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

Felipe A. L. Reis
falreis@gmail.com
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Brazil

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. Some works, aiming to increase performance, decided to train individual convolutional blocks to force this behavior. This option, 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 techniques to combine features resulting from different layers of convolutional neural networks to produce boundary detection and region segmentation, using previous results from well-know object detection / classification networks. It evaluates the influence of the number of side-outputs, intermediate results 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 in a single reference map, to increase convergence. The creation of simple or even trivial methods favors the use in different scenarios, once there is no attempt to solve the uniqueness of each problem. 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, the best developed network reached 0.780 ODS on the BSDS500 dataset, at 44.8 FPS.

PUCMinas 2021, December 19, 2021, Belo Horizonte, Brazil

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

KEYWORDS

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

ACM Reference Format:
Felipe A. L. Reis. 2021. Combining Multiscale Features in Convolutional Neural Networks for Region Segmentation and Border Detection. In *Tópicos em A2DI - Visualização de Dados, December 19, 2021, Belo Horizonte, Brazil.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Segmentation subdivides an image into its constituent regions or objects, while edge detection techniques identify the boundaries between objects [18]. An edge, according to [18], is defined as a set of connected pixels that border two regions.

In addition to [18]'s definition, [37] 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" [37].

[25] 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 [25] . Contours can be combined into a UCM representation, an indexed hierarchy of regions as a soft boundaries [3].

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 [37]. 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. More recent techniques

propose supervised methods using neural networks for edge detection and region segmentation, in which image characteristics are learned by algorithms [37] [5].

In recent years, CNNs have emerged to solve increasingly complicated applications. After the rise of CNNs, some architectures have emerged to produce also region segmentation/classification and edge detection [19, Ch. 1] [5]. In edge detection, HED architecture have emerged using intermediate outputs to produce maps that are then combined, generating a final proposition [56].

Due to their architecture, a common CNN produces different information along its layers. Each layer contributes specifically to the final result, with its own characteristics, that indicates how the network adjusted its weights. This is based on a characteristic of some convolutional networks: the sequence of convolutions gradually decreases the size of images along their layers [47] [59] [16] [20].

To produce final results, some works such as HED, RCF and BDCN trained convolutional blocks individually. This procedure produces resources in macro and detailed information, with its own characteristics at different scales [56] [31] [21]. This option, however, does not take advantage of transfer learning, and increases the cost and time of training, forcing several layers to produce the same result.

To enhance training speed and accuracy, many projects developed its own loss functions and techniques to combine intermediate results, extracted from the network and called side-outputs. Also, they made some changes in well-known convolutional networks, as VGG16 [47] and ResNet [22], adding extra ones to create new features and refine intermediate results [57] [35] [6].

Training of side-outputs individually makes the networks produce results in a targeted manner, without taking advantage of information previously generated by correlated problems. Take advantage of related problems could reduce training cost and increase speed. Thus, it may be possible to simplify some methods and architectures, to produce good results with a low number of training iterations, making it suitable for proof of concepts and problems with a close deadline.

This work seeks to develop simple methods to produce results using previous information from well-know object detection networks, generic loss functions and trivial operations to combine side-outputs, such as average, maximum and sum. The creation of simple or even trivial methods favors the use in different scenarios, since they have no particularities found in each problem.

2 RELATED WORK

Edge detection has been researched over the last 50 years, with several proposed algorithms [32]. 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 [40].

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 [40]

In the current decade, works with manually developed features were also proposed, such as the Sketch Tokens method, proposed by [29]. However, new approaches have emerged using machine learning methods, as *Structured Edges*, which uses random decision forests [15].

At the same time, some studies have been carried out using deep convolutional neural networks for edge detection. These studies started with the success of neural networks for object detection [5]. *DeepContour* [46], *Holistically-nested Edge Detection (HED)* [56] and *Convolutional Oriented Boundaries (COB)* [35] networks, are some examples of the usage of this technique.

DeepContour and HED 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, using intermediary results of the well known VGG16 [47], to produce edge maps that are combined in a final layer.

Compared with VGGNet, HED removed the classification layers and added side-outputs on the last convolutional layer of each of the VGG's 5 stages. Also HED forced all side-outputs to produce predictions, using loss functions for each of them. Then, these outputs have been combined to generate a final edge map output [56].

HED's changes increased the state of art of BSDS500 data set [4] and become a success. HED proposal of side-outputs was adopted in many other projects. [27] used HED network and combine proposals in three different scales, producing a single output. The work increased the performance, but it required an architecture 3 times bigger.

Other approaches also used side-outputs, as the works of [33] and [10]. The first one captured relaxed labels from simple detectors that were merged to the ground truth, used to supervise the training and remove some hard-to-classify false positives (relaxed labels), while the last one used Semantic Segmentation and Domain Transform (DT) to improve its predictions [33] [10]. Both method use some kind of side-outputs, but the first reach better BSDS500' ODS values than HED, while the second performed worse.

The proposed architecture COB is based on ResNet [22]. It generates 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 output [34].

Contrary to the tendency of usage of side-outputs, [52] and [13] approaches didn't use them. [52] used a simple convolutional network followed by a morphological thin operation to increase accuracy. [13] 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 contours. Both results from simple convolutional architectures did not reach the results of other methods.

Following the side-output approach, it was observed by [53] 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 [53]. The border crispness was also observed by [14], that proposed refinement modules to reduce them.

Some level of refinement was used also by [57]. 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 networkd called Attention-guided Multi-scale Hierarchical deepNet (AMH-Net) for contour detection [57].

A network that proposed to create a multi-scale feature graph is the Proeminent Edge detection [6]. It uses a different approach, with architecture based on U-Net [44] and influences of ResNet, 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 every network layer [6].

An Encoder and Decoder Network was also proposed by [58]. In this approach, it was developed a Generative Adversarial Network for edge detection, called ContourGAN. 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 [58].

A more traditional work is RCF, 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 of the network, 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 [31].

Assessing the tendency in previous work to sum isolated side-outputs, with uneven qualities, [48] proposed C-Net Cumulative Network. This network 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 [48].

Similar as [48]'s work, [54] 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 to produce edge detection [54].

The Hybrid Convolutional Features network, proposed by [24], followed the usage of side pipeline to increase performance of edge detectors. It proposed fusion of multi-level information, based on feature-maps. To do it, it creates auxiliary branches, convolution, normalization, up-sampling, concatenation and a auxiliary loss function to increase the performance. Its proposed architecture is called HybridNet, and combine VGGNet and ResNet architectures [24].

Bi-Directional Cascade Network (BDCN) network, proposed by [21], 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 [21]. This work reached the state of the art in the BSDS500 data set, despite not having achieved the best performance in the NYUDv2 data set [39].

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 and NYUDv2 data sets 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 [54], 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 [56]. Then some papers started to create its own functions, as Class-balanced Cross-entropy, proposed by [56] and also used in [48] and [14] papers, that aims to balance edges and non-edges pixels.

Some works, like [27], [34], [31] and [54], adapted class-balanced cross-entropy to create their own versions, exploring parameters improvement or addind some extra features. Others, like [24] and [21], 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 [31], [58] and [21] (state-of-art) needs near 40k iterations, while others, like [6] needed 110k.

3 METHOD

The project architecture uses VGG16 neural network with multiple side-outputs, in order to obtain multi-level features. It proposes to train the network without forcing side-outputs to produce similar predictions, to reduce the number of training iterations. The architectures here are inspired by HED [56] and RCF [32] networks.

Neural Network Architecture

VGG16's architecture 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 layer with the softmax activation function, totaling 16 weight layers [47].

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. In the first proposal, side-outputs were added in the last convolution of each stage, while in the second, secondary outputs were added after each convolutional layer. Details regarding these architectures, influence on training and accuracy will be discussed in Sections 3 and 5. 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 [22], despite its higher performance.

Side-outputs

In addition to following the VGG16 architecture, the work used side-outputs, like several other existing approaches described in Section 2. 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) [56] [32].

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 resized to a single size, it is necessary that the transposed convolutions have a variable size.

Side-outputs Merging Techniques

Working with side-outputs requires that methods be used 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 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;
- *AVG*: calculates the average value of the outputs to create a proposal that represents learning at all levels;

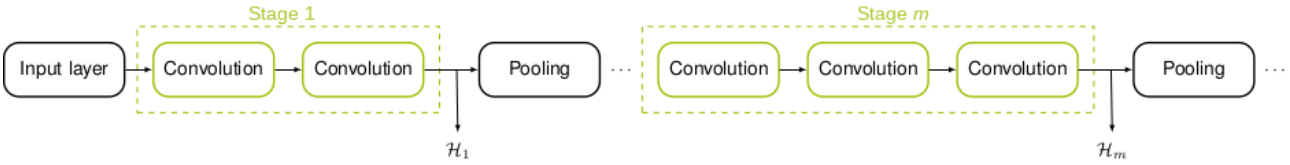


Figure 1: SLO Neural Network Architecture.

Figure 2: ALO Neural Network Architecture.

- *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:

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

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

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

Merged Output

After combining the side-outputs as detailed in Section 3, 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.

Class Balancing Methods

Edge detection seeks to identify abrupt change in an image aspect, like color or brightness [37]. 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 [30], aims helps to solve problems with high imbalance between classes. It is a change in Cross Entropy for binary classification function, that seeks to devalue pixels that are easy to identify and to value pixels that are hard to classify [30].

4 ORGANIZATION OF EXPERIMENTS

Data sets

The methods present in this work were evaluated in the following data sets, widely used and cited by different works 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 [17];
- 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. [4];

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. [17];
- BSDS500 Benchmark: BSDS500 benchmark for segmentation quality assessment. It uses precision and recall methods to classify results of segmentation and boundaries [4];

Experimental Setup

Our networks were built using using Keras [12] with Tensorflow [1]. We used a pre-trained VGG16 model to initialize the

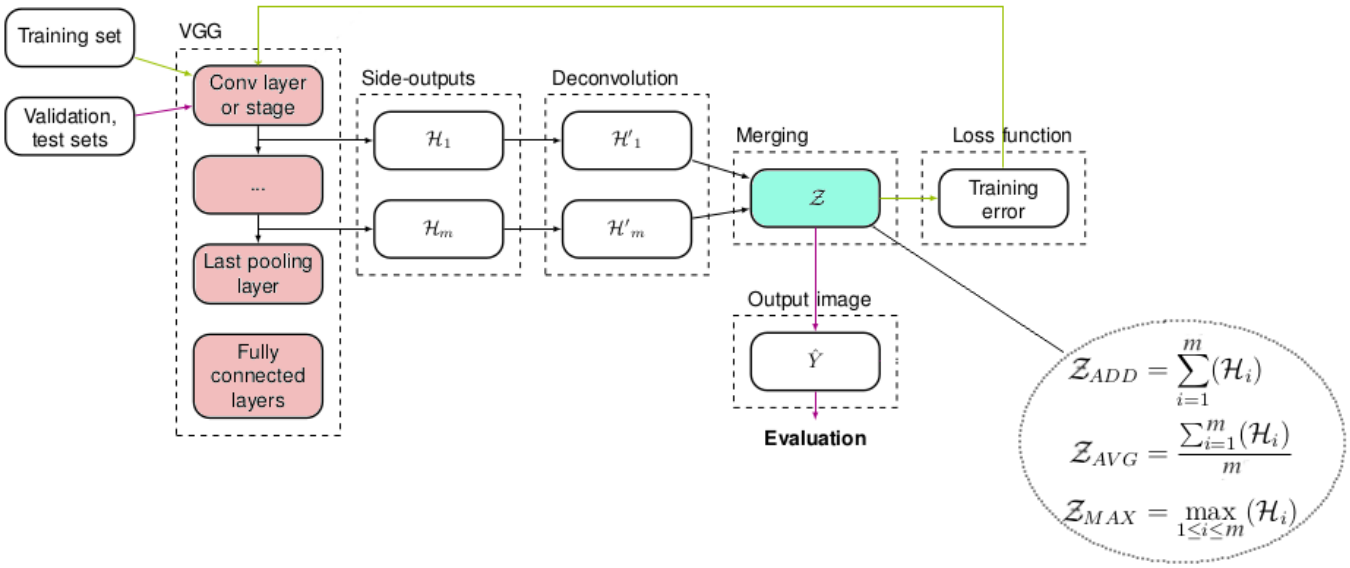


Figure 3: Overview of our method.

weights. Training experiments were performed in GeForce GTX 1080 8GB GPU[1].

Dataset Visualization

The understanding of information can be facilitated with the aid of visual representation and data presentation. Both can help in the perception, interpretation and understanding of information. It also can be used to summarize information and give diferent perspective of information than can't be seen without these techniques [26].

Data visualization can also be used as an initial step during a machine learning process. In 1973, [2] showed that basic statistics such as mean, standard deviation and correlation, may not provide enough information about the data. Then, visualize datasets can provide insights, help to understanding the problem and assist in selecting important features for learning [36].

Despite importance of visualization, as indicated previously, translate raw data into useful images to provide information can be challenging. Identify relationships among features, detect patterns and relevant profiles to improve decision-making, and also present the best representation for the information could be not be a easy task [49]. Then, some works like [55], [7] and [38], can provide some inspiration to represent some data, like graphs and trees.

In addition to representing unstructured data, visualization have been used to help understand black-box methods. Such methods, such as neural networks, do not provide information about learning and despite the good results, they are often questioned in some areas (eg. medicine). Visual representation of how networks learn can provide greater confidence, making it possible to expand the use of machine learning techniques [8].

Some recent works try to explain and show the internal structures of deep neural networks, helping to understand the behavior of algorithms. Such works help in decision making for the construction of new solutions with greater reliability. [8], [23] and [50] indicate the main contributions related to these activities in recent years.

Finally, data visualization can represent the entire knowledge process obtained throughout the construction of the work. Some information such as network training, architecture details, among others, can be represented through images, summarizing the process of obtaining the results obtained by the author.

5 REGION SEGMENTATION EXPERIMENTS

The first set of experiments aimed to analyze the feasibility of the method described in Section 3. For an initial analysis, the region segmentation task was considered more appropriate, as it requires a lower level of precision than edge detection problems. Also, it was evaluated the influence of the amount of side-outputs and the performance of the methods to combine them.

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

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

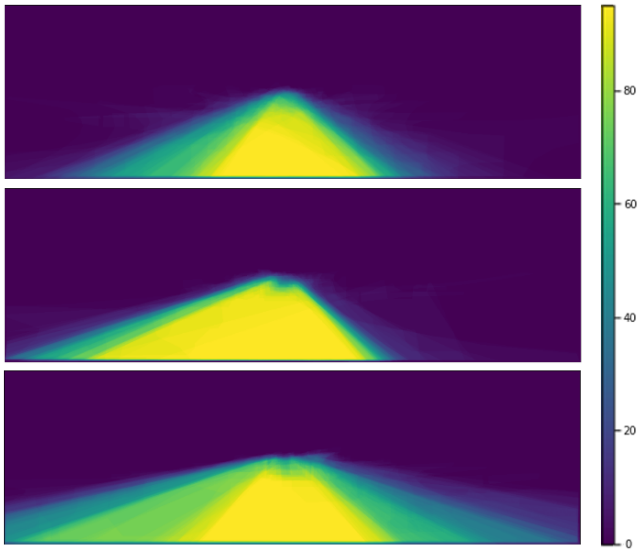


Figure 4: KITTI *uu_road*, *um_road* and *umm_road* combined ground truth colormap.

the area of the road that makes up all lanes; and (*ii*) **lane** is the ego-lane where the vehicle is currently in [17].

KITTI data set contains only ground truth for training data. The test set is evaluated using the KITTI Server Evaluation. In the work developed, 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**), (*iii*) urban multiple marked lanes (**umm_road**) [17].

To visualize the data set and understand better the problem, Figure 4 was developed. In this image, ground truths were combined into images of each category. Categories *uu_road*, *um_road* and *umm_road* contains, respectively, 13.68%, 16.22% and 23.41% of road pixels, respectively. Although the highway and background classes are unbalanced, the pixel position favors the learning of the neural network, and it is not necessary to use pixel balancing methods.

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.

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 3. 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 5 shows the loss curves during the training phase for the validation set. Once the number of trained networks is big, the curves were separated according to the architecture, in a small multiple format. This type of graph facilitates the understanding of the information and make analysis more clear.

It is observed in the first part of the experiment 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*. Figure **??** was developed to show the accuracy of the network in this experiment.

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.

[2]VGG16 model was trained using ImageNet dataset [45].

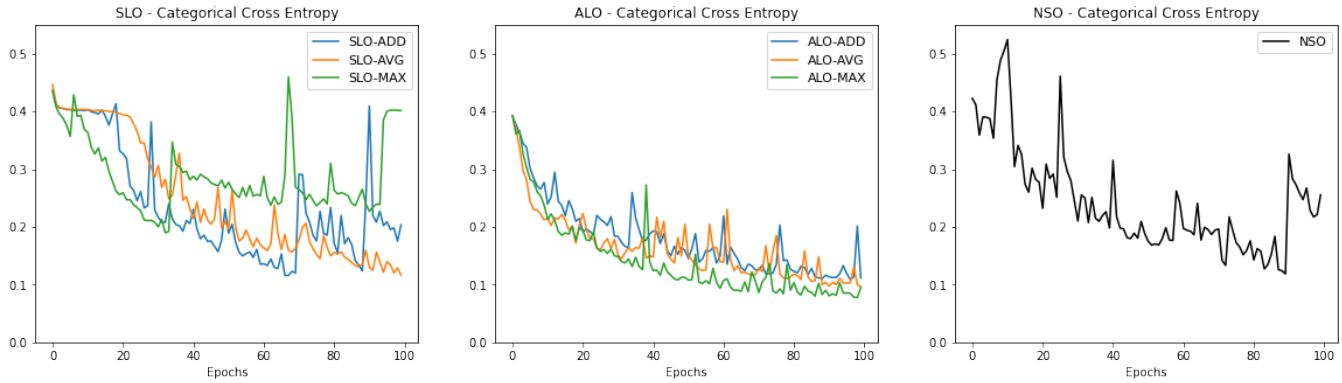


Figure 5: Validation loss using *Categorical Cross Entropy* metric in first experiment.

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

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 1: Segmentation performance on KITTI Road Dataset.

Side-Outputs Contribution

The layers in each merging strategy learn uniquely. The contributions to the final prediction differ according to the merging method adopted, as shown in Figure 6. To make the analysis of the image clearer, only last side-output maps of each stage were plotted. In addition, the images were transformed into monochromatic (binary), with white pixels representing the road and black pixels representing the background.

It can be seen from Figure 6 that the first two stages ($\mathcal{H}_2$ and $\mathcal{H}_4$) side-outputs does not produce meaningful information. The images have mostly white pixels, indicating classification of all pixels as road. In the third stage (output $\mathcal{H}_7$), the merging methods ALO-AVG and ALO-ADD, contains a obvious separation between road pixels and the background. The output map $\mathcal{H}_7$, using ALO-MAX method, however, does not clearly separate road from non-road pixels.

Figure 6 further shows that the best side-output maps for all networks ALO are generated in the fourth stage $\mathcal{H}_{10}$. Road markings are visible even if there are noises, especially in the ALO-MAX method. The final side-output, $\mathcal{H}_{13}$, contains high noise level. This feature indicates that the layer was not able to learn correctly.

The merged output combines all side-outputs, including other intermediate results not available in Figure 6, for prediction. Despite the poor results in some layers, the learning process adjusts itself so that even bad results can be used by the model.

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.

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 to the KITTI Evaluation Server, with name of **ALO-AVG-MM**[3]. The server returned the following metrics: MaxF, AP, PRE, REC, FPR and FNR, whose results for each road scene category[4], as described in Section 5, are available in Table 1.

Compared to the best result on the KITTI platform named PLARD [11][5], our method had an overall performance 5.0%

[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, [43], *PLARD* had not yet been published in the literature, being an anonymous submission on KITTI platform.

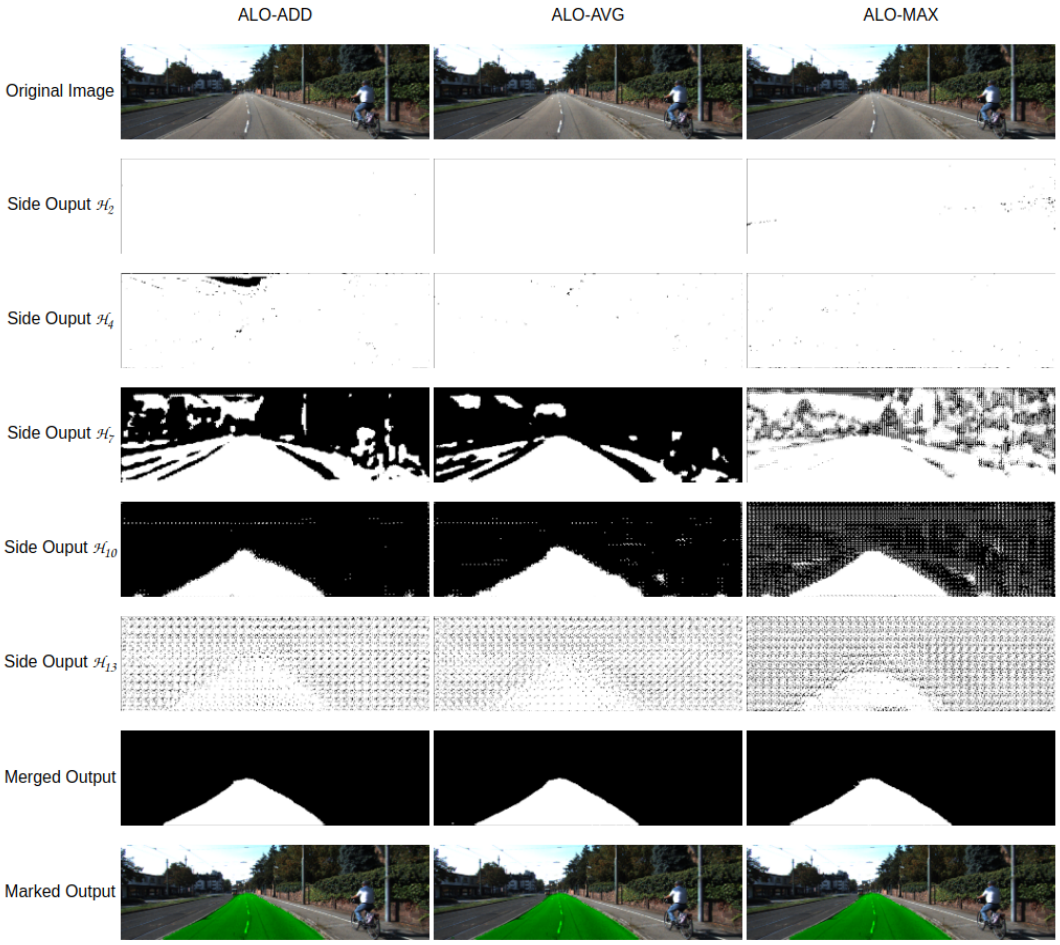


Figure 6: Side-output maps for ALO network merging strategies in KITTI Road/Lane data set.

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 7 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.

6 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 [4].

To visualize the data set and understand better the problem, Figure 8 was developed. Unlike the KITTI database, the BSDS database does not contain pixels of interest distributed in a specific position of the image, making training difficult. Thus, it becomes necessary to use methods that can work with unbalanced classes, as discussed in the Section 3.

BSDS500's test set is evaluated using BSDS500 Benchmark, provided by the authors [4]. In our experiments, training and validation sets were merged into one group, in order

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

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

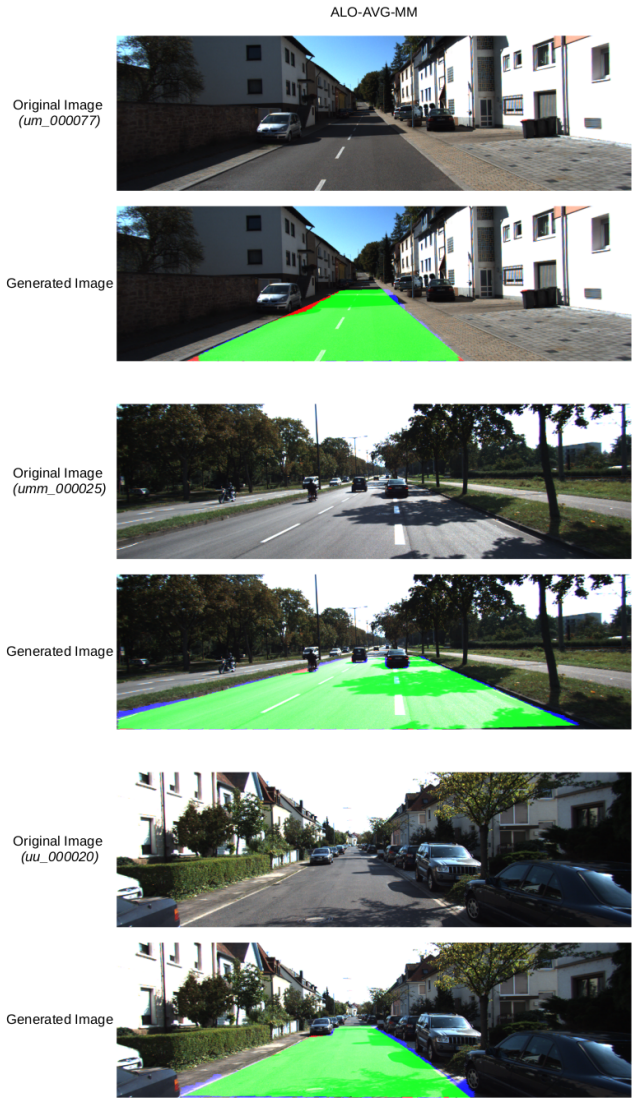


Figure 7: Visual representation of ALO-AVG-MM predictions

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, 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,

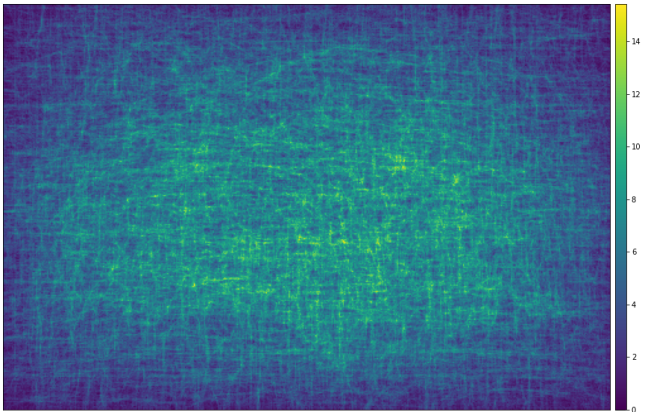


Figure 8: BSDS500 combined ground truth colormap.

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 6.

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 7.

Ground truth analysis

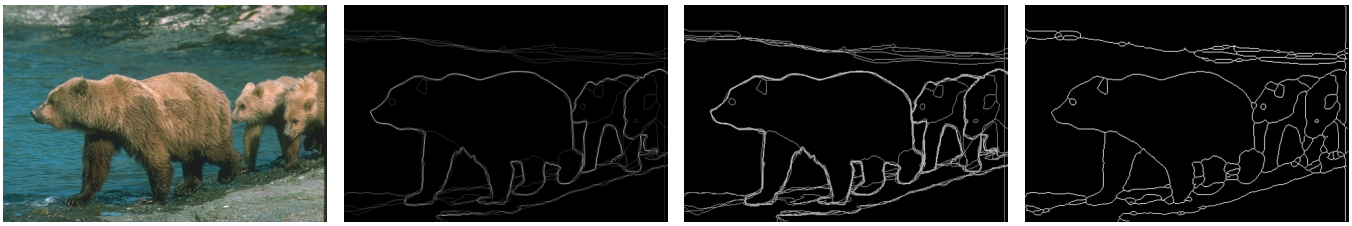
As briefly reported previously, BSDS500's ground truths are manually annotated by five different subjects on average [4]. Due multiples ground truths, some experiments were conducted to define how to combine them, in order to help the network to learn properly. To do that, we developed 3 different methods 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 9b.

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



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

Figure 9: (a) BSDS500 image and its ground truth, using (b) ALL, (c) UPPER and (d) MORPH representations.

ground truth representation method is available in Figure 9c.

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 9d.

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.

Protocol for experiments

As described in Section 6, the use 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;

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 step) and 1×10^{-7} (others)
- Learn. decay: 1×10^{-10} (first step) 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.

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.

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 6.

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 [30] paper.

ALO-ADD. The first network trained in the protocol was the ALO-ADD, which proved to be more stable in different preliminary tests. 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 6). Close to the 500th epoch, the network showed again poor training loss decrease and it was finished in the 567th epoch.

ALO-AVG. The second network trained in the protocol was the ALO-AVG, which had similar performance to ALO-ADD in preliminary tests. ALO-AVG promptly converged, with a behavior similar to ALO-ADD. After around 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}).

ALO-MAX. ALO-MAX was the last network trained in this experiment. 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).

It is also possible to notice that 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.

The second phase indicates that the network was almost training and did not improved its results. It is possible to conclude that only the training of the deconvolution layers was enough for the network to reach the best possible value.

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 10. To evaluate the gain of UPPER and ALL ground truths usage, it was decided to evaluate MORPH performance also. As explained in Section 6, the network was trained until it does not improve for 50 epochs, making MORPH almost unable to perform better.

Figure 10 was developed using small multiples technique to show the difference between methods. ALO-ADD and ALO-AVG were plotted in different images once the results were similar and one data was above other. This results contrasts with the poor performance of ALO-MAX network.

Table 2: Number of training epochs using Focal Loss.

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

It is possible to notice, in the analysis of Figure 10, 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 seens to influence the evaluation, once the benchmark compare borders with each human annotation in each ground truth.

The overall results of ALO-ADD and ALO-AVG were really similar, but ALO-ADD had better performance using both ODS and OIS metrics. The number of training epochs to reach the best results were also similar, as described in Table 2, but with less epochs for ALO-AVG. These results indicates that both methods are similar in performance and number of training epochs. Table 2 also shows that 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.

Due the weak convergence of ALO-MAX in all previous tests and preliminary experiments, it was decided that the ALO-MAX method will no longer be tested, due to its low performance. The following sections will provide and discuss experiments using only the ALO-ADD and ALO-AVG methods.

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 converge more quickly. It is important to clarify

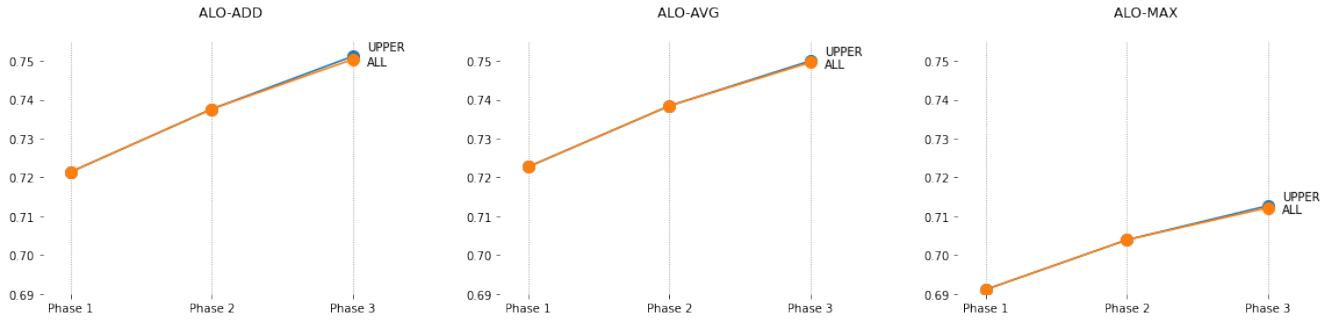


Figure 10: Border detection ODS performance on BSDS500 for ALO-AVG, ALO-ADD and ALO-MAX.

Table 3: Border detection performance on BSDS500 for ALO-ADD and ALO-AVG.

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

that this change will mainly affect the ground truths MORPH and UPPER, that contains edge values far from background values. This new metric, named PEFL, can be simple defined as Equation 6:

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

In conjunction with the new loss function, some hyper-parameter tuning were made[9]. 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 6.

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 6. 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 6, using the default parameterization, as can be seen in Table 3.

Table 4 shows that the number of training epochs has decreased compared to the results of the previous training, available in Table 2. Comparing the number of epochs

[9] The experiment described in Section 6 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.

Table 4: Number of training epochs of Pixel Error Focal Loss.

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

of both experiments, it is possible to perceive that ALO-ADD reduced 16.97% in MORPH/ALL method and 11.05% in 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.

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 [23] and, also, improve results trustworthiness [8].

Contrary to what was done in Section 5, where only the last layer of each stage was plotted, Figure 11 combine results from side-outputs inside one distinct stage while Figure 12 combine results of all previous stages, until the current one. Figure 11 helps to understand how each stage combine to the final output while 12 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 11 and 12, 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 11 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 3, helps to separate these colors and tones, making the result close to the ground truth.

Figure 12 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.

Experiment 2.3 - Removing last stage layers

After analyzing the intermediate results, as indicated in Section 6, a new experiment was developed with the purpose of evaluating the impact of removing layers from the last stage. As observed in Figures 11 and 12, the layers of the fifth stage apparently does not contribute effectively to the final result.

In this experiment, the same parameterization described in Section 6 was used, except for the loss decreasing and the minimum learning rate (MinLR). The value, once fixed into $1{\times}10^{-9}$, 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.

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

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 6, ALO-AVG was chosen to be used in the following experiments. Then, ALO-AVG network was completely trained using the protocol defined in Section 6 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-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 5, 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.

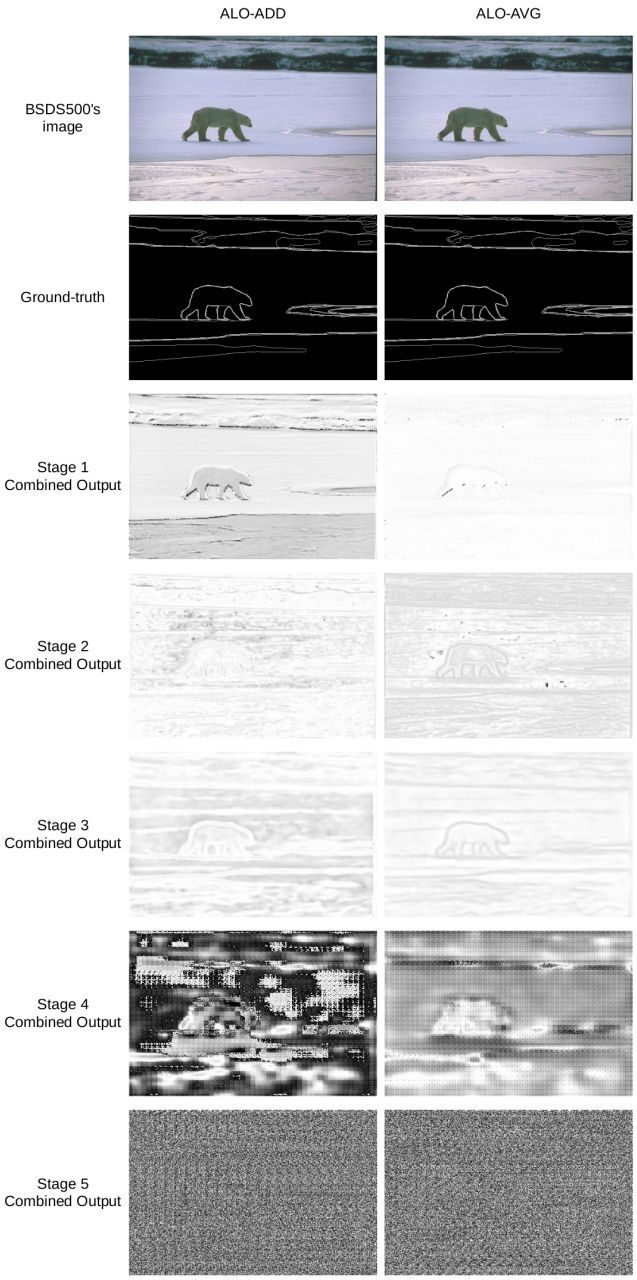


Figure 11: Side-outputs of each stage of ALO-ADD and ALO-AVG methods, trained with BSDS 500 using MORPH/UPPER ground truth methods.

Table 5: Border detection performance on BSDS500 for ALO-AVG with 4 stages.

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

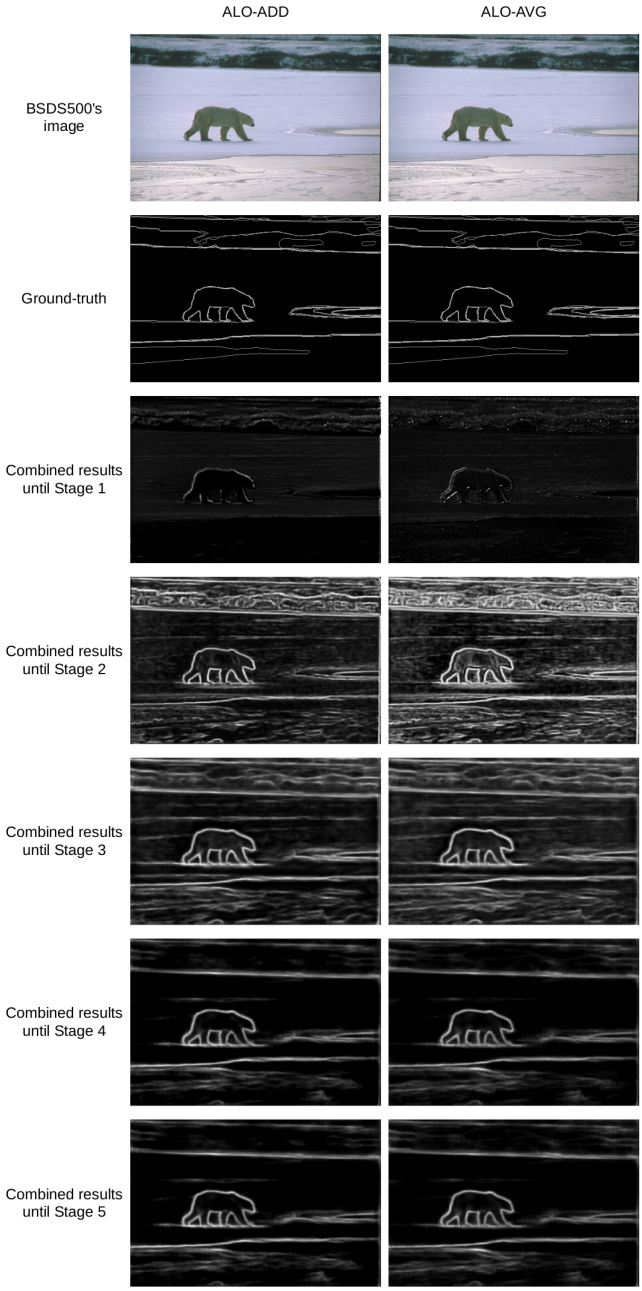


Figure 12: Results of ALO-ADD and ALO-AVG methods with side-outputs until each stage, trained with BSDS 500 using MORPH/UPPER ground truth methods.

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 [47] paper.

Table 6: Border detection performance on BSDS500 for ALO-AVG with 4 stages, using VGG16 pre-processing.

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

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 6. The training procedure left 490 epochs and produced the results in Table 6. 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.

Experiment 2.5 - Improving ALO network

As described in Section 3, 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 made and in order to differentiate it from the ALO network, 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 6.

- 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 7. The number of training epochs

Table 7: Border detection performance on BSDS500 for XLO-ADD and XLO-AVG.

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

Table 8: Number of training epochs of Pixel Error Focal Loss.

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

Table 9: FPS evaluation of ALO-AVG with 4 stages and XLO-AVG network

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

are available in Table 8 and the some visual samples of the best results are available in Figure 13. It is possible to see in Table 7 that the results increased when compared to Section 6, the best results so far. Also, the number of training epochs has not increased considerably when compared to previous experiments.

The results indicating a better behavior of XLO networks when compared to all previous versions of ALO network, although the performance is not considerably superior. However, due to the performance gain in all versions of the network, combining side-outputs with ADD or AVG functions, 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 9. This table contains the speed performance of both networks to predict images[10]. As can be seen in Table 9, XLO network is 8,74% faster than ALO network, with better ODS and OIS values.

Experiment Summary

The evolution of results due to experiments using BSDS500 are available in Figure 14. The figure summarizes the experiments in the BSDS500 dataset, indicating the main modifications at each step. The figure was created to facilitate the understanding of each of the experiments, indicating the main modifications carried out.

[10] 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.

7 DISCUSSION

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

The results obtained in Section 5 indicated equivalence between side-output merging operations. However, as verified later in Section 6, 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 5 and later in Section 6, 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 6, it was possible to reduce the network, eliminating layers with small contributions to the result.

To train the network on the BSDS500 dataset, 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[11]. 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 15 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 [48].

Based on the training protocol created, it was possible, in the last experiments, to indicate the superiority of the

[11] This condition was also reported in the work of [51], analyzed later.

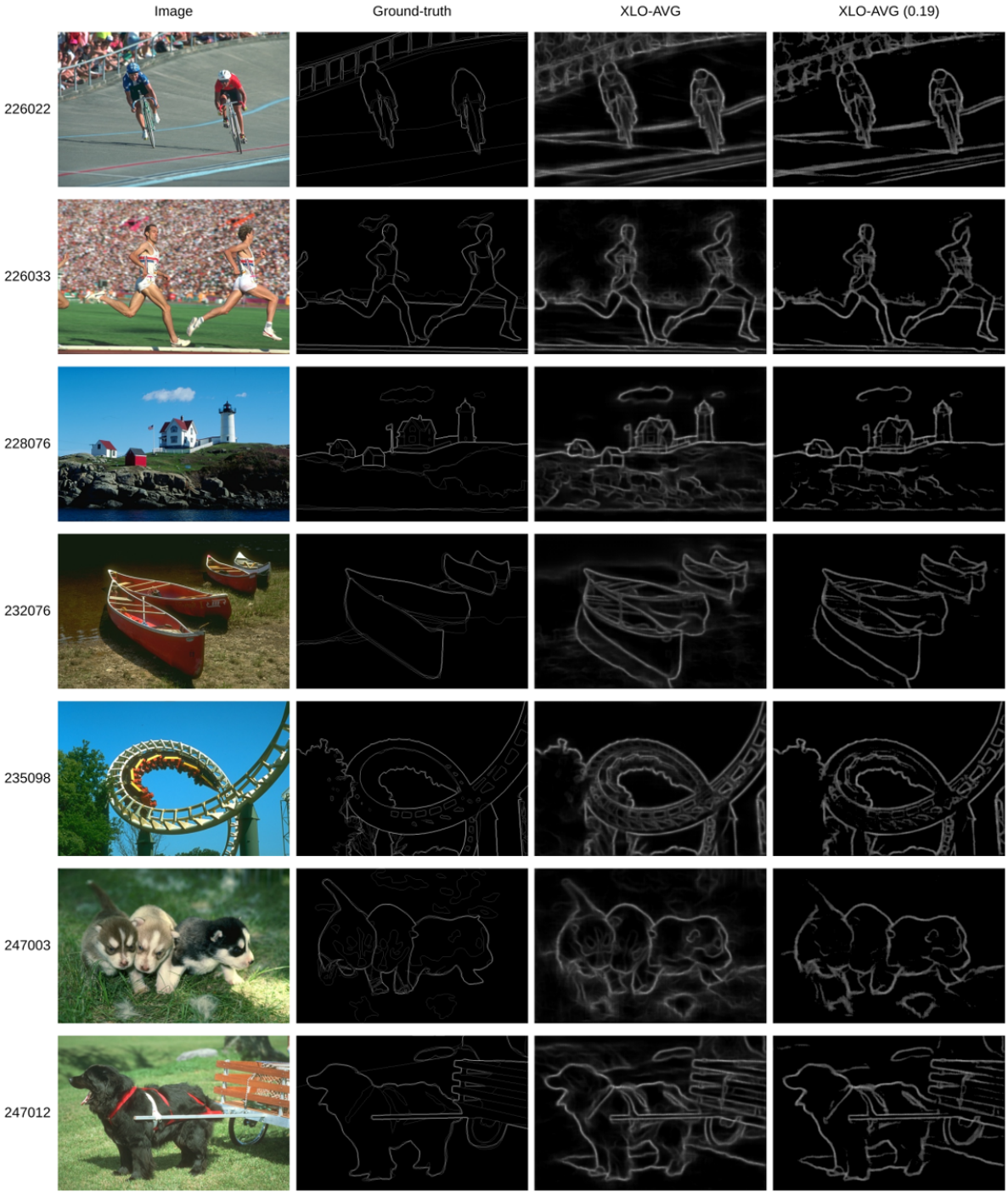


Figure 13: Results of XLO-AVG method trained with BSDS 500 using MORPH/ALL ground truth methods.

AVG when contrasted with ADD operation for merging the side-outputs of the network.

Regarding the training epochs on the BSDS500 dataset, 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.

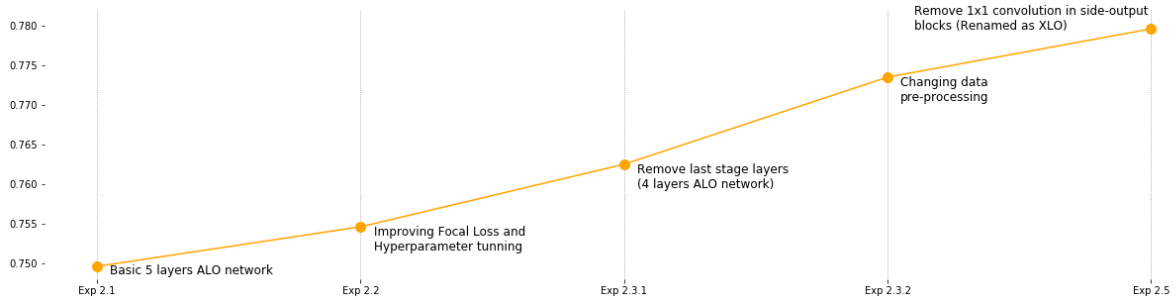


Figure 14: Results of ALO-AVG due experiments.

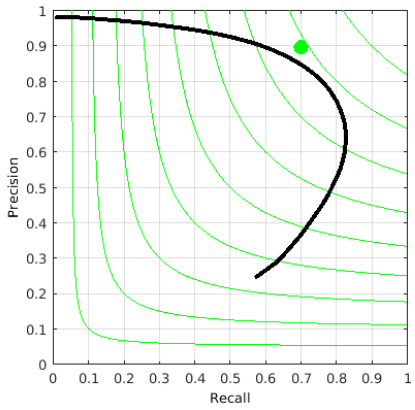


Figure 15: Precision Recall curve for XLO-AVG network (ALL ground truth)

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.

8 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 the work is to 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 [30] 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, 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 [43].

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 [42] [28] [41].

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*.

9 AUTHORS AND AFFILIATIONS

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu

Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] F. J. Anscombe. 1973. Graphs in Statistical Analysis. *The American Statistician* 27, 1 (1973), 17–21. https://doi.org/10.1080/00031305.1973.10478966

[3] P. Arbelaez. 2006. Boundary Extraction in Natural Images Using Ultrametric Contour Maps. In *2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*. 182–182. https://doi.org/10.1109/CVPRW.2006.48

[4] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. 2011. Contour Detection and Hierarchical Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 5 (May 2011), 898–916. https://doi.org/10.1109/TPAMI.2010.161

[5] V. Badrinarayanan, A. Kendall, and R. Cipolla. 2017. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 12 (Dec 2017), 2481–2495. https://doi.org/10.1109/TPAMI.2016.2644615 arXiv:1511.00561

[6] S. Cai, J. Huang, J. Chen, Y. Huang, X. Ding, and D. Zeng. 2018. Prominent edge detection with deep metric expression and multi-scale features. *Multimedia Tools and Applications* (08 2018). https://doi.org/10.1007/s11042-018-6581-5

[7] Yuan-chin Ivan Chang, Yuh-Jye Lee, Hsing-Kuo Pao, Mei-Hsien Lee, and Su-Yun Huang. 2016. Data Visualization via Kernel Machines. See [9], 539–559.

[8] Angelos Chatzimparmpas, Rafael M. Martins, Ilir Jusufi, and Andreas Kerren. 2020. A survey of surveys on the use of visualization for interpreting machine learning models. *Information Visualization* 19, 3 (2020), 207–233. https://doi.org/10.1177/1473871620904671 arXiv:https://doi.org/10.1177/1473871620904671

[9] Chun-Houh Chen, Wolfgang Hädrdle, and Antony Unwin. 2016. *Handbook of Data Visualization*. Springer Publishing Company, Incorporated.

[10] L. Chen, J. T. Barron, G. Papandreou, K. Murphy, and A. L. Yuille. 2016. 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)*. 4545–4554. https://doi.org/10.1109/CVPR.2016.492

[11] Z. Chen, J. Zhang, and D. Tao. 2019. Progressive LiDAR adaptation for road detection. *IEEE/CAA Journal of Automatica Sinica* 6, 3 (May 2019), 693–702. https://doi.org/10.1109/JAS.2019.1911459

[12] François Chollet et al. 2015. Keras. https://keras.io.

[13] T. W. Chua and L. Shen. 2017. Contour detection from deep patch-level boundary prediction. In *2017 IEEE 2nd International Conference on Signal and Image Processing (ICSIP)*. 5–9. https://doi.org/10.1109/SIPROCESS.2017.8124495

[14] R. Deng, C. Shen, S. Liu, H. Wang, and X. Liu. 2018. Learning to predict crisp boundaries. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11210 LNCS (10 2018), 570–586. https://doi.org/10.1007/978-3-030-01231-1_35

[15] P. DollÃąr and C. L. Zitnick. 2015. Fast Edge Detection Using Structured Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, 8 (Aug 2015), 1558–1570. https://doi.org/10.1109/TPAMI.2014.2377715

[16] S. Fidler and A. Leonardis. 2007. Towards Scalable Representations of Object Categories: Learning a Hierarchy of Parts. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 1–8.

[17] Jannik Fritsch, Tobias Kuehnl, and Andreas Geiger. 2013. A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms. In *International Conference on Intelligent Transportation Systems (ITSC)*.

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

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[20] Isma Hadji and Richard P. Wildes. 2018. What Do We Understand About Convolutional Networks? *ArXiv* abs/1803.08834 (2018).

[21] J. He, S. Zhang, M. Yang, Y. Shan, and T. Huang. 2019. Bi-Directional Cascade Network for Perceptual Edge Detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 3823–3832.

[22] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[23] Fred Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau. 2019. Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers. *IEEE Transactions on Visualization and Computer Graphics* 25, 8 (2019), 2674–2693. https://doi.org/10.1109/TVCG.2018.2843369

[24] X. Hu, Y. Liu, K. Wang, and B. Ren. 2018. Learning hybrid convolutional features for edge detection. *Neurocomputing* 313 (11 2018), 377–385. https://doi.org/10.1016/j.neucom.2018.05.088

[25] Ramesh Jain, Rangacher Kasturi, and Brian Schunck. 1995. *Machine Vision*. McGraw-Hill Science/Engineering/Math.

[26] Andy Kirk. 2016. *Data Visualisation: A Handbook for Data Driven Design* (2 ed.). Sage Publications Ltd. 368 pages.

[27] Iasonas Kokkinos. 2016. Pushing the Boundaries of Boundary Detection using Deep Learning. *4th International Conference on Learning Representations (ICLR 2016)* 4. https://arxiv.org/abs/1511.07386v2

[28] Mingchun Li, Dali Chen, Shixin Liu, and Fang Liu. 2020. Grain boundary detection and second phase segmentation based on multi-task learning and generative adversarial network. *Measurement* 162 (2020), 107857. https://doi.org/10.1016/j.measurement.2020.107857

[29] J. J. Lim, C. L. Zitnick, and P. DollÃąr. 2013. Sketch Tokens: A Learned Mid-level Representation for Contour and Object Detection. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 3158–3165. https://doi.org/10.1109/CVPR.2013.406

[30] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss for Dense Object Detection. *CoRR* abs/1708.02002 (2017). arXiv:1708.02002 http://arxiv.org/abs/1708.02002

[31] Y. Liu, M. Cheng, X. Hu, J. Bian, L. Zhang, X. Bai, and J. Tang. 2019. Richer Convolutional Features for Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 8 (Aug 2019), 1939–1946. https://doi.org/10.1109/TPAMI.2018.2878849

[32] Y. Liu, M. Cheng, X. Hu, K. Wang, and X. Bai. 2017. Richer Convolutional Features for Edge Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5872–5881. https://doi.org/10.1109/CVPR.2017.622

[33] Y. Liu and M. S. Lew. 2016. Learning Relaxed Deep Supervision for Better Edge Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 231–240. https://doi.org/10.1109/CVPR.2016.32

[34] K. Maninis, J. Pont-Tuset, P. ArbelÃąez, and L. Van Gool. 2018. Convolutional Oriented Boundaries: From Image Segmentation to High-Level

Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (April 2018), 819–833.

[35] Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Pablo Arbeláez, and Luc Van Gool. 2016. Convolutional Oriented Boundaries. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 580–596.

[36] Stephen Marsland. 2014. *Machine Learning: An Algorithm Perspective* (2 ed.). CRC Press. Disponível em: https://homepages.ecs.vuw.ac.nz/ marslast/MLbook.html.

[37] D. R. Martin, C. C. Fowlkes, and J. Malik. 2004. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26, 5 (May 2004), 530–549. https://doi.org/10.1109/TPAMI.2004.1273918

[38] George Michailidis. 2016. Data Visualization Through Their Graph Representations. See [9], 103–120.

[39] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. 2012. Indoor Segmentation and Support Inference from RGBD Images. In *ECCV*.

[40] Hélio Pedrini and William Robson Schwartz. 2008. *AnaÌĄlise de Imagens Digitais - Princípios, Algoritmos e Aplicações*. Thomson Learning, São Paulo.

[41] Jau Woei Perng, Ya Wen Hsu, Ya Zhu Yang, Chia Yen Chen, and Tang Kai Yin. 2020. Development of an embedded road boundary detection system based on deep learning. *Image and Vision Computing* 100 (2020), 103935. https://doi.org/10.1016/j.imavis.2020.103935

[42] Z. Qu, Z. Yang, and C. Ru. 2020. 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)*. 146–149. https://doi.org/10.1109/NANO47656.2020.9183671

[43] Felipe Reis, Raquel Almeida, Ewa Kijak, Simon Malinowski, Silvio Guimarães, and Zenilton Patrocínio Jr. 2019. Combining convolutional side-outputs for road image segmentation. In *2019 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN.2019.8851843

[44] O. Ronneberger, P.Fischer, and T. Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI) (LNCS)*, Vol. 9351. Springer, 234–241. (available on arXiv:1505.04597 [cs.CV]).

[45] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[46] Wei Shen, Xinggang Wang, Yan Wang, Xiang Bai, and Z. Zhang. 2015. DeepContour: A deep convolutional feature learned by positive-sharing loss for contour detection. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3982–3991. https://doi.org/10.1109/CVPR.2015.7299024

[47] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556 http://arxiv.org/abs/1409.1556

[48] J. Song, Z. Zhou, L. Gao, X. Xu, and H.T. Shen. 2018. Cumulative nets for edge detection. *MM 2018 - Proceedings of the 2018 ACM Multimedia Conference* (10 2018), 1847–1855. https://doi.org/10.1145/3240508.3240688

[49] Alfredo Vellido, José David Martín-Guerrero, Fabrice Rossi, and Paulo J. G. Lisboa. 2011. Seeing is believing: The importance of visualization in real-world machine learning applications. In *ESANN 2011, 19th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 27-29, 2011, Proceedings*. https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2011-3.pdf

[50] Jagoda Walny, Christian Frisson, Mieka West, Doris Kosminsky, Søren Knudsen, Sheelagh Carpendale, and Wesley Willett. 2020. Data Changes Everything: Challenges and Opportunities in Data Visualization Design Handoff. *IEEE Trans. Vis. Comput. Graph.* 26, 1 (2020), 12–22. https://doi.org/10.1109/TVCG.2019.2934538

[51] Guoxia Wang, Xiaochuan Wang, Frederick W. B. Li, and Xiaohui Liang. 2018. DOOBNet: Deep Object Occlusion Boundary Detection from an Image. In *Computer Vision – ACCV 2018*, C.V. Jawahar, Hongdong Li, Greg Mori, and Konrad Schindler (Eds.). Springer International Publishing, Cham, 686–702.

[52] R. Wang. 2016. Edge detection using convolutional neural network. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9719 (2016), 12–20. https://doi.org/10.1007/978-3-319-40663-3_2

[53] Yupei Wang, Xin Zhao, and Kaiqi Huang. 2017. Deep Crisp Boundaries. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[54] C. Wen, P. Liu, W. Ma, Z. Jian, C. Lv, J. Hong, and X. Shi. 2018. Edge detection with feature re-extraction deep convolutional neural network. *Journal of Visual Communication and Image Representation* 57 (11 2018), 84–90. https://doi.org/10.1016/j.jvcir.2018.10.017

[55] Leland Wilkinson. 2016. Graph-theoretic Graphics. See [9], 122–150.

[56] Saining Xie and Zhuowen Tu. 2015. Holistically-Nested Edge Detection. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 1395–1403. https://doi.org/10.1109/ICCV.2015.164

[57] D. Xu, W. Ouyang, X. Alameda-Pineda, E. Ricci, X. Wang, and N. Sebe. 2017. Learning deep structured multi-scale features using attention-gated CRFs for contour prediction. *Advances in Neural Information Processing Systems* 2017-December (2017), 3962–3971.

[58] Hongju Yang, Yao Li, Xuefeng Yan, and Fuyuan Cao. 2019. ContourGAN: Image contour detection with generative adversarial network. *Knowledge-Based Systems* 164 (01 2019), 21 – 28. https://doi.org/10.1016/j.knosys.2018.09.033

[59] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 818–833.