what is considerably worse than SLO-AVG and other networks.

The results visually indicated the edges, however, presented numerous noises, which could be considered as soft edges. However, it was noticed that the hue of the soft edges was very similar to the hard edges, which probably caused the poorly results. This small difference could lead to multiple false positives, even with different thresholds.

TABLE VII: Border detection performance on BSDS500 in Experiment 2.4.

Network	Loss	Side-out.	GT	TH	ODS	OIS
ALO-AVG	PEFL	11	ALL	0.18	0.7259	0.7418
SLO-AVG	PEFL	5	ALL	0.18	0.7640	0.7795
ALO-PLUS	PEFL	2	ALL	0.18	0.7573	0.7729
SLO-ADD	PEFL	5	ALL	0.18	0.7117	0.7231

TABLE VIII: Border detection performance on BSDS500 for XLO-ADD and XLO-AVG in Experiment 2.5.

Network	Loss	Ground truth	TH	ODS	OIS	AP
XLO-ADD	PEFL	ALL	0.18	0.7760	0.7901	0.7023
XLO-ADD	PEFL	UPPER	0.30	0.7741	0.7897	0.6846
XLO-AVG	PEFL	ALL	0.19	**0.7796**	0.7937	**0.7209**
XLO-AVG	PEFL	UPPER	0.31	0.7793	**0.7961**	0.6966

5) Side-outputs training summary: The summary of the experiments performed in this section are presented in Table VII. Analyzing the results, it is possible to see that SLO-AVG network had the best ODS and OIS values. However, the results are smaller than those achieved by ALO-AVG net in Section VI-G1.

I. Experiment 2.5 - Improving ALO network

As described in Section III-B, ALO's side-output blocks are composed by an 1×1 convolution, followed by a transposed convolution. It was observed that the 1×1 convolution could be removed without great impact in the network, except for the possibility of making it a little less stable.

Due to the architectural change and in order to differentiate it from the ALO networks, the new neural network was named *eXtreme All-layers Outputs* (XLO). The XLO network was trained using the default protocol with AVG and ADD merging methods, ALL and UPPER ground truths and the following parameterization different from the default one, described in Section VI-B.

- Learning rate: 1×10^{-6} (first step) and 1×10^{-7} (others)
- Learn. decay: 1×10^{-10} (first step) and 1×10^{-11} (others)
- Loss: PEFL (γ=1.0; α=0.25)
- Pre-processing: VGG-16

After training the networks, the results were evaluated and summarized in Table VIII. The number of training epochs is available in Table IX and the some visual samples of the best results are available in Figure 10. It is possible to see in Table VIII that the results increased when compared to Section VI-G, the best results so far. Also, the number of training epochs has not increased considerably when compared to previous experiments.

TABLE IX: Number of training epochs in Experiment 2.5.

Ground truth	XLO-ADD	XLO-AVG
MORPH / ALL	527	442
MORPH / UPPER	534	596

TABLE X: FPS performance evaluation of ALO-AVG with 4 stages and XLO-AVG network.

GPU	ALO-AVG	XLO-AVG
Nvidia GTX 1080	41.2	**44.8**

The results indicating a better behavior of XLO networks when compared to all previous versions of ALO networks, although the performance is not considerably superior. However, due to the performance gain in all versions of the network, the XLO network can be considered superior to the ALO network and, consequently, to the SLO network.

To compare the speed of the best versions of XLO and ALO networks, it was created the Table X. This table contains the speed performance of both networks to predict images[11]. As can be seen in Table X, XLO network is 8,74% faster than ALO network, with better ODS and OIS values.

VII. Discussion

The experiments carried out in the KITTI data set allowed the conclusion that adding side-outputs in the network helps both the performance and the convergence during the training phase. It was also possible to directly relate the increase in network performance with the increase in the number of side outputs.

The results obtained in Section V-A indicated equivalence between side-output merging operations. However, as verified later in Section VI-D, the MAX operation performed considerably less than the ADD and AVG. The first results, which indicated similar performances, can be attributed to the relative simplicity of highway segmentation when compared to the edge detection, a much more complex and accurate task.

As verified in Section VI-F, network learning is influenced by the method used to combine the outputs. After the individual assessment of the contribution of each of the stages, in Section VI-G, it was possible to reduce the network, eliminating layers with small contributions to the result.

To train the network on the BSDS500 data set, it was necessary to change the loss function to help with the imbalance between borders and non-borders pixels. After some tests, it was observed that *Focal Loss* could learn edges but produced mostly soft borders[12]. From small experiments, it was possible to notice that the increase in the difference in tone between edge and background pixels allowed a faster network convergence. In this way, edge pixels were binary classified, regardless of the number of times they appeared in multiple ground truths.

However, multiple ground truths when placed overlapping, caused noise and thick edges. To solve this problem, a detailed study of the ground truth was carried out and a framework for experiments was created. In the framework protocol, the first phases uses large differences between the pixels, with morphological thinning operation to reduce noise, while in the following phases the differences were more subtle, which made it possible to refine the results.

Due to the smooth identification of edges, the network, despite the good results in the ODS and OIS metrics, presented low results in the AP metric, when compared to other methods in the literature. It can also be inferred from the PR curve of Figure 11 that the network does not distinguish the edges well, being necessary to consider low thresholds as edges. Low thresholds between borders and background have also been adopted in other studies, such as the DOOBNet network [33].

Based on the training protocol created, it was possible, in the last experiments, to indicate the superiority of the AVG when contrasted with ADD operation for merging the side-outputs of the network.

Regarding the training epochs on the BSDS500 data set, all networks converged with less than 700 epochs (the best ODS result converged in 442 epochs). Despite the low number of epochs, the number of images processed due to data augmentation was really high.

Due to the good results produced by the ALO and XLO networks, post-processing methods were not performed on the results of the BSDS500 data set. The usage of post-processing methods had little chance of improving the overall performance, being unjustified.

VIII. Conclusions

The present work innovatively indicates that even trivial operations such as average, sum, and maximum can be used to combine side-outputs of neural networks, increasing training speed and forecast accuracy. The paper also presents and compares characteristics of each of the trivial operations, indicating its performance both in the training and testing phases, in the region segmentation and border detection tasks.

Another contribution of this work is indicate the influence of the number of features extracted from the network: the greater the number of features, the better the performance. In addition, increasing the number of side-outputs tends to make training more stable, with fewer sudden changes between good and bad results. The network is also less susceptible to parameter initialization, requiring less care in this step.

Also, the work indicated that it is not necessary to train layers or stages individually, making side-outputs to produce similar results. Combining them before a single loss function provide similar effects to using multiple losses across the network. In this context, using different merging methods could provide different learning and, from the analysis of the intermediate results it was possible to reduce the network size, making it faster without reduce performance.

For loss functions, the work had the merit of evaluating the Focal Loss [37] function for border detection task. In addition, it was suggested simple modifications to improve convergence, creating PEFL function. The work also evaluated how to combine multiple ground truths in a single reference map, in order to obtain the faster convergence, specially in the early training stages.

The work also has as virtue its results, when compared to the literature. Despite the low number of training epochs,

[11] The images that fed the network had the size of 384×288 and were resized to the default image size of 321×481 inside the FPS evaluation.

[12] This condition was also reported in the work of [46], analyzed later.

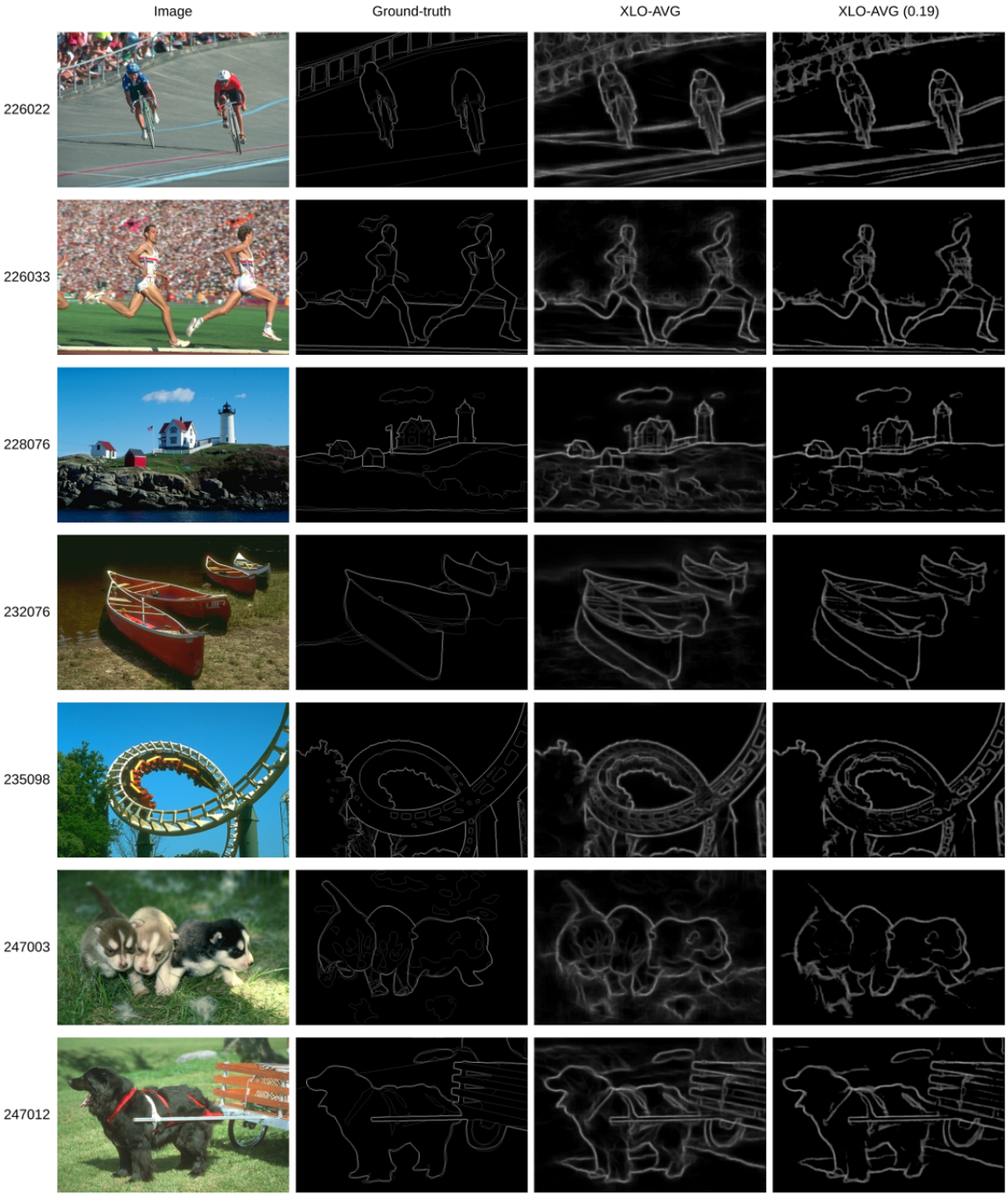


Fig. 10: Results of XLO-AVG method trained with BSDS 500 using MORPH/ALL ground truth methods.

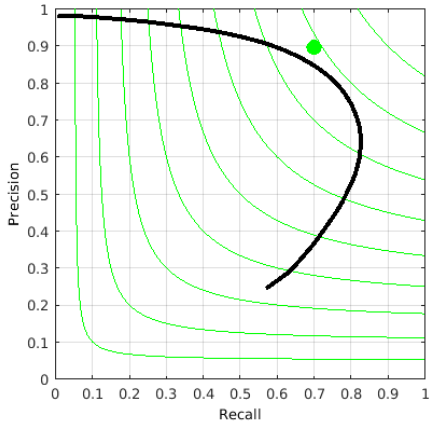


Fig. 11: Precision Recall curve for XLO-AVG network (ALL ground truth)

the method achieved performance close to the best algorithms available in the KITTI Road/Lane data set. The results of this work could be used for improvement in VB-DAS, highway monitoring and traffic management [13].

In border detection, the work presented relevant results, when compared to those existing in the literature. Despite its simple and generic proposition, the method has achieved near human threshold results in the well-known BSDS500 data set. Also, the method achieved the performance of 44.8 FPS, making it suitable for real-time applications, as microscope images information extraction and road boundary detection [47] [48] [49].

The code and the list of dependencies to reproduce the experiments (under Anaconda environment) are publicly available online in https://github.com/falreis/alo-seg-edge.

ACKNOWLEDGMENT

The authors would like to thank FAPEMIG (PPM-00006-16), CNPq, PUC Minas and CAPES for the financial support to this work.

REFERENCES

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (2nd Edition)*. Upper Saddle River, N.J: Prentice Hall, 2002.

[2] D. R. Martin, C. C. Fowlkes, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 5, pp. 530–549, May 2004.

[3] R. Jain, R. Kasturi, and B. Schunck, *Machine Vision*. McGraw-Hill Science/Engineering/Math, 1995.

[4] P. Arbelaez, "Boundary extraction in natural images using ultrametric contour maps," in *2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*, June 2006, pp. 182–182.

[5] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, Dec 2017. [Online]. Available: http://arxiv.org/abs/1511.00561

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[7] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 818–833.

[8] S. Fidler and A. Leonardis, "Towards scalable representations of object categories: Learning a hierarchy of parts," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[9] I. Hadji and R. P. Wildes, "What do we understand about convolutional networks?" *ArXiv*, vol. abs/1803.08834, 2018.

[10] S. Xie and Z. Tu, "Holistically-nested edge detection," *Int. J. Comput. Vision*, vol. 125, no. 1-3, pp. 3–18, 12 2017. [Online]. Available: https://doi.org/10.1007/s11263-017-1004-z

[11] Y. Liu, M. Cheng, X. Hu, J. Bian, L. Zhang, X. Bai, and J. Tang, "Richer convolutional features for edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 8, pp. 1939–1946, Aug 2019.

[12] J. He, S. Zhang, M. Yang, Y. Shan, and T. Huang, "Bi-directional cascade network for perceptual edge detection," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 02 2019, pp. 3823–3832.

[13] F. Reis, R. Almeida, E. Kijak, S. Malinowski, S. Guimarães, and Z. Patrocínio Jr., "Combining convolutional side-outputs for road image segmentation," in *2019 International Joint Conference on Neural Networks (IJCNN)*, July 2019, pp. 1–8.

[14] Y. Liu, M. Cheng, X. Hu, K. Wang, and X. Bai, "Richer convolutional features for edge detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 5872–5881.

[15] H. Pedrini and W. R. Schwartz, *Analise de Imagens Digitais - Princípios, Algoritmos e Aplicações*. São Paulo: Thomson Learning, 2008.

[16] J. J. Lim, C. L. Zitnick, and P. Dollr, "Sketch tokens: A learned mid-level representation for contour and object detection," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, June 2013, pp. 3158–3165.

[17] P. Dollr and C. L. Zitnick, "Fast edge detection using structured forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 8, pp. 1558–1570, Aug 2015.

[18] W. Shen, X. Wang, Y. Wang, X. Bai, and Z. Zhang, "DeepContour: A deep convolutional feature learned by positive-sharing loss for contour detection," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 3982–3991.

[19] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, May 2011.

[20] I. Kokkinos, "Pushing the boundaries of boundary detection using deep learning," vol. 4, 05 2016. [Online]. Available: https://arxiv.org/abs/1511.07386v2

[21] Y. Liu and M. S. Lew, "Learning relaxed deep supervision for better edge detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 231–240.

[22] L. Chen, J. T. Barron, G. Papandreou, K. Murphy, and A. L. Yuille, "Semantic image segmentation with task-specific edge detection using cnns and a discriminatively trained domain transform," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 4545–4554.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[24] K. Maninis, J. Pont-Tuset, P. Arbelez, and L. V. Gool, "Convolutional oriented boundaries: From image segmentation to high-level tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 819–833, April 2018.

[25] R. Wang, "Edge detection using convolutional neural network," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9719, pp. 12–20, 2016.

[26] T. W. Chua and L. Shen, "Contour detection from deep patch-level boundary prediction," in *2017 IEEE 2nd International Conference on Signal and Image Processing (ICSIP)*, Aug 2017, pp. 5–9.

[27] Y. Wang, X. Zhao, and K. Huang, "Deep crisp boundaries," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 07 2017.

[28] R. Deng, C. Shen, S. Liu, H. Wang, and X. Liu, "Learning to predict crisp boundaries," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11210 LNCS, pp. 570–586, 10 2018.

[29] D. Xu, W. Ouyang, X. Alameda-Pineda, E. Ricci, X. Wang, and N. Sebe, "Learning deep structured multi-scale features using attention-gated crfs for contour prediction," vol. 2017-December, 2017, pp. 3962–3971.

[30] S. Cai, J. Huang, J. Chen, Y. Huang, X. Ding, and D. Zeng, "Prominent edge detection with deep metric expression and multi-scale features," *Multimedia Tools and Applications*, 08 2018.

[31] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, vol. 9351. Springer, 2015, pp. 234–241, (available on arXiv:1505.04597 [cs.CV]).

[32] H. Yang, Y. Li, X. Yan, and F. Cao, "ContourGAN: Image contour detection with generative adversarial network," *Knowledge-Based Systems*, vol. 164, pp. 21 – 28, 01 2019.

[33] J. Song, Z. Zhou, L. Gao, X. Xu, and H. Shen, "Cumulative nets for edge detection," 10 2018, pp. 1847–1855.

[34] C. Wen, P. Liu, W. Ma, Z. Jian, C. Lv, J. Hong, and X. Shi, "Edge detection with feature re-extraction deep convolutional neural network," *Journal of Visual Communication and Image Representation*, vol. 57, pp. 84–90, 11 2018.

[35] X. Hu, Y. Liu, K. Wang, and B. Ren, "Learning hybrid convolutional features for edge detection," *Neurocomputing*, vol. 313, pp. 377–385, 11 2018.

[36] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[37] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. [Online]. Available: http://arxiv.org/abs/1708.02002

[38] J. Fritsch, T. Kuehnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[39] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[40] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[42] Z. Chen, J. Zhang, and D. Tao, "Progressive lidar adaptation for road detection," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 3, pp. 693–702, May 2019.

[43] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau, "Visual analytics in deep learning: An interrogative survey for the next frontiers," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 8, pp. 2674–2693, 2019.

[44] A. Chatzimparmpas, R. M. Martins, I. Jusufi, and A. Kerren, "A survey of surveys on the use of visualization for interpreting machine learning models," *Information Visualization*, vol. 19, no. 3, pp. 207–233, 2020. [Online]. Available: https://doi.org/10.1177/1473871620904671

[45] K.-K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. Van Gool, "Convolutional oriented boundaries," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 08 2016, pp. 580–596.

[46] G. Wang, X. Wang, F. W. B. Li, and X. Liang, "Doobnet: Deep object occlusion boundary detection from an image," in *Computer Vision – ACCV 2018*, C. Jawahar, H. Li, G. Mori, and K. Schindler, Eds. Cham: Springer International Publishing, 07 2018, pp. 686–702.

[47] Z. Qu, Z. Yang, and C. Ru, "Edges detection of nanowires and adaptively denoising with deep convolutional neural network from sem images," in *2020 IEEE 20th International Conference on Nanotechnology (IEEE-NANO)*, 2020, pp. 146–149.

[48] M. Li, D. Chen, S. Liu, and F. Liu, "Grain boundary detection and second phase segmentation based on multi-task learning and generative adversarial network," *Measurement*, vol. 162, p. 107857, 2020.

[49] J. W. Perng, Y. W. Hsu, Y. Z. Yang, C. Y. Chen, and T. K. Yin, "Development of an embedded road boundary detection system based on deep learning," *Image and Vision Computing*, vol. 100, p. 103935, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885620300676

Felipe A. L. Reis received B.S. degree in Computer Engineering from Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG) in 2012, a long-term graduate course in Industrial Engineering and a M.S. degree in Informatics, from PUC Minas, in 2017 and 2020, respectively. Nowadays, he is PH.d. candidate in Informatics, from PUC Minas. He worked as a software developer since 2006, in projects in accountant, educational and finance areas.

PLACE PHOTO HERE

Raquel Almeida Biography text here.

PLACE PHOTO HERE

Ewa Kijak Biography text here.

PLACE PHOTO HERE

Simon Malinowski Biography text here.

PLACE PHOTO HERE

Silvio Jamil F. Guimarães Biography text here.

Zenilton K. G. do Patrocínio Jr. Biography text here.