Incremental Learning in Semantic Segmentation

Bosio Riccardo
Data Science and Engineering
Politecnico di Torino
s291299@studenti.polito.it

Fantolino Edoardo
Data Science and Engineering
Politecnico di Torino
s286008@studenti.polito.it

Macchia Beatrice
Data Science and Engineering
Politecnico di Torino
s292178@studenti.polito.it

Abstract

This document contains our ideas about the project assigned by Barbara Caputo and Fabio Cermelli concerning the state of the art of Semantic Segmentation (SS) techniques applied in an Incremental Learning (IL) setting.

Keywords: *Incremental Learning, Semantic Segmentation, Deep Convolutional Neural Networks.*
Code at: https://github.com/edoardofantolino/MLDLproject4

1. Introduction

The project is divided into 4 parts that we can resume as:

- Study of the literature of IL and SS
- Implementation of a Real-Time SS architecture
- Inclusion of the previous architecture in a IL setting
- Introduction of a contribution

2. Study of the literature

2.1. Semantic Segmentation

If we look at the history of Computer Vision we can clearly see a trend. Firstly, we started to ask to algorithms to recognize what they were seeing in an image, this is classification. The next step was to ask where the objects are, and this is object detection using bounding boxes. Semantic Segmentation goes beyond and also answers the question: "What is the shape of the object?" (Figure 1). Therefore Semantic Segmentation is an important and complex topic in Computer Vision. Thanks to techniques like FCNs and human annotated datasets, we made huge steps forward in this field and we are able to achieve satisfactory results in offline scenario.

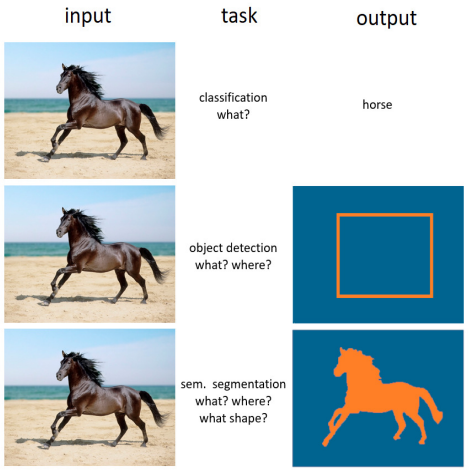

Figure 1. From image-level classification to pixel-level classification. Semantic Segmentation is a dense task that assigns a label to each pixel. In SS field we need to introduce a new and special class: the background class. Any pixel that doesn't belong to other classes is associated with this special class.

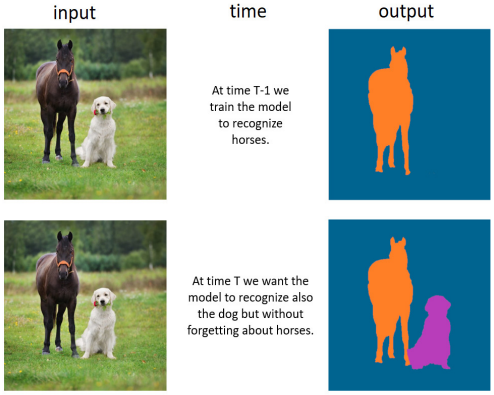

Figure 2. Here we represent the concept of Incremental Learning. This field of research consists in increasing the knowledge of a model during time. The main issue is the so called Catastrophic Forgetting. While the model learns new task, it easily forgets about what it learnt before.

2.2. Incremental Learning

The problem with Semantic Segmentation arises when we try to insert the deep neural networks in a more realistic context as the Incremental Learning one. Basically, we want to increase the knowledge of our model adding new classes to recognize, as shown in Figure 2. A simple strategy would be to re-train every time the model with the previous data plus the new one, but in many cases it is unfeasible. Therefore we need to respect the following memory constraints: we cannot store previously seen examples and we cannot increase the size of our model. Given this assumption, if we implement some naive algorithm we will easily fall into the problem of Catastrophic Forgetting. It means that while the model learns new tasks, it will loose the skills acquired before. During the project we will see different techniques

in order to tackle the issue.

3. Implementation of the BiSeNet architecture in an offline scenario

In this step, we need to familiarize with the architecture developed in [1]. It is called BiSeNet and it stands for Bilateral Segmentation Network. The main idea behind this paper is the introduction of a new architecture that tackles the following critical topics in SS: spatial information and receptive field. The authors developed the so called Spatial Path (SP) and Context Path (CP) in order to solve the previous issues respectively. This architecture was used because we had limited resources in terms of time, computation and GPU allocation. BiSeNet is a suitable trade off because it is able to provide performances comparable to DeepLab, but in a fewer amount of time.

We started our analysis using ResNet18 as ContextPath of our model.

We performed our studies on the dataset PascalVOC2012. It includes 20 object classes plus the background class.

3.1. Batch size and Crop size

The first two parameters that we wanted to optimize were the batch size and the crop size. We want to underline that our aim in this part was to completely fill the GPU at our disposal and not maximise the performances in terms of accuracy and "mean Intersection over Union" (mIoU). So, we had to find a trade off between the 2 parameters.

Usual values of batch size are: 16, 32 and 64; while a good range for the crop size is [320;512]. We started with a trial and error approach, basically increasing and decreasing the values of batch size and crop size if we encountered an "out of memory" error or if we where able to proceed with the computation. Then, to keep track of the memory usage we founded "wandb" and by importing this package we were able to precisely see how much of the available GPU we were using. The results are reported in Table 1.

Table 1. Optimization process for batch size and crop size.

Attempt	batch size	crop size	GPU allocated
1	16	344	40%
2	16	368	43%
3	32	368	72%
4	32	512	78%
5	64	392	96%
6	96	320	96%

Given that the maximum usage of GPU was equal for the last two cases, we choose the best in terms of performances. The precisions were basically equal, but the setting with batch=64 and crop=392 gave us the best mIoU, equal to 0.58. So, at the end, attempt 5 was our final choice.

3.2. Learning rate

After we tailored the mini-batch size and the crop size, we focused our attention to the learning rate (LR), that is one of the most important parameters in the algorithm since it defines the step size of our descent.

We didn't perform a classic grid search, but given that we had little time we used the following approach: first we compared three different learning rates: 10^{-1}, 10^{-2} and 10^{-3}. We selected 10^{-2} as the best one in terms of precision and mIoU. Then we multiplied it for the following values: 2.5, 5.0 and 7.5.

Looking at the results our final choice is:

$$LR_{best} = 5.0 \times 10^{-2}$$

To summarize, with resnet 18, batch size 64, cropsize 392 and the learning rate written above we obtained a precision of 0.89 and a mIoU equal to 0.59.

3.3. Backbone

The next topic that we had to address was the "backbone" that basically is the CP of our BiSeNet. We wanted to achieve a large receptive field and also efficient computation. ResNet is a suitable choice given our limited resources in terms of GPU and time. ResNet is also a lightweight model that fits well with our project.

We compared the performances of ResNet18 and ResNet101. The difference with these two architectures is the depth. As a matter of fact ResNet101 has more layers with respect to ResNet18, so more parameters and it uses an higher amount of GPU. A quite obvious issue arise. Given that ResNet 101 is more "heavy", we need to tailor and adapt the batch size in order to avoid the "out of memory" error. After updating the batch size, we used an empirical formula to adapt the learning rate:

$$LR_{new} = \frac{LR_{old}}{\frac{Batch_{old}}{Batch_{new}}}$$

To better analyze the problem, we also developed the Context Path with ResNet50. In Table 2 you can find the results.

Table 2. Comparison of the results with different ResNet as CP. BK=BackBone, BS/CS=Batch and Crop Size, LR=Learning Rate

BB	BS	CS	LR	precision	mIoU
18	64	392	5.0×10^{-2}	88.7%	0.59
50	40	392	3.1×10^{-2}	90.5%	0.65
101	32	392	2.5×10^{-2}	91.0%	0.67

Table 3. Data Augmentation results in terms of mIoU for ResNet50 and Resnet 101

	ResNet50		ResNet101	
epoch	**No Aug**	**Aug**	**No Aug**	**Aug**
10	62.1	47.7	66.2	53.6
20	63.4	61.2	66.0	65.9
30	64.1	64.9	66.0	68.7

3.4. Data Augmentation

As final step, it was required to perform some Data Augmentation (DA) technique. We need to say that as default we already used crop size (not centered but random). For our previous experiments we had also the classical random horizontal flip. Usually if you use a deeper architecture you need more data and if you have a limited dataset you should augment it with some techniques. This is why we studied data augementation just for ResNet50 and Resnet101. You can find the results in ternms of mIoU in Table 3. As data augmentation we used the following techniques: Pad Center Crop, Random Rotation, Color Jitter and Random Horizontal Flip. We observed that the more Data Augmentation we apply, the more epochs we need to find good results. At the end, We didn't find huge differences for ResNet50 because we go from mIoU=64.1 when we had no data augmentation to mIoU=64.9 with data augmentation. For ResNet101 we get a satisfactory improvement, because without DA mIoU is equal to 66.0, while with DA the value increases to 68.7 (2.7 points).

4. BiSeNet in a Incremental Learning setting

In this step, we have to change from static learning to Incremental Learning. Therefore we need to adapt BiSeNet to this scenario. Instead of inserting the architecture in a additional module, we decided to change the architecture developing an Incremental BiSeNet. The main difference is that the number of classes is now a list, where each element corresponds to a specific step and the associated value is the number of classes that the model needs to learn at that step. So we need to adapt the modules that need the number of classes as input. Instead of changing each module to a module list, we introduced a fixed number of channels as output channels for the Supervision modules and as number of classes for the Feature Fusion Module. The final classifier is instead a module list where at each step we have a different convolutional layer, where the output channels are equal to the number of classes at that step. To be sure that the strategy of using the fixed number of channels would not worsen the previous obtained result, we performed a non-regression test: we introduced this change in the static BiseNet and observed that it produced similar results.

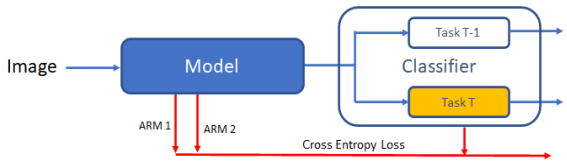

Figure 3. Fine Tuning

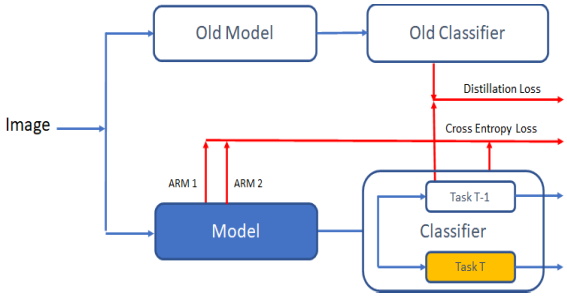

Figure 4. Learning without Forgetting

4.1. Fine Tuning

The Fine Tuning (FT) strategy (Figure 3) consists in extending the old classifier adding randomly initialized weights for the new task. Then during the training phase the architecture optimizes its parameters from the original values to minimize the loss in the new task.

4.2. Learning without Forgetting

Learning without Forgetting [4] (Figure 4) is a method that during the training is able to learn parameters that are discriminative for the new task, while preserving the ability to recognize also the elements of the old tasks. It can achieve this behaviour thanks to the introduction of the concept of Knowledge Distillation. Knowledge Distillation allows the new model to keep the output closer to the prediction of the old one on the new training data. The final Loss is obtained by combining the Cross Entropy Loss computed on the output and the Knowledge Distillation Loss. We want to underline that during the training of the new task, the model doesn't use samples from already learnt classes.

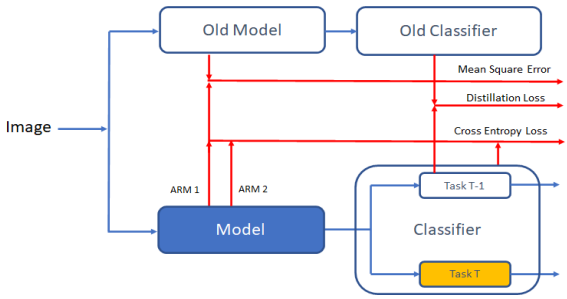

Figure 5. Incremental Learning Techniques

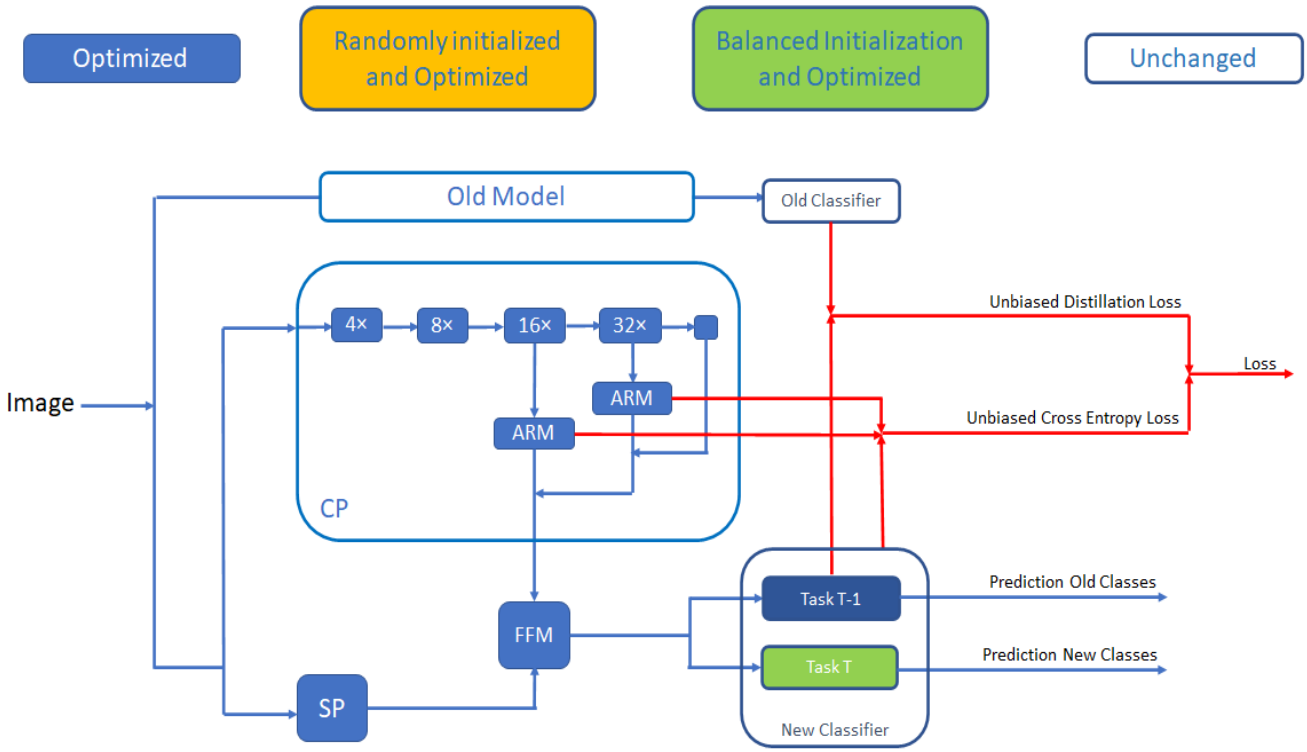

Figure 6. The complete structure of Incremental BiSeNet with MiB protocol. In the picture you can see how we performed the MiB protocol to train our model.

4.3. Incremental Learning Techniques

The main novelty introduced in [5] is the Knowledge Distillation Loss applied to the intermediate feature space. (Figure 5)

4.4. Modeling the Background

The main contribution of the paper [2] are threefold:

- revisiting Cross-Entropy Loss
- revisiting Distillation Loss
- specific classifier's parameters initialization

The standard Cross-Entropy Loss

$$L_{ce}^{\theta^t}(x,y) = -\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} log\ q_x^t(i,y_i)$$

does not consider that the training set $\mathcal{T}^t$ only contains information about novel classes and so in the background class we can find pixels associated with previously seen classes. In this conditions, the Catastrophic Forgetting is even enhanced.

To avoid this issue the following change is done, in order to predict the new class and account for the uncertainty of the background class:

$$L_{ce}^{\theta^t}(x,y) = -\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} log\ \tilde{q}_x^t(i,y_i)$$

where:

$$\tilde{q}_x^t(i,c) = \begin{cases} q_x^t(i,c) & if\ c \neq b \\ \sum_{k\in\mathcal{Y}^{-1}} q_x^t(i,k) & if\ c = b. \end{cases}$$

Concerning the distillation loss, the previous works did not take into account that the background class is shared among different learning steps: annotations for background in a given step might include pixels of classes that will be learnt in the following steps. They tackled the issue by updating the standard Distillation Loss as follows:

$$L_{kd}^{\theta^t}(x,y) = -\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\sum_{c\in\mathcal{Y}^{t-1}} q_x^{t-1}(i,c) log\ \hat{q}_x^t(i,c),$$

where

$$\hat{q}_x^t(i,c) = \begin{cases} q_x^t(i,c) & if\ c \neq b \\ \sum_{k\in\mathcal{C}^t} q_x^t(i,k) & if\ c = b. \end{cases}$$

Finally, in order to avoid the misalignment among the features extracted by the model and the randomly initialized

parameters of the classifier itself, they propose to initialize the parameters in such a way that the probability of the background at time $t-1$ is uniformly spread among the classes to be learned at time t. Looking at the performances in Table 2 we decided to use ResNet 101 as CP. In Table 4 we summarized the results in terms of mIoU obtained by our Incremental BiSeNet following the different methods. We did this for two different tasks: 15-5, where the model learns 15 classes and then 5 classes all at once; and 15-5s, where, after the first step, the model learns 5 classes sequentially. The joint task represents the offline setting, where we feed to the model all the classes at once. This represents the Upper Bound, the best result possible. As we expected with FT, we observe good result for the new tasks while we suffer from Catastrophic Forgetting for the old tasks. In LWF we tackle the problem with Distillation Knowledge and the performances improved. Regarding ILT, we did not have encoder-decoder (body-head) and so, as intermediate features we used the output of the first ARM and we had to tailor the λ parameter for the corresponding loss. Finally, MiB as expected seems to be the best method to deal with the phenomenon of Catastrophic Forgetting.

Table 4. Method results

	15-5			15-5s		
	1-15	**16-20**	**all**	**15-5**	**16-20**	**all**
FT	0.336	29.66	0.766	0.000	1.479	0.369
LWF	50.09	32.22	45.62	5.806	11.37	7.198
ILT	54.88	33.28	49.48	4.742	5.509	4.933
MIB	65.08	39.48	58.68	36.67	9.989	30.00
Joint	73.31	64.64	71.15	73.31	64.64	71.15

The complete structure of Incremental BiSeNet with MiB protocol is represented in Figure 6.

5. Our Contribution

In this section of the project we need to introduce our own new contribution in the field of Semantic Segmentation in Incremental Learning setting. We had different choices and we analyzed multiple options. At the end we decided to introduce a change in the Unbiased Cross Entropy Loss in order to improve the performances of our model.

We took inspiration from [3]. Basically we wanted to extend the concept of Focal Loss and apply it into the MiB Unbiased Cross Entropy Loss.

In order to explain the main concept of the new loss, you can take a look to Figure 8, which is a simple analysis of how the loss changes with respect to the certainty of the predictions of the model on the example image 7. It is a simple image made by 20×20 pixels that represents a boat in the ocean. We have 3 main classes to recognize: the boat, the ocean and the cloud. Everything that does not belong to this 3 classes will be assigned to the background class as in SS. In Figure 8 for each class we can see the prediction certainty of the model for each pixel (yellow corresponds to full certainty). In the first column the model assigns labels with full certainty, so the two losses are equal. In the following columns the certainty keeps decreasing, so the difference between the values of the Cross Entropy Loss and Focal Loss increases.

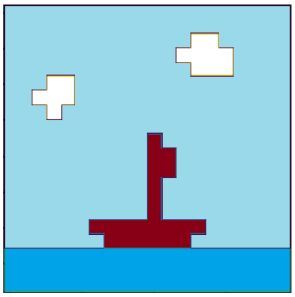

Figure 7. The example image with 3 classes(boat, ocean, cloud) plus the background.

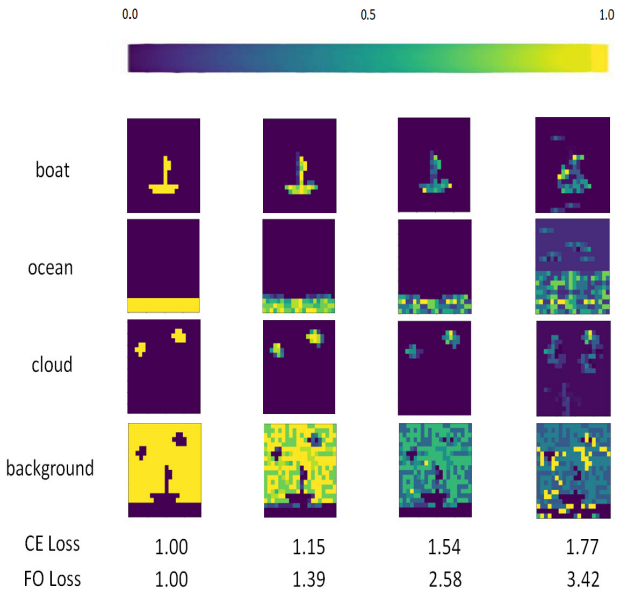

Figure 8. The bar at the top represents the color scale of the percentage from 0 to 1. If the model is certain of the presence of a given class it will appear a yellow pixel, otherwise a darker one. At the bottom of the image there is the comparison of the Standard Cross Entropy Loss and the Focal Loss.

5.1. Focal Loss

The main concept that are behind the introduction of the Focal Loss are the following:

- Reduce the relative loss for well-classified examples.
- Increase the relative loss on hard, misclassified examples.

Focal Loss can be seen as a variation of Balanced Cross-Entropy. To explain it we will focus on the binary case for semplicity.

$$L_{BCE}(p_t) = -\alpha_t log(p_t).$$

where

$$p_t = \begin{cases} p & if\ y = 1 \\ 1 - p & otherwise. \end{cases}$$

$y \in \{+1, -1\}$ specifies the ground-truth class and $p \in [0, 1]$ is the model's estimated probability for the class with label $y = 1$. The α_t parameter is defined analogously to how we defined p_t and it is used to tune false negatives and false positives.

The Focal Loss modifies the Balanced Cross-Entropy Loss by adding a modulating factor $(1 - p_t)^\gamma$:

$$L_{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma log(p_t)$$

Therefore, when an example is misclassified and p_t is small, the factor $(1 - p_t)^\gamma$ is near 1 and the loss is unaffected. Instead, when p_t is near 1, the factor is near 0 and the loss is down-weighted. Moreover the parameter γ is used to regulate at which rate "easy" examples are down-weighted.

In this way, we aim to penalize more and concentrate the effort of the model on the images where the prediction is not "certain", while we give little importance to the images that we classified with an higher value of certainty.

In Table 5 results are reported: we used Focal Loss for the task 15-5 tuning different values of α and γ. We decided to use 20 epochs in order to reduce the computing time, since after that number of epochs the loss keeps decreasing but mIoU does not increase.

Table 5. MIoU results tuning α and γ parameters. UNCE stands for Unbiased Cross-Entropy and represents our baseline, while FUCE means Focal Unbiased Cross-Entropy Loss.

	alpha	gamma	15-5	16-20	all
UNCE	/	/	67.66	30.47	58.36
FUCE	1	1	67.27	28.46	57.57
FUCE	1	1.5	67.73	28.91	58.02
FUCE	1	2	67.27	28.17	57.50
FUCE	1	2.5	67.95	27.44	57.82
FUCE	2	1.5	67.29	32.91	58.70
FUCE	3	1.5	65.77	33.08	57.60

We observe that increasing α, mIoU increases in the last 5 classes, but decreases in the first 15; instead, increasing γ, mIoU slightly increases in the first 15 classes, but decreases in the last 5.

6. Conclusion

We identified as possible future development the implementing of the Distance Map Penalized Cross Entropy Loss (DPCE), which is another distribution-based loss derived from cross entropy. It aims to minimize dissimilarity between two distributions. Distance Maps can be defined as distance (euclidean, absolute, etc.) between the ground truth and the prediction. DPCE loss weights cross entropy by distance maps and it aims to guide the network's focus towards hard-to-segment boundary regions.

$$L_{DPCE} = -\frac{1}{N}(1 + D)\sum_{i \in N}\sum_{j \in K} y_j log\hat{y}_j$$

where N is the number of samples and K the number of classes.

References

[1] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, Nong Sang. *BiSeNet: Bilateral Segmentation Network for Real-time Semantic Segmentation*. 2018.

[2] Fabio Cermelli, Massimiliano Mancini, Samuel Rota Bulò, Elisa Ricci, Barbara Caputo. *Modeling the Background for Incremental Learning in Semantic Segmentation*. 2020.

[3] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollàr *Focal Loss for Dense Object Detection*. 2018.

[4] Zhizhong Li, Derek Hoiem *Learning without forgetting*. 2017.

[5] Umberto Michieli, Pietro Zanuttigh *Incremental learning techniques for semantic segmentation.* 2019.