Deep Segmentation: Using deep convolutional networks for coral reef pixel-wise parsing

Aljoscha Steffens[1], Antonio Campello[1,2], James Ravenscroft[1], Adrian Clark[2], and Hani Hagras[2]

[1]Filament AI, United Kingdom
[2]University of Essex, United Kingdom

Abstract. In this paper, we describe a deep-convolutional network based method to segment coral reef images into different types of substrates. The method described in the paper includes data preparation, model summary, specific techniques to deal with class imbalance, and downstream post-processing computer vision tasks, such as morphological operations and polygon generation from pixel segmentation. We present the results of our method in the ImageCLEFcoral pixel-wise parsing task, evaluated across the different classes of substrate.

1 Introduction

Semantic segmentation models have received significant attention in computer vision due to their applicability in medical imagining, autonomous driving, and full-scene understanding. In this paper, we consider the ImageCLEF 2019 pixel-wise parsing competition [4], [9], which consists of segmenting pictures from coral reefs into 13 different substrates. We are particularly interested in evaluating the applicability of deep convolutional neural networks (DCNNs) to the coral images, taken under real conditions in the ocean. A model that performs well on the task of automatic segmentation of corals could be beneficial to the conservation of reefs by measuring the amounts of different corals, their condition and other characteristics.

This paper is organised as follows. In section 2 we explore the ImageCLEFcoral dataset [4], how to split it into training and validation set in order to keep the distributions balanced, as well as a data augmentation approach. In section 3, we describe DeeplabV3 [6], a DCNN designed for semantic segmentation, alongside with our pipeline. Our method includes post-processing tasks such as morphological operations and polygon filling. Training, bootstrapping and inference are also described. In section 4 we discuss possible routes to improve the results.

2 Data

The data that was provided consisted of 240 images of size $4032 \times 3024 \times 3$ as well as a text file containing the polygons in the images for the following classes/substrates:

Hard Coral - Branching	*Soft Coral*
Hard Coral - Submassive	*Soft Coral - Gorgonian*
Hard Coral - Boulder	*Sponge*
Hard Coral - Encrusting	*Sponge - Barrel*
Hard Coral - Table	*Fire Coral - Millepora*
Hard Coral - Foliose	*Algae - Macro or Leaves*
Hard Coral - Mushroom	

Table 1: Substrate names

We mapped the 13 substrates to integers $\{0, 1, \ldots 13\}$ (the background corresponding to 0) and created a 4032×3024 integer matrix for each image defined as

$$M_{ij}^{k} = c \text{ if pixel } i, j \text{ corresponds to substrate } c, \tag{1}$$

with ties broken arbitrarily. The corresponding matrix acts as a per-class "mask" for each substrate, an example can be seen in figure 1.

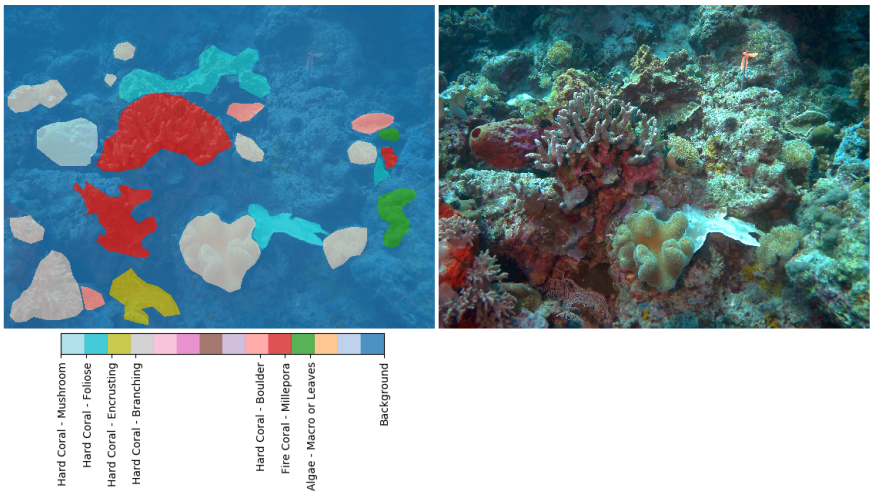

Fig. 1: Substrate mask and image for id 2018_0714_112417_024

The class-distribution of the pixel reflects the naturally occurring composition of corals in the photographed area and is thus highly imbalanced, see figure 2. Also, the class-distributions between two images can vary strongly as only a limited selection of coral types will be present in each image. Both the overall class-imbalance and the difference in inter-image class-distributions need to be taken into account and are discussed in section 3.2 and 2.1 respectively.

2.1 Data Split

The data was split into a training and validation set with 204 images (85%) for training and 36 for validation. Due to reasons discussed in section 2.2, we chose to split the data on a per-image basis. This posed a problem given the difference in inter-image class-distributions and the relatively small number of images: for a random split it would be highly likely that the overall, training, and validation class-distribution would differ strongly which would have an effect on the evaluation of the model performance.

In order to achieve balanced distributions for the training and validation set, we created N training and validation sets of constant size, with randomly selected images and compared the resulting training/validation distributions by using a cosine distance that is weighted by the overall class-distribution.

$$dist(v^1, v^2, w) = \frac{\sum_{i=1}^{d} {w_i}^2 \times v_i^1 \times v_i^2}{\sqrt{\sum_{i=1}^{d} w_i \times v_i^1} \sqrt{\sum_{i=1}^{d} w_i \times v_i^2}} \tag{2}$$

From those N splits, we chose the split that resulted in the lowest distance. Afterwards, we selected one item from each set that, if swapped, resulted in the biggest decrease in the weighted cosine distance. Swaps were performed until there was no further decrease possible by swapping individual items. The whole procedure - N random splits, choosing the best split, optimise by swapping - was performed several times in order to increase the chance of finding an good final split.

While the given approach does not result in an optimal solution, it is fast and did achieve satisfying result for the given task. The three distributions (overall, training, validation) can be seen in figure 2.

2.2 Data Augmentation

Splitting the data on image level was mainly done due to how we prepared the data before feeding it into the Neural Network. With their large spatial dimensions that resulted in both rich details and many different coral types per image, it seemed sensible to use random cropping as a main preparation step. For each image that was loaded into memory during training, 16 random crops with square sizes between 400×400 and 1400×1400 were performed. The crops were then bi-linearly scaled to 256×256 and randomly flipped in vertical and

horizontal direction before they were fed into the network. Both the flipping and the random cropping served as a data augmentation method that ensured that the network was exposed to a variety of relative coral sizes, orientations, and image-compositions. This way, the network never saw the exact same image twice which reduced overfitting.

For the validation data, random crops and re-scaling were done prior to training the network and thus always the same. This was done to make the metrics that were computed after each epoch comparable.

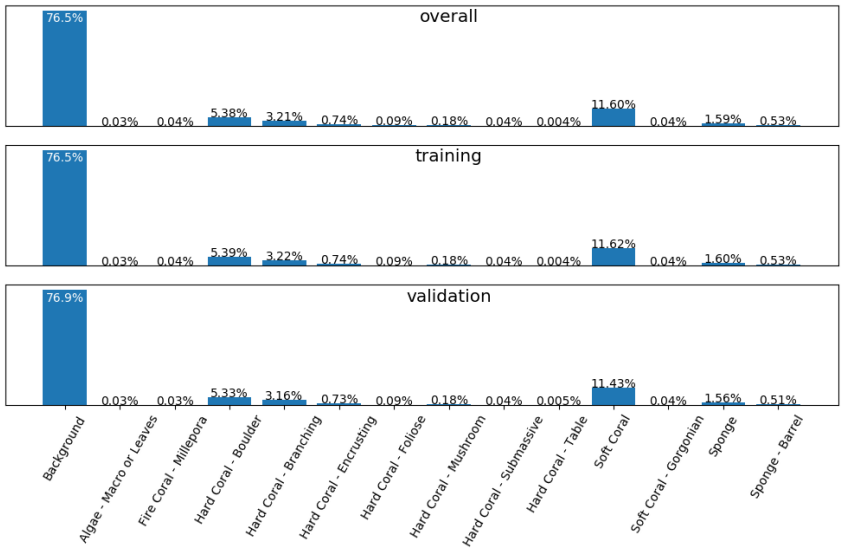

Fig. 2: Class distributions overall and for training and validation split

3 The model

We used DeeplabV3 [6], a deep convolutional neural network that improves over existing networks. In particular, DeeplabV3 extends Deeplab [5] and avoids the need of a post-processing machine learning model (such as conditional random fields). Nevertheless, since the challenge is evaluated over polygons, we needed to apply image processing techniques (in particular, morphological transformations and region-filling algorithms) in order to generate the final file. Note that the polygon-filling operations have more degrees-of-freedom than the training pre-processing, therefore adding extra parameters to the model. These operations are described in more details in 3.5. A high-level operational diagram of the model and evaluation can be found below. More details will be described in the next sections.

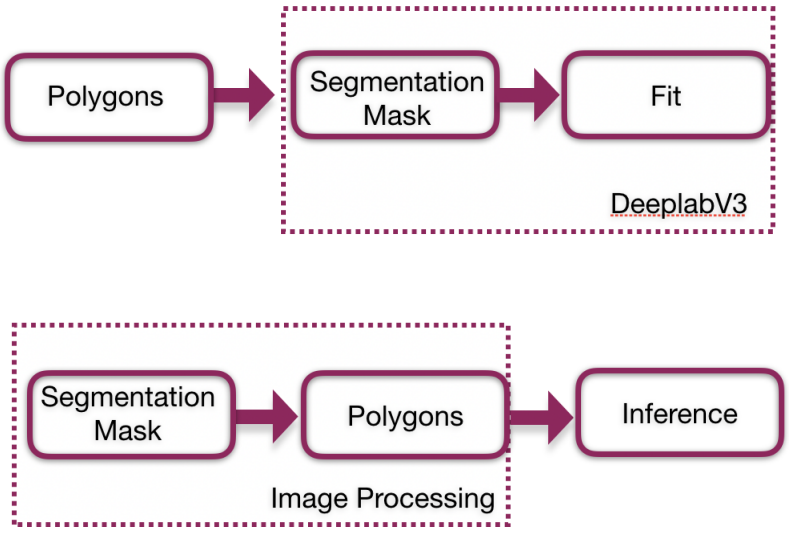

Fig. 3: Training and inference flows.

3.1 DeeplabV3

Deeplab V3 is a Deep Convolutional Neural Network (DCNN) for semantic image segmentation proposed by Chen et al in [6] in 2017. It is a state-of-the-art network architecture that, with pretraining on the ImageNet [7] and JFT-300M [10] dataset resulted in a mIOU of 86.9% on the PASCAL VOC 2012 test set [6]. The model consist of two parts: The first part is a feature extracting backbone that is not strictly limited to be of a given type - in the paper the authors use a ResNet-50 and ResNet-101 but other architectures can be employed as well. The second part is where the novelty happens with an extensive use of atrous convolutions. An atrous convolution filter kernel layout is defined by the normal size of say 3x3 and in addition to that, an atrous rate that specifies how many 0-values are between the individual filter entries along the spatial dimensions. With 0-values between, the atrous convolution would be the same as a normal convolution; inserting one zero between two neighbouring values in a 3x3 convolution would make it have the same receptive field as a 5x5 convolution, while only employing 9 weights instead of 25. Deeplab V3 uses atrous convolutions for constructing feature pyramids. Feature pyramids are used to combine features from different scales into one feature map, which is done by using atrous convolutions with different rates on the same feature map and concatenating the individual outputs to a new feature map. For our submission we used a PyTorch implementation of DeepLab V3 found from [3] with a ResNet101 backbone [8] and an output-stride of 16

Prior to polygon-filling post-processing, the model outputs, for every pixel, a probability that such pixel belongs to a class, or more formally:

$$f_{ij}^{k}(c) = \text{probability that pixel } ij \text{ on image } k \text{ belongs to class } c \tag{3}$$

3.2 Class imbalance and weighted loss function

As discussed in section 2, the class-distribution of the data is highly skewed. This is a problem, as the model will emphasize more on classifying frequent classes correctly to achieve lower errors if no counter-measures are in place. One approach that is often used - and that was used here as well - is to weigh the loss function based on the class distribution. We used a pixel-wise cross-entropy loss and weighted the individual components with following weights:

$$w(c) = \frac{1}{\log(\alpha + p(c))} \tag{4}$$

with $p(c)$ being the relative occurrence of class c and α being a hyper-parameter that scales the weights (in our submission $\alpha = 1.025$ yielded good results). This was done as there are orders of magnitudes between the individual relative occurrences, and the model would over-emphasize on the infrequent classes if we just used the reciprocal.

The final cross-entropy loss is as follows:

$$\frac{1}{N \times \text{height} \times \text{width}} \times \sum_{k=1}^{N} \sum_{i=1}^{\text{height}} \sum_{j=1}^{\text{width}} \sum_{c=1}^{\text{C}} -y_{ij}^{k}(c) \times w(c) \times \log\left(\frac{\exp(f_{ij}^{k}(c))}{\sum \exp(f_{ij}^{k}(c))}\right), \tag{5}$$

where $f_{ij}^{k}(c)$ and $y_{ij}^{k}(c)$ describe the predicted confidence (f) and ground truth (y) for image k at position ij and class c respectively and N is the number of images that are included in the loss.

3.3 Training and Bootstrapping

We trained the Neural Network for 50 epochs, with a batch size of 32 (2 images per batch, 16 crops per image) on a Nvidia GeForce GTX 1080 Ti. After the training, we used the network to predict the training images and cropped out areas where the network was particularly bad. The network was then trained on those images for another 30 epochs.

3.4 Inference

In order to predict a full-sized image at inference time, we used a sliding window approach. With window-sizes of 500×500, 1000×1000, and 1500×1500 corresponding step-sizes of 400, 800, and 1200 we cut each 4032×3024 image into 112 partially overlapping sections. Each section was then scaled to 256×256

and fed into the Neural Network. The results were scaled to their original size and added at their respective position to a $4032 \times 3024 \times 14$ confidence matrix C. For each position $C_{i,j}$ the number of votes were stored (meaning how often a given pixel was predicted) so that the average confidence could be calculated subsequently. The final classification for pixel i, j was then given by

$$c = \underset{k \in \{0,\ldots,13\}}{\operatorname{argmax}} \; C_{i,j}(k) \tag{6}$$

By using sliding windows with different window-sizes we made sure that each pixel was predicted at several different resolutions and thus also with different amounts of context.

3.5 Post-processing

After calculating the classification mask for a predicted image, we used several basic computer vision algorithm for post-processing and transforming the data into the given submission format.

1. Find connected components.
2. Morphological opening with kernel size $= (31, 31)$
3. Morphological closing with kernel size $= (31, 31)$
4. Flood fill
5. Polygon approximation using Douglas-Peucker algorithm [2], with maximum distance to correct output ε equals 0.1% of the contour arc-length for that connected component.

We used the OpenCV 3.4.2 implementations [1] of the corresponding algorithms.

4 Final results and further work

The average intersection over union for all classes, as reported in the official ImageCLEF 2019 Coral competition, over the test set, is described in Table 2. Soft corals and hard corals (boulder) performed relatively well in comparison to the other classes, which was expected, anticipated by the class abundance, as shown in figure 2. Surprisingly, mushroom pixels have been correctly identified 21%, in spite of the small number of samples. A full analysis and visualisation of the results per class is left for future investigation.

There are a number of areas where our pipeline can potentially be improved in the future and things that can be investigated:

- Increasing the input size to the model
- Increase the batch size
- Test different model backbones
- Tuning hyper-parameters (both for training and for post-processing)
- Investigate the impact of crop sizes
- Investigate the impact of bootstrapping
- Try different methods to counteract the class-imbalance

Substrate	mIoU (%)
Hard Coral - Branching	9.58
Hard Coral - Submassive	0.0
Hard Coral - Boulder	16.59
Hard Coral - Encrusting	4.46
Hard Coral - Table	0.0
Hard Coral - Foliose	0.65
Hard Coral - Mushroom	21.9
Soft Coral	13.0
Soft Coral - Gorgonian	1.86
Sponge	5.73
Sponge - Barrel	8.89
Fire Coral - Millepora	0.0
Algae - Macro or Leaves	0.07

Table 2: Accuracy per class

References

1. Open source computer vision library 4.1.0. https://docs.opencv.org/4.1.0/, 2019.
2. Opencv contour features. https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html, 2019.
3. pytorch-deeplab-xception. https://github.com/jfzhang95/pytorch-deeplab-xception, 2019.
4. J. Chamberlain, A. Campello, J. P. Wright, L. G. Clift, A. Clark, and A. García Seco de Herrera. Overview of ImageCLEFcoral 2019 task. In *CLEF2019 Working Notes*, volume 2380 of *CEUR Workshop Proceedings*, 2019.
5. L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(4):834–848, 2018.
6. L. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.
7. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
8. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.
9. B. Ionescu, H. Müller, R. Péteri, Y. D. Cid, V. Liauchuk, V. Kovalev, D. Klimuk, A. Tarasau, A. B. Abacha, S. A. Hasan, V. Datla, J. Liu, D. Demner-Fushman, D.-T. Dang-Nguyen, L. Piras, M. Riegler, M.-T. Tran, M. Lux, C. Gurrin, O. Pelka, C. M. Friedrich, A. G. S. de Herrera, N. Garcia, E. Kavallieratou, C. R. del Blanco, C. C. Rodríguez, N. Vasillopoulos, K. Karampidis, J. Chamberlain, A. Clark, and A. Campello. ImageCLEF 2019: Multimedia retrieval in medicine, lifelogging, security and nature. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, Proceedings of the 10th International Conference of the CLEF Association (CLEF 2019), Lugano, Switzerland, September 9-12 2019. LNCS Lecture Notes in Computer Science, Springer.

10. C. Sun, A. Shrivastava, S. Singh, and A. Gupta1. Revisiting unreasonable effectiveness of data in deep learning era. *ICCV*, 2017.