Pontifícia Universidade Católica
do Rio de Janeiro

Ian Monteiro Nunes

Open-set semantic segmentation for remote sensing images

Tese de Doutorado

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática.

Advisor : Prof. Marcus Vinicius Soledade Poggi de Aragao
Co-advisor: Dr. Hugo Neves de Oliveira

Rio de Janeiro
January 2023

Ian Monteiro Nunes

Open-set semantic segmentation for remote sensing images

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática. Approved by the Examination Committee:

Rio de Janeiro, January 4th, 2023

Ian Monteiro Nunes

Graduated in Computer Engineering in 2003 from Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro - Brazil. He also received a Master's degree in Data Science in 2008 from the Computer Science Department of the Pontifical Catholic University of Rio de Janeiro (PUC-Rio). Research areas include Machine Learning, Deep Learning, Domain Adaptation, Computer Vision, Remote Sensing, Clustering and Open Set Recognition.

Bibliographic Data

Monteiro Nunes, Ian

Open-set semantic segmentation for remote sensing images / Ian Monteiro Nunes; advisor: Marcus Vinicius Soledade Poggi de Aragao; co-advisor: Hugo Neves de Oliveira. – 2023.

211 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2023.

Inclui bibliografia

1. Open-set – Teses. 2. Segmentation – Teses. 3. Remote sensing – Teses. 4. Conjunto aberto. 5. Segmentação. 6. Redes Neurais Convolucionais. 7. Sensoriamento remoto. 8. Aprendizado profundo. 9. Ciência de dados. 10. Classificação. 11. Auto-encoder. I. Soledade Poggi de Aragão, Marcus Vinicius. II. Neves de Oliveira, Hugo. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

To my children Julia and Gael and my life partner Maria Luisa.
Thank you for everything you had to give up during these years so that my goals could be achieved.
This achievement is ours!
I love you all!

Acknowledgments

First, I would like to thank my parents for their unconditional support throughout my journey. Without your encouragement and support at all times in my life, everything would be much more difficult. Ivan and Isabel, thank you for being my examples and inspiration.

I also thank my friend and advisor Marcus Poggi for, as he says, disorienting me throughout the process, questioning everything and pushing me towards the most challenging paths.

I would like to thank my friend and co-supervisor, Hugo Oliveira, for whom, without his guidance and help, this work would have taken a very different course.

I would like to thank Professor Jefersson Santos for accepting me into the PATREO research group, and for all the discussions and partnerships.

I would like to thank my great work partner Matheus Pereira for all the hours we spent talking and studying the models, insights and great lessons that were offered to me.

I cannot fail to mention all the affection, support and guidance that my friend Arnaldo Lyrio offered throughout the entire work!

I also would like to thank everyone who somehow contributed to make this period a little lighter and simpler.

Thank you very much!

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Monteiro Nunes, Ian; Soledade Poggi de Aragão, Marcus Vinicius (Advisor); Neves de Oliveira, Hugo (Co-Advisor). **Open-set semantic segmentation for remote sensing images**. Rio de Janeiro, 2023. 211p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Collecting samples that exhaust all possible classes for real-world tasks is usually difficult or impossible due to many different factors. In a realistic/feasible scenario, methods should be aware that the training data is incomplete and that not all knowledge is available. Therefore all developed methods should be able to identify the unknown samples while correctly executing the proposed task to the known classes in the tests phase.

Open-Set Recognition and Semantic Segmentation models emerge to handle this kind of scenario for, respectively, visual recognition and dense labeling tasks. Initially, this work proposes a novel taxonomy aiming to organize the literature and provide an understanding of the theoretical trends that guided the existing approaches that may influence future methods. This work tested the proposed techniques on remote sensing data, establishing new state-of-the-art results for the used datasets. Remote sensing data differs from RGB data as it deals with a plethora of sensors and with a high geographical variation.

Open set segmentation is a relatively new and unexplored task, with just a handful of methods proposed to model such tasks. This work also proposes two distinct techniques to perform open-set semantic segmentation. First, a method called OpenGMM extends the OpenPCS framework using a Gaussian Mixture of Models to model the distribution of pixels for each class in a multimodal manner. Second, the Conditional Reconstruction for Open-set Semantic Segmentation (CoReSeg) method tackles the issue using class-conditioned reconstruction of the input images according to their pixel-wise mask. CoReSeg conditions each input pixel to all known classes, expecting higher errors for pixels of unknown classes.

Qualitative results observation suggested that both proposed methods produce better semantic consistency in their predictions than the baselines, resulting in cleaner segmentation maps that better fit object boundaries. Also, OpenGMM and CoReSeg outperformed state-of-the-art baseline methods on Vaihingen and Potsdam ISPRS datasets.

The third proposed approach is a general post-processing procedure that uses superpixels to enforce highly homogeneous regions to behave equally, rectifying erroneous classified pixels within these regions. We also proposed a novel superpixel generation method called FuSC.

All proposed approaches improved the quantitative and the qualitative results for both datasets. Besides that, CoReSeg's prediction post-processed with FuSC achieved state-of-the-art results for both datasets.

The official implementation of all proposed approaches is available at `https://github.com/iannunes`.

Keywords

Open-set; Segmentation; Convolutional Neural Networks; Remote Sensing; Deep Learning; Data Science; Recognition; Auto-encode.

Resumo

Monteiro Nunes, Ian; Soledade Poggi de Aragão, Marcus Vinicius; Neves de Oliveira, Hugo. **Segmentação semântica de conjunto aberto aplicada a imagens de sensoriamento remoto**. Rio de Janeiro, 2023. 211p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Coletar amostras que esgotam todas as classes possíveis para tarefas do mundo real geralmente é difícil ou impossível devido a muitos fatores diferentes. Em um cenário realista/viável, os métodos devem estar cientes de que os dados de treinamento estão incompletos e que nem todo o conhecimento está disponível. Portanto, todos os métodos desenvolvidos devem ser capazes de identificar as amostras desconhecidas enquanto executam corretamente a tarefa proposta para as classes conhecidas na fase de testes.

Modelos de Reconhecimento de Conjunto Aberto e Segmentação Semântica surgem para lidar com esse tipo de cenário para, respectivamente, tarefas de reconhecimento visual e rotulagem densa. Inicialmente, este trabalho propõe uma nova taxonomia com o objetivo de organizar a literatura e fornecer uma compreensão das tendências teóricas que guiaram as abordagens existentes que podem influenciar métodos futuros. Este trabalho testou as técnicas propostas em dados de sensoriamento remoto, estabelecendo novo estado-da-arte para os resultados dos conjuntos de dados utilizados.

A segmentação de conjuntos abertos é uma tarefa relativamente nova e inexplorada, com apenas um punhado de métodos propostos para modelar tais tarefas. Este trabalho também propõe duas técnicas distintas para realizar a segmentação semântica de conjunto aberto. Primeiro, um método chamado OpenGMM estende a estrutura OpenPCS usando uma mistura gaussianas para modelar a distribuição de pixels para cada classe de maneira multimodal. Em segundo lugar, o método de Reconstrução Condicional para Segmentação Semântica de Conjunto Aberto (CoReSeg) aborda o problema usando a reconstrução condicionada por classe das imagens de entrada de acordo com sua máscara. CoReSeg condiciona cada pixel de entrada para todas as classes conhecidas, esperando erros maiores para pixels de classes desconhecidas.

A observação dos resultados qualitativos mostra que ambos os métodos propostos produzem melhor consistência semântica em suas predições do que as métodos de referência, resultando em mapas de segmentação mais limpos que se ajustam melhor aos limites do objetos. Além disso, OpenGMM e CoReSeg superaram o estado-da-arte estabelecido pelos métodos de referência para conjuntos de dados de Vaihingen e de Potsdam disponibilizados pelo ISPRS.

A terceira abordagem proposta é um procedimento geral de pós-processamento que usa superpixels para forçar regiões altamente homogêneas a se comportarem igualmente, corrigindo pixels mal classificados dentro dessas regiões. Também propusemos um novo método para geração de superpixels chamado FuSC.

Todas as abordagens propostas melhoraram os resultados quantitativos e qualitativos para ambos os conjuntos de dados. Além disso, CoReSeg pós-processado com FuSC estabeleceu um novo estado-da-arte para segmentacao de ambos os conjuntos de dados.

A implementação oficial de todas as abordagens propostas está disponível em `https://github.com/iannunes`.

Palavras-chave

Conjunto aberto; Segmentação; Redes Neurais Convolucionais; Sensoriamento remoto; Aprendizado profundo; Ciência de dados; Classificação; Auto-encoder.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AUROC	Area Under the Receiver Operating Characteristic Curve
BudaNet	Boundless Universal Domain Adaptation Network
CAM	Channel Attention Mechanism
CDF	Cumulative Distribution Function
CV	Campo Verde dataset
C2AE	Class conditioned auto-encoder for open-set recognition
CROSR	Classication-reconstruction learning for open-set recognition
SS	Closed-set Semantic Segmentation
CGDL	Conditional Gaussian Distribution Learning for Open Set Recognition
CRF	Conditional Random Field
CoReSeg	Conditional Reconstruction for Open-Set Semantic Segmentation
CBAM	Convolutional Block Attention Module
CNN	Convolutional Neural Network
DES	Deep Edge Superpixels
DNN	Deep Neural Network
DenseNet	Densely Connected Convolutional Networks
DWT	Discrete Wavelet Transforms
EVT	Extreme Value Theory
FiLM	Feature-wise Linear Modulation
FPR	False Positive Rate
FCN	Fully Convolutional Network
FuSC	Fusing Superpixels for Semantic Consistency
GMM	Gaussian Mixture of Models

GAN	– Generative Adversarial Network
HOG	– Histogram of Oriented Gradients
HSI	– Hyper-Spectral Images
KKC	– Known Known Class
KUC	– Known Unknown Class
LDA	– Linear Discriminant Analysis
LOCO	– Leave One Class Out
ML	– Machine Learning
NLP	– Natural Language Processing
NN	– Neural Network
OpenFCN	– Open Fully Convolutional Network
OpenGMM	– Open Gaussian Mixture of Models
OpenPCS	– Open Principal Component Scoring
OSR	– Open-Set Recognition
OSS	– Open-Set Semantic Segmentation
OE	– Outlier Exposure
OOD	– Out-of-Distribution
PiCoCo	– Pixelwise Contrast and Consistency Learning
PCA	– Principal Component Analisys
RNN	– Recurrent Neural Networks
ROC	– Receiver Operating Characteristic
ResNet	– Residual Network
SAM	- Spatial Attention Mechanism
SGD	– Stochastic Gradient Descent
SPS	– Superpixel Segmentation
SVM	– Support-vector machine
SAR	– Synthetic Aperture Radar

TPR	True Positive Rate
UKC	Unknown Known Class
UUC	Unknown Unknown Class
VEGP	vanishing or exploding gradient problem
VAE	Variational Auto-Encoder
VGG	Very Deep Convolutional Networks
WideResNet	Wide Residual Network
WRN	Wide Residual Network

1
Introduction

With the evolution of the internet and the immense spread of ubiquitous sensor networks in our society, we observe an increasing volume and dispersion of data available for commercial or academic use. These data still lack methods and studies that allow their application to the most diverse applications in real-world scenarios.

Never before in human history has the term *future shock*, coined by Toffler (1970), made so much sense while our society has never changed so quickly and in such an interconnected way. This interconnection permeates everyone and generates previously unthinkable volumes of data at an ever-increasing speed, allowing data to be widely accessed and shaping a new data-based era (Bentley et al., 2014).

The body of knowledge of image processing incorporated Machine Learning (ML) techniques in recent decades. Shallow image processing techniques were the tools used to extract meaningful information for decision-makers until the emergence of modern neural network architectures called deep neural networks in the early 2010s. The so-called shallow techniques are handcrafted for each use case and execute all process steps sequentially, relying on features extracted by an expert that may not be representative and may lead the algorithm to error.

The use of shallow techniques began for remote sensing images with the acquisition of the first multi-spectral or high-spectral (HSI) remote sensing images in the 1960s. The *toy example* in Figure 1.1presents a shallow image processing pipeline used for decades.

Each step of the shallow pipeline can use many different techniques. As examples, we can cite some well-known methods such as Principal Component Analysis (PCA) (Jolliffe, 1990), Linear Discriminant Analysis (LDA) (Duda et al., 1973), Discrete Wavelet Transforms (DWT) (Bruce et al., 2002), edge detection techniques (Kumar et al., 2013), Histogram of Oriented Gradients (HOG) (Zhu et al., 2006), etc. Many different algorithms became popular with time in learning how to classify data: Decision Trees (Safavian and Landgrebe, 1991), Logistic Regression, support vector machine (SVM) (Cortes and Vapnik, 1995), Random Forests (Breiman, 2001), etc.

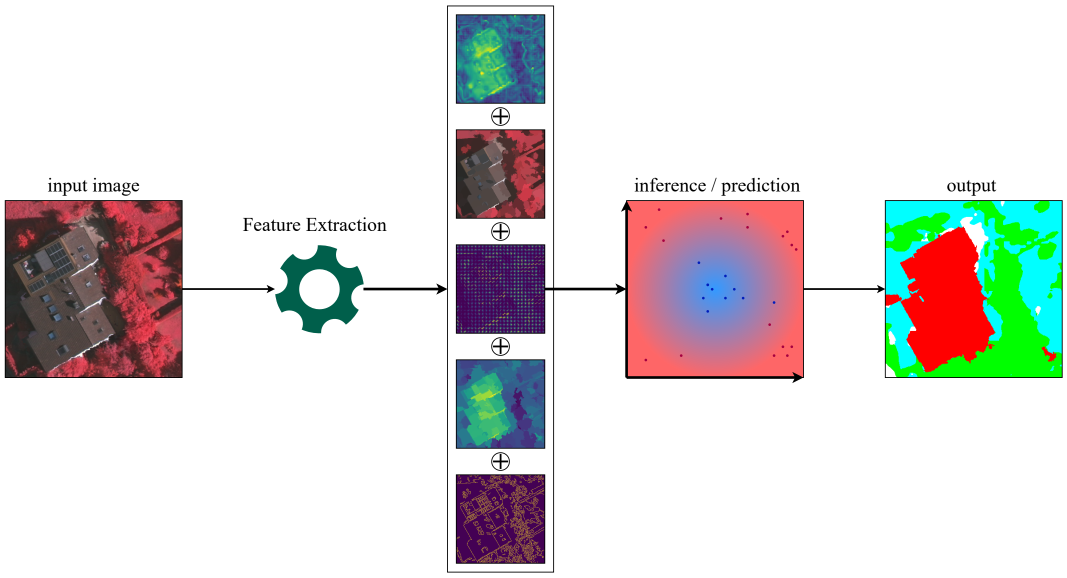


Figure 1.1: Generic example of shallow image learning/processing pipeline.

A major revolution in computer vision research occurred in the last decade, as the AlexNet proposed by Krizhevsky et al. (2012) made deep neural networks (DNNs) the most used approach. DNNs have significantly improved many aspects of visual recognition (Bendale and Boult, 2016), achieving human-level performance for a multitude of tasks. Despite the many advances, there are still difficult challenges when facing real-world problems (Sun et al., 2020).

DNNs changed the shallow pipeline used for decades, replacing the feature extraction and statistical inference steps with a DNN model capable of doing both steps by itself, learning to optimize the desired task in an end-to-end manner.

Traditional vision (RGB) and remote sensing tasks share classification pipelines, but there are crucial differences among the images used. In addition to the differences shown in Table 1.1, the nature of objects captured between RGB and remote sensing images is not the same. The objects represented have different geometry and relationships to their neighbors. There are also physical differences in the environment and the used sensors, like the nature of the noise, and the presence of clouds, among others.

1.1 Motivation

Closed-set semantic segmentation, classification, and recognition tasks are limited due to the difficulty of collecting labeled or classified training samples that exhaust all possible classes in the real world. The expected scenario in real-world problems is Open with objects of classes not seen during

Characteristic	RGB	HSI/ Multispectral	SAR
number of channels	tipically 3	up to thousands	tipically 4
spatial resolution of the images	vary for each input	the same for images from the same sensor	
captured wavelenght spectral width	> 400nm and < 700nm	any range	> 1 cm
number of available sensors	many	few	few
number of available training samples	many	limited	limited
radiometric resolution	8 bits	up to 11 bits	up to 16 bits

Table 1.1: Key differences among RGB, Hyperspectral (HSI)/Multispectral, and Synthetic Aperture Radar (SAR) images.

training that may be submitted to the model during testing or deployment phases (Geng et al., 2020).

Incomplete knowledge of the world during training or the existence of unknown samples during inference is a major unsolved challenge. So-called Open Set Recognition (OSR) tasks have caught the interest of the research community with multiple methods recently proposed for classification problems (Bendale and Boult, 2016; Sun et al., 2020; Oza and Patel, 2019; Cui et al., 2020; Guo et al., 2021). However, few publications have dealt with distinct visual tasks, such as segmentation or object detection.

Semantic Segmentation is a classification problem that classifies every pixel in an image with a semantic label (class) according to Minaee et al. (2021). The Open version of the Semantic Segmentation problem is called Open-set Semantic Segmentation (OSS). It refers to the set of algorithms that address the identification of pixels of unknown or out-of-distribution (OOD) classes at inference time while correctly classifying pixels of the known classes learned in training (Oliveira et al., 2021). OSS is an inherently harder problem due to its dense labeling nature compared to Open-set classification. It is hard to learn open-set semantic segmentation precisely in real-world scenarios (Brilhador et al., 2021). This fact may explain why there is still a gap in the literature with only a handful of articles tackling the issue (Cui et al., 2020).

1.2 Problem Definition

This work seeks to find ways to make semantic segmentation robust in real-world scenarios where not all classes are known at training time but may appear at test time.

All reported methods for OSS can be viewed as post-processing techniques since they used pre-trained DNN backbones to extract features and train another classifier, set a threshold, or replace the network tail. Post-processing models allow DNN backbone replacement and can be readily adapted to new datasets. As a disadvantage, all reported methods lack semantic consistency and have many pixels misidentified as OOD pixels, especially at object boundaries.

In this work, we focus on finding methods that improve segmentation results and semantic consistency for semantic segmentation of open-set semantic segmentation. Improving the semantic consistency of open-set semantic segmentation allows the deployment in real-world scenarios. Models with little semantic consistency deployed in real-world scenarios can lead to avoidable errors and make their use unfeasible.

Bellow, the three research questions that guided the development of our research.

$\mathcal{RQ}_1$: Is it possible to develop a model capable of improving known benchmarks for recognizing OOD pixels?

$\mathcal{RQ}_2$: Could a deep end-to-end model improve semantic consistency while improving known benchmarks for semantic segmentation on open-set scenarios?

$\mathcal{RQ}_3$: Is there a way to improve quantitative results and semantic consistency for existing OSS methods?

1.3 Approach

First, we executed a systematic mapping of the literature proposing a taxonomy to organize and assist in defining the path for our research (Nunes et al., 2022a). With the open-set segmentation publications identified and mapped, we propose three distinct approaches to tackle the research questions. First, based on Open Principal Component Scoring (OpenPCS) (Oliveira et al., 2021) that uses PCA to compress the representation extracted from the closed-set backbones and uses the generated representation to detect the OOD pixels. We propose a change in the framework replacing the PCA used by OpenPCS with the Gaussian Mixture of Models (GMM) (Rasmussen, 2003). GMM's multimodal representation should be better suited for real-world pixels modeling that may not conform to the unimodal representation produced by PCA.

Inspired by the class conditioned auto-encoder for open-set recognition (C2AE) method proposed by Oza and Patel (2019), we propose our second approach called Conditional Reconstruction for Open-set Semantic Segmentation (CoReSeg) (Nunes et al., 2022b). CoReSeg is a fully convolutional end-to-end method for OSS that uses two CNNs: one for traditional closed-set segmentation and the other for conditional image reconstruction.

The intuition behind CoReSeg is that the network learns rightfully to reconstruct a pixel conditioned to its class. The higher the reconstruction error, the higher the chance the class is unknown. The pixel is set as OOD if the minimum reconstruction error among all class-conditioned reconstructions is higher than a chosen threshold.

The first two proposed methods improved baseline quantitative results and semantic consistency compared to baseline methods. However, as the baseline methods, both proposed methods produced open-set segmentations that erroneously classify pixels within larger, well-defined objects. As expected, naturally low-confidence regions, such as the edges of objects, are more commonly missegmented.

The third proposed approach is a superpixel post-processing strategy to mitigate the lack of semantic consistency. The final segmentation quality relies on selecting adequate hyperparameters for the superpixel segmentation algorithm. The superpixel post-processing can improve the results, but a wrong hyperparameter setting can lead to a worse post-processed result. We also propose a novel superpixel generation procedure called Fusing Superpixels for Semantic Consistency (FuSC) that makes post-processing hyperparameter selection more reliable and robust while still improving results.

1.4 Contributions

This thesis presents five contributions: an OSS method called OpenGMM, a novel end-to-end fully-convolutional method for OSS called CoReSeg; a general superpixel post-processing technique; a novel superpixel generation algorithm called FuSC; and a systematic mapping of the literature for open-set segmentation with the proposal of taxonomy to organize the related literature. The publications derived from this work are listed in Appendix C.

The two OSS methods improved baseline quantitative results and semantic consistency: OpenGMM is a modification of the framework OpenPCS (Oliveira et al., 2021) replacing the used PCA by the GMM to represent the compressed feature space, and CoReSeg is a novel end-to-end fully convolutional method (Nunes et al., 2022b).

The general superpixel post-processing strategy improved the quantitative results and the semantic consistency of all tested OSS. Post-processing with FuSC as the superpixel generation algorithm improved the robustness of hyperparameter selection while producing better, more stable, and reliable results.

At last, we propose a taxonomy to organize and assist in better understanding the existing articles and trends in deep open-set segmentation (Nunes et al., 2022a).

1.5 Document Organization

The remaining of this document is organized as follows: Chapter 2 presents the base theory supporting this work; Chapter 3 presents a systematic mapping of the OSS literature proposing a taxonomy to understand and organize the work in the field detailing the methods most related to our proposals; Chapter 4 describes the OSS methods and post-processing strategies proposals; Chapter 5 describes the experimental setup used to evaluate our proposal; Chapter 6 presents an ablation study to define the best hyperparameters to use for all final tests with brief discussion; Chapter 7 discuss and present the achieved results; and Chapter 8 summarizes the main contributions of this work and outlines directions for future work.

2
Theoretical Background

2.1
Convolutional Neural Networks

Convolutional neural networks (CNNs) proposed by LeCun et al. (1989) is a machine learning method to image recognition. CNNs became the dominant method for visual tasks in the last decade since Krizhevsky et al. (2012) proposed AlexNet for learning feature representations and improved CNNs scalability. AlexNet uses larger convolutional kernels and has eight convolutional layers before the final fully connected layer. Alexnet was a deeper neural network than the first proposed CNNs. The Very Deep Convolutional Networks (VGG) proposed by Simonyan and Zisserman (2014) increased the depth of the network due to the use of smaller convolutional kernels (size 3, stride 2, and padding equal to 1). VGG obtained better results using smaller kernels with networks of 16-19 layers in-depth than with shallower networks with larger convolutional kernels.

Training deep neural networks is more complex and computationally expensive in deeper networks. Networks with more than 20 layers deep suffer from the problem of vanishing gradients, when the backpropagated gradients do not reach the first layers of the network, resulting in a loss of performance. Residual Networks (ResNet) were proposed by He et al. (2016) and used residual learning to tackle the degradation caused by the vanishing gradients. The residual learning added shortcut connections (skip connections) that simply perform the summation of the identity mapping of input to the output of the stacked layers. The skip connections allow the gradient to backpropagate properly not needing extra parameters or adding computational complexity. Residual networks with more than 152 layers showed little benefit or even degraded performance. WideResNet (WRN) proposed by Zagoruyko and Komodakis (2016) is the most notorious architecture derived from the standard ResNet and yielded relevant improvements. WRNs are lower in depth but wider, the architecture increases the original residual block channels.

Densely Connected Convolutional Networks (DenseNet) Huang et al. (2017) which uses as input for each layer the concatenation of the output feature maps of all previous layers. The benefits of this strategy are: better feature propagation, handles the problem of vanishing gradients better, encourages feature reuse, and substantial reduces the number of needed parameters.

The state of art of CNNs architectures is not the scope of this work, but many other architectures were proposed and it is worth mentioning: EfficientNet Tan and Le (2019), EdgeNet Pradeep et al. (2018), Squeeze-Excitation Roy et al. (2018), MobileNet Howard et al. (2017), DiceNet Mehta et al. (2020) and HRNet Wang et al. (2020).

2.2 Semantic Segmentation

Semantic segmentation, also known as pixel-wise classification or dense labeling, is the task of clustering neighboring pixels in images that belong to the same semantic class as in Figure 2.1. Deep semantic segmentation (SS) is when a DNN is used to learn and predict semantic segmentation. Many CNN architectures were adapted to perform semantic segmentation after Alexnet made DNN the epicenter of computer vision research. In the last decade, DNN presented the best results and prevailed as the most used technique used to semantic segmentation.

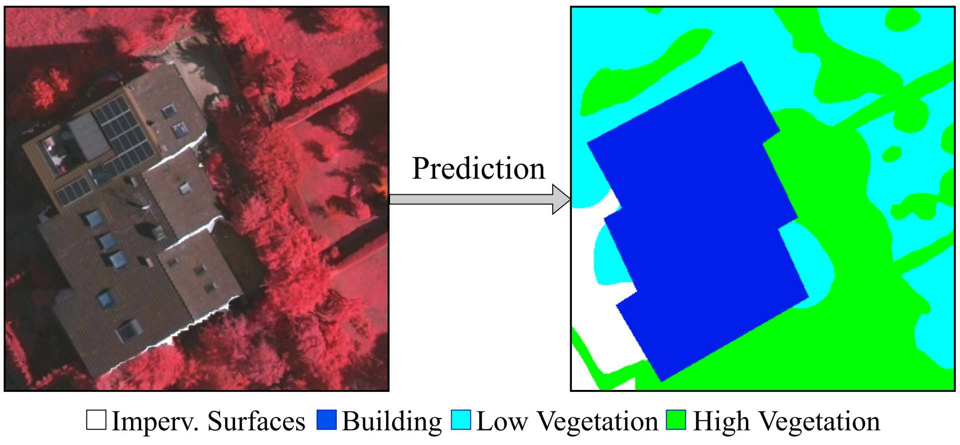


Figure 2.1: An example showing an urban image and its closed-set semantic segmentation.

2.2.1 Fully Convolutional Networks

CNN architectures like AlexNet (Krizhevsky et al., 2012), VGG Simonyan et al. (2013), ResNet He et al. (2016), DenseNet Huang et al. (2017) can be adapted to Fully Convolutional Networks (FCNs) to perform dense prediction as shown in Figure 2.2. Fully convolutional networks compute nonlinear image filters, while a general DNN computes general nonlinear functions. The end-to-end dense learning is achieved by replacing the CNN's fully connected layers with convolutional layers and adding a spatial loss Long et al. (2015). FCN training is equivalent to patchwise training, where each batch consists of a patch for each pixel in each image in a set of images Long et al. (2015).

A crucial part of training DNNs is the use of an adequate loss function allowing the model to learn correctly the desired task. According to Garcia-Garcia et al. (2018), Cross Entropy loss is one of the most used losses for image semantic segmentation and can be expressed by the Equation 2-1 below:

$$\mathcal{L}(Y, \hat{y}) = -Y \log(\hat{y}) - (1 - Y) \log(1 - \hat{y}), \tag{2-1}$$

where Y represents the pixel-wise semantic map and $\hat{y}$ the probabilities for each class for a given sample. The Jadon (2020) survey can be consulted for more information on losses used for semantic segmentation.

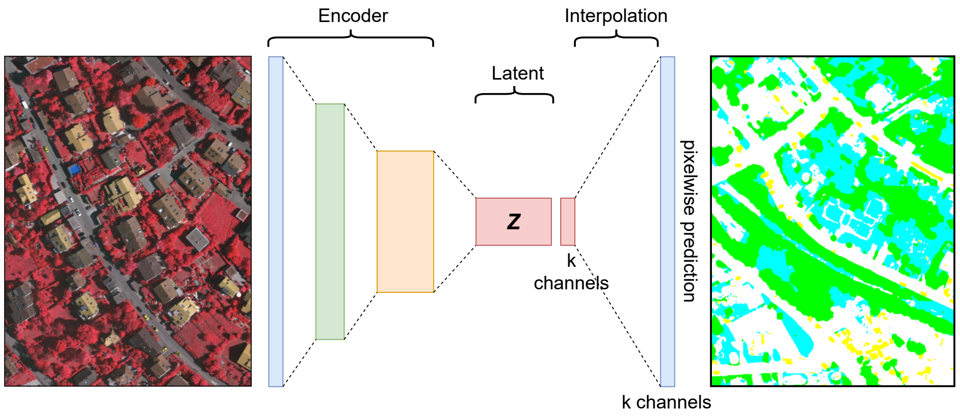


Figure 2.2: FCN is efficient in learning dense tasks like semantic segmentation. The encoder receives the input and reduces the spatial dimensions increasing the semantic dimensions with a higher number of channels in each layer, the final layer of an Encoder is called the latent representation (z). To perform semantic segmentation, the FCN compact the z layer into a k channel space layer, with k equal to the number of known semantic classes, then interpolated to the original input size. Figure based on Long et al. (2015).

The vanishing or exploding gradient problem (VEGP) is well documented and a fundamental obstacle in training neural networks, especially for deep neural networks that use gradient-based optimization techniques. The VEGP occurs when the derivative of the loss in the Stochastic Gradient Descent (SGD) update is very large or very small for a set of trainable parameters according to Hanin (2018). VEGP occurs if the network weights are too small to be relevant or too high to be precise.

While exploding gradients, in general, can be avoided using a small learning rate, a standard loss, scaling the target variables, and using normalization to configure the network, the vanishing gradient is trickier and needs some extra structures to handle it.

Skip connections provide alternative paths allowing the backpropagation of the gradients to mitigate the effects of the vanishing gradient problem. Skip connections alter the flow of gradients, allowing backpropagated gradients to reach the previous layers of the network without being processed by them Drozdzal et al. (2016).

Besides that, skip connections share features from the contracting path with the expanding path of a network to recover spatial/low-level pixel information lost during the down-sampling process Drozdzal et al. (2016). Skip connections also helps deal with diminishing feature reuse problems, enforcing learning of distinct features in distinct parts of the network according to Zagoruyko and Komodakis (2016).

2.2.2 Encoder-Decoder Architectures

An Encoder-decoder (EA) network can be defined as a symmetric network divided into 2 main parts: the Encoder receives the input and, while reducing the spatial dimensions, increases the semantic dimensions with a greater number of channels in each layer of reduced sampling; the Decoder reverses the processing made by the encoder, increasing the spatial dimensions, and decreasing the number of channels for each up-sampled layer. The symmetrical design allows the direct use of skip connections between the encoder and the decoder counterparts using the symmetrical feature vectors.

The Encoder side of the network uses standard convolutions to learn kernels to process the image and to down-sample the input channels. The symmetric Decoder side, in general, uses a transpose convolution to perform the inverse operation of the Encoder and learn kernels to upsample the input channels. The transpose convolution is a key development since the original FCNs used the non-learnable bilinear interpolation to upsample and recover the original spatial resolution of the image.

The U-net architecture proposed by Ronneberger et al. (2015) and shown in Figure 2.3 is an Encoder-Decoder network. U-net uses the skip connections to send to the decoder feature maps that allow the model to map higher-level contextual/semantic information to spatial/lower-level pixel information.

Using a very similar architecture, SegNet proposed by Badrinarayanan et al. (2017) usually uses the same 13 convolutional layers as VGG-16 Simonyan and Zisserman (2014) as the encoder with a mirrored decoder. The main difference to U-net is that instead of the feature maps, SegNet uses the

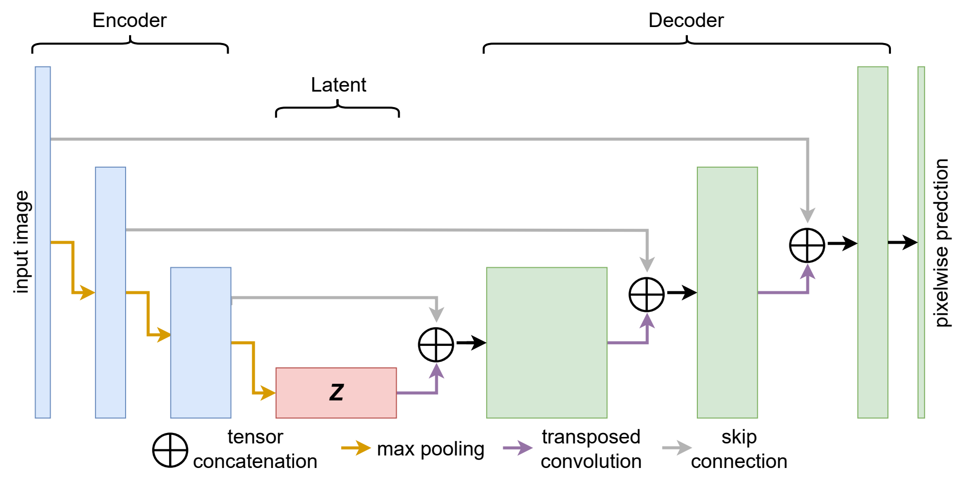


Figure 2.3: The figure is based on the proposed U-net architecture by Ronneberger et al. (2015). ⊕ represents the concatenation of copied feature maps with the output of the transposed convolution of the last network layer. The blue and green blocks are convolutional blocks of the Encoder and Decoder, respectively. The width of the blue and green boxes represents the number of channels, and the height represents the spatial dimension of the layer. Bigger green boxes represent a greater number of input channels.

pooling indices and the activations of the indices as skip connections. Also, an advantage of using the VGG-16 encoder is that it allows the use of pre-trained weights for the desired task.

2.2.3 Semantic Consistency in Segmentation

Semantic consistency is rarely explicitly addressed in semantic segmentation papers. In the following lines, we present an overview of the few existing trends in deep semantic consistency.

Through an end-to-end trainable network that combines 2 branches, one for edge detection and one for traditional semantic segmentation, Ji et al. (2020) managed to improve the performance and the spatial consistency of the resulting segmentation for PASCAL VOC 2012, PASCAL-Context and Cityscapes datasets.

PixMatch, proposed by Melas-Kyriazi and Manrai (2021), uses heavy augmentation and a loss term composed by the summation of two cross-entropy terms. The first loss term is standard for SSeg, and the second is calculated over a slightly perturbed image and mask. The new loss enforces the notion of smoothness in the target domain to enhance intra-object segmentation consistency.

Pixel-wise Contrast and Consistency Learning (PiCoCo), proposed by Kang et al. (2021), seeks consistency in closed-set semantic segmentation using a joint loss function that is the summation of a supervised loss term, a contrast loss term, and a consistency loss term. The supervised loss is a standard semantic segmentation loss composed of cross-entropy and dice loss terms. The contrastive loss uses a selection of positive and negative samples to enforce the model to improve its generalization capabilities. The consistency loss consists of a summation of cross-entropy and a dice loss of heavily augmented pairs of input and labels to enforce semantic consistency and robustness to the learning process.

Ratajczak et al. (2020) proposed a post-processing that combines unsupervised colorization and deep edge superpixels (DES) to enhance the semantic segmentation of panchromatic aerial images. The authors propose to assess if applying a colorization algorithm could improve the strength of the pairwise potentials used in a conditional random field (CRF) post-processing, this work defined DES using the Watershed Hu et al. (2015) with the intermediate activation maps obtained before each pooling layer to the output space of a Holistically-Nested Edge Detection Network Xie and Tu (2015), they use the generated superpixel segmentation (SPS) with the mean value for intensity together with CRF to improve the final semantic consistency.

The use of supervoxels to improve segmentation consistency was used by Zhang et al. (2014). A 3D-CNN was used to learn discriminative hierarchical features from spatiotemporal volumes.

Our work introduced post-processing for OSS that uses a superpixel segmentation algorithm to improve the semantic consistency of the resultant open-set prediction. Post-processing the open-set segmentations produced better results in all tested scenarios. We also proposed a new superpixel segmentation generation algorithm called FuSC. FuSC benefits from merging different input segmentations to produce a final one with better results in most tested scenarios compared to the same post-processing using the single algorithm SPS.

2.3 Auto-Encoder

Auto-encoders are also encoder-decoder networks, the goal of an autoencoder network is to learn a good representation of the input. Figure 2.4 shows a general symmetric structure for an auto-encoder model. Auto-encoders can be trained in a completely unsupervised fashion, learning data representations that can be used to perform different tasks. Like PCA, auto-encoders are very powerful in producing data representations in distinct dimensionality of the input. The main difference between PCA and auto-encoders is the ability to handle nonlinearities. An auto-encoder produces similar results to PCA if there is no non-linear activation in the model.

Figure 2.4 shows a general schematic for an auto-encoder where the input is compressed to latent layer Z and then decompressed to its original size. While each auto-encoder layer reduces the input size by a factor, the number of feature maps increases. Usually, the leaned latent representation Z layer have more channels/feature than the previous layers.

Auto-encoder models can learn to reconstruct the input data. The encoder learns a compressed latent feature representation Z, and the decoder reconstructs the input from Z Cheng et al. (2020). Auto-encoders typically contract spatial dimensions while increasing the number of channels in return. Auto-encoders usually minimize the L1, L2, or cross-entropy loss functions to reconstruct data.

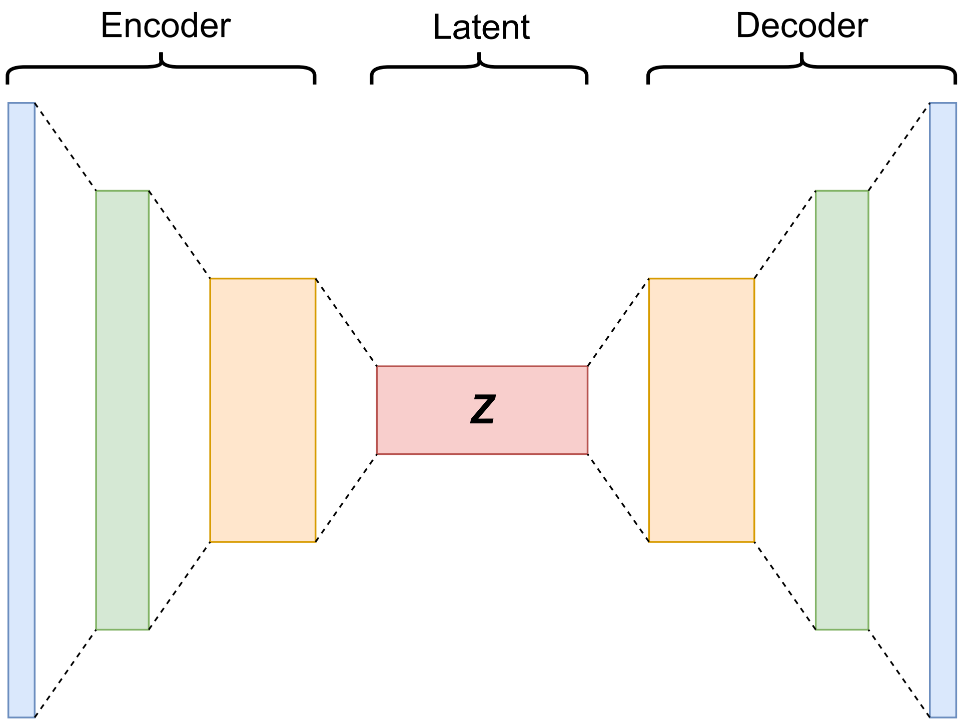


Figure 2.4: The figure shows a general schematic for an auto-encoder. The encoder compresses the input data to the latent Z layer. The latent layer is then up-scaled/decompressed to its original size.

Besides reconstruction, auto-encoders can: learn the denoising of an image Gondara (2016); generate a sparse representation of the input Ng et al. (2011); Makhzani and Frey (2013); Sun et al. (2018); for learning probability distributions with variational auto-encoders Kingma and Welling (2019); do image synthesis (Huang et al., 2018).

2.4 Attention

The notion of attention used in machine learning extends the meaning of the word attention as “a condition of readiness for such attention involving especially a selective narrowing or focusing of consciousness and receptivity” (Merriam-Webster, 2022). In this sense, the term “attention” should be understood as the notion of a non-uniform spatial distribution of the representation of relevant features for a specific task, together with the scalar representation of their relative relevance as stated by Jetley et al. (2018).

More specifically, for this work, attention should be understood as a trainable global (soft) attention mechanism (de Santana Correia and Colombini, 2022). The usage of attention mechanisms allows the network to consider the entire input and not only local sliding windows like CNNs or local temporal inputs like in RNNs.

The soft attention mechanism assigns values between 0 and 1 to each input element. The mechanism grades the focus on each tensor element considering the global interdependence between the input and the target. The use of sigmoid or softmax makes the entire attention mechanism deterministic and differentiable. Soft attention mechanisms can be spatial or channel-wise. Spatial attention increases the weights to focus on the most relevant areas of the image. The channel attention mechanism increases the weights of the most relevant feature maps (channels).

During the optimization process, the model learns where to focus and prioritizes these locations over others. Attention is a set of techniques that help a model to perceive relevant characteristics during the training of a DNN Jetley et al. (2018). Attention takes into account the entire feature space: the two-dimensional array for spatial and all channels for channel attention.

According to Chaudhari et al. (2021), attention became a determinant tool to obtain better results for multiple natural language processing (NLP) tasks. Using attention mechanisms improves DNN’s interpretability since it identifies the most relevant parts of any input. Attention mechanisms also help RNNs to overcome performance degradation with the increase in the length of input and the computational inefficiencies from the sequential processing.

2.4.1 Convolutional Block Attention Module

Convolutional block attention module (CBAM) proposed by Woo et al. (2018) is a simple and effective attention module for CNNs. CBAM is a general module and can integrate into any CNN architecture. CBAM sequentially learns attention maps using a spatial attention mechanism (SAM) and channel attention mechanism (CAM). Figure 2.7 shows an overview of the attention module, showing that CAM and SAM modules are serially applied.

$$\begin{aligned} F_1 &= CAM(F) \otimes F \\ F_{final} &= SAM(F_1) \otimes F_1 \end{aligned} \tag{2-2}$$

Equation 2-2 summarizes the functioning of CBAM presented in Figure 2.7, the multiplication of input tensor by the attention modules outputs broadcast the attention results. In the equation, the symbol $\times$ denotes element-wise multiplication, the $F \in \mathbb{R}^{C\times H\times W}$ denotes the input feature map, CAM and SAM the attention modules as presented in Figures 2.5 and 2.6. CAM learns a 1D channel attention map with the output $\in \mathbb{R}^{C\times 1\times 1}$, and SAM learns a 2D spatial attention map with the output $\in \mathbb{R}^{1\times H\times W}$.

Since each channel on a feature map can be seen as a feature detector, the CAM module enhances the relevance of the most meaningful features of the feature map. To perform the attention operation, CAM squeezes the input feature map $\in \mathbb{R}^{C\times H\times W}$ to $\in \mathbb{R}^{C\times 1\times 1}$. The squeezed feature map is used as input for a two-layer multilayer perceptron (MLP) to learn which channels are more relevant to the target. CAM uses the sum of average pooling and max pooling as squeeze operations to compute the output. Equation 2-3 summarizes the functioning of CAM, where σ denotes the sigmoid function.

$$CAM = \sigma(MLP(AvgPool(F)) + MLP(MaxPool(F))) \tag{2-3}$$

While CAM learns which features are more relevant, SAM is the spatial attention mechanism and learns to focus on the most relevant areas of the input tensor. SAM uses average and max pool operations to squeeze the input feature map along the channel axis generating two 2D maps. The 2D maps are concatenated producing a tensor $\in \mathbb{R}^{2\times H\times W}$ that are submitted to a convolution to compute the final attention map $\in \mathbb{R}^{1\times H\times W}$. The Equation 2-4 shows the operations of SAM where f^{7x7} denotes a convolution operation with a kernel of size 7.

$$SAM = \sigma(f^{7x7}([AvgPool(F)); MLP(MaxPool(F)])) \tag{2-4}$$

2.5 Conditioning Deep Neural Networks

Conditioning in machine learning often refers to context-based processing in the sense that some input is processed in the context of another piece of information. The conditioning mechanism allows processing an image in the context of a question to extract some information from the image and

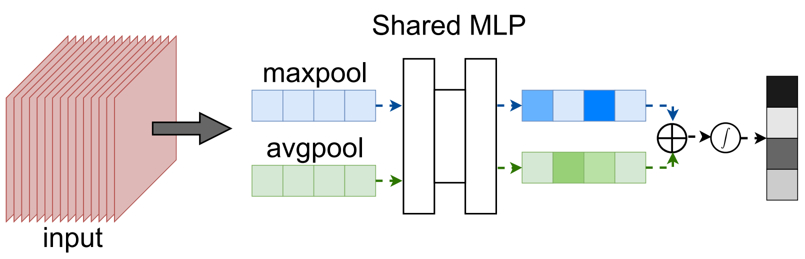


Figure 2.5: The figure depicts the workflow of the channel attention sub-module. The ⊕ symbol represents the concatenation operation and the ∫ represents the sigmoid operation.

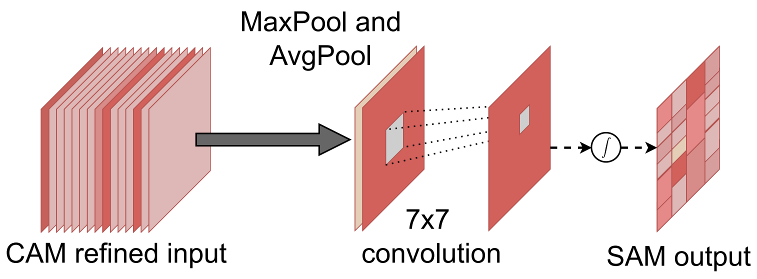


Figure 2.6: The figure depicts how spatial attention works. The ∫ represents the sigmoid operation.

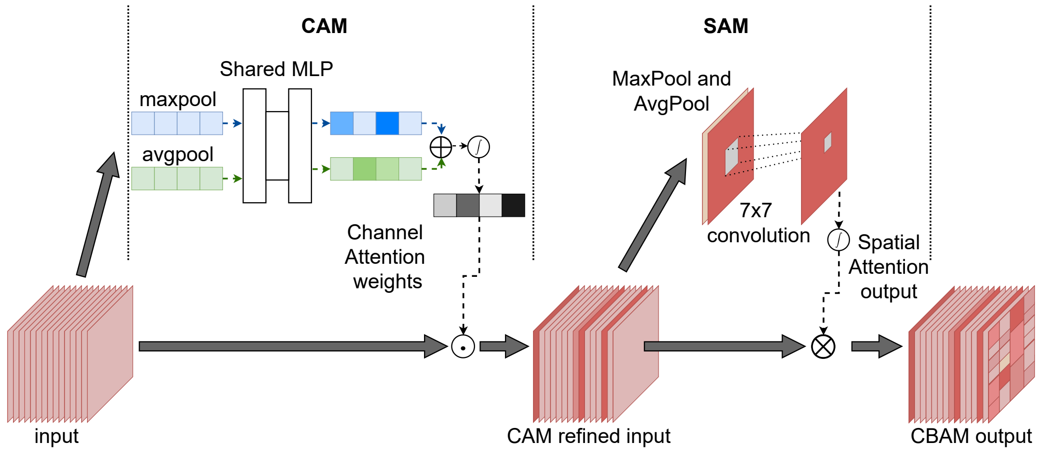


Figure 2.7: The figure shows an overview of CBAM with the refinement of the input made by the attention sub-modules. The ⊕ symbol represents the concatenation operation, ∫ represents the sigmoid operation, ⊗ represents the pixel-wise multiplication, and the ⊙ symbol stands for a vector-tensor multiplication.

Figure 2.8: Figure 2.7 is shown an overview of CBAM with the two sub-modules showing how the input feature map is adaptively refined through the module Woo et al. (2018). Figure 2.5 shows the sub-module for channel attention and Figure 2.6 shows the sub-module that handles spatial attention.

answer a question (Dumoulin et al., 2018). Another example will be given in this work using an Encoder-Decoder network to learn how to reconstruct images conditioned to their own semantic segmentation masks imposing that the model learns better representations for each known class present in the ground truth.

By conditioning a neural network, we adopt the notion of task representation. This notion of task representation allows changing the behavior of the model due to the external information coded as the condition input according to Dumoulin et al. (2018).

According to Dumoulin et al. (2018) there are three different ways of conditioning a model:

1. concatenation-based - concatenate the condition with the input and force the network to carry the conditioning information until it is needed. The concatenation approach is parameter efficient since the conditioning is only passed with the input;

2. biasing or additive - maps the condition tensor to a bias tensor and adds the bias tensor to hidden layers. Parameter efficient and lightweight;

3. scaling or multiplicative - similar to biasing, the condition tensor maps to a scaling tensor used to scale hidden layers with a dot product (multiplication);

Biasing conditioning can be seen as a different implementation of concatenation-based conditioning since we can decompose the concatenation based on operations equivalent to the biasing conditioning. Both biasing and scaling conditioning have interesting characteristics, scaling or multiplicative interactions can learn relations among inputs, and the dot product allows to amplify or identify similar inputs. The biasing or additive conditioning is more appropriate to applications that are less dependent on both inputs simultaneously (Dumoulin et al., 2018).

Combining biasing and scaling conditioning has emerged as a more suitable option to take advantage of the characteristics of both conditioning strategies. Perez et al. (2018) proposed the Feature-wise Linear Modulation (FiLM) by defining an affine operation (Equation 2-5) combining scaling and biasing conditioning.

The FiLM is a general-purpose conditioning method for neural networks. FiLM layers are highly effective for visual reasoning, being capable of answering image-related questions that require a high-level process. The FiLM can be seen as a generalization of conditional normalization methods, replacing the parameters of the feature-wise affine transformation with a learned function derived from some conditioning input.

Figure 2.9 shows the affine transformation expressed by Equation 2-5 applied to the network's intermediate features. The applied transformation guide FiLM to learn how to highlight or suppress feature maps based on conditioning information.

$$FiLM(F_{i,c} \mid \gamma_{i,c},\ \beta_{i,c}) = \gamma_{i,c}F_{i,c} + \beta_{i,c} \tag{2-5}$$

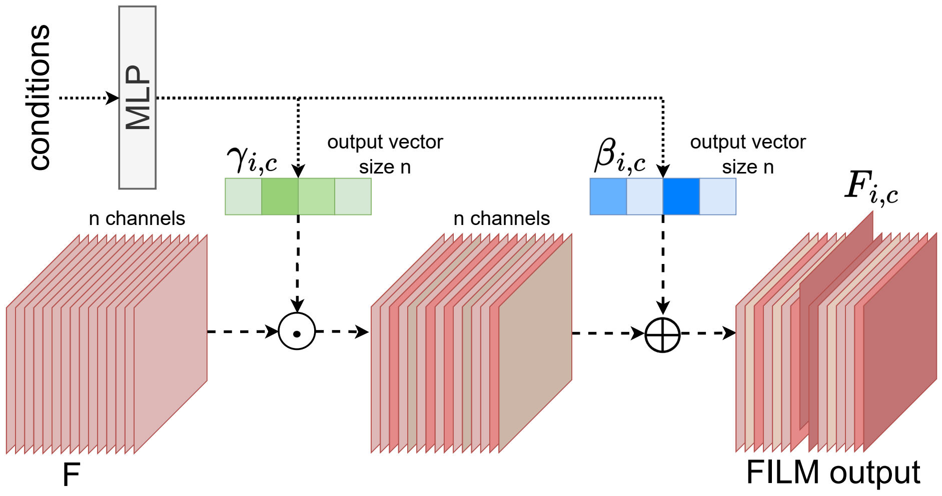


Figure 2.9: The figure shows the functioning of the FiLM layer applied to a CNN. The $\odot$ symbol stands for the Hadamard product between the γ and the channels, and the $\oplus$ symbol stands for the summation of the β vector with the channels. The subscripts c and i stands for the c^{th} feature map of i^{th} input.

2.6 Superpixel Segmentation

For this work, we consider a superpixel as a group of contiguous pixels in a given image that is grouped according to some criterion of homogeneity. As spatiality is crucial to any SPS, neighboring superpixels should be perceptually different. Nevertheless, non-neighboring superpixels may have similar values and shapes. All pixels inside a superpixel should assume as value some representative measure like the mean or median value for each image band.

SPSs are an active research area, and many distinct methods were proposed to generate superpixels from an image. As examples of well-known methods proposed in the last two decades, we can cite: Felzenszwalb (Felzenszwalb and Huttenlocher, 2004), Quickshift (Vedaldi and Soatto, 2008), TurboPixels (Levinshtein et al., 2009), ERS (Liu et al., 2011), SLIC (Achanta et al., 2012), GSM (Morerio et al., 2014), Eikonal-based (Buyssens et al., 2014), SEEDS (Bergh et al., 2012), LSC (Li and Chen, 2015), Waterpixels (Machairas et al., 2015), BASS (Rubio et al., 2016), SAS (Achanta et al., 2018), SH+FDAG (Wang et al., 2019), content-based (Zhang et al., 2020) and SPFCM (Elkhateeb et al., 2021).

Among all possible choices of SPS algorithms to use in this work, we choose three algorithms that have fundamentally different strategies to generate the superpixels: SLIC (Achanta et al., 2012), Quickshift (Vedaldi and Soatto, 2008) and Felzenszwalb (Felzenszwalb and Huttenlocher, 2004). In the following paragraphs, we briefly present these three superpixel algorithms.

2.6.1 Simple Linear Iterative Clustering

The Simple linear iterative clustering (SLIC) algorithm proposed by Achanta et al. (2012) groups pixels into perceptually meaningful contiguous regions. The method adapts the K-means algorithm Lloyd (1982) to generate the superpixels. Figure 2.10 shows examples of SLIC algorithm results.

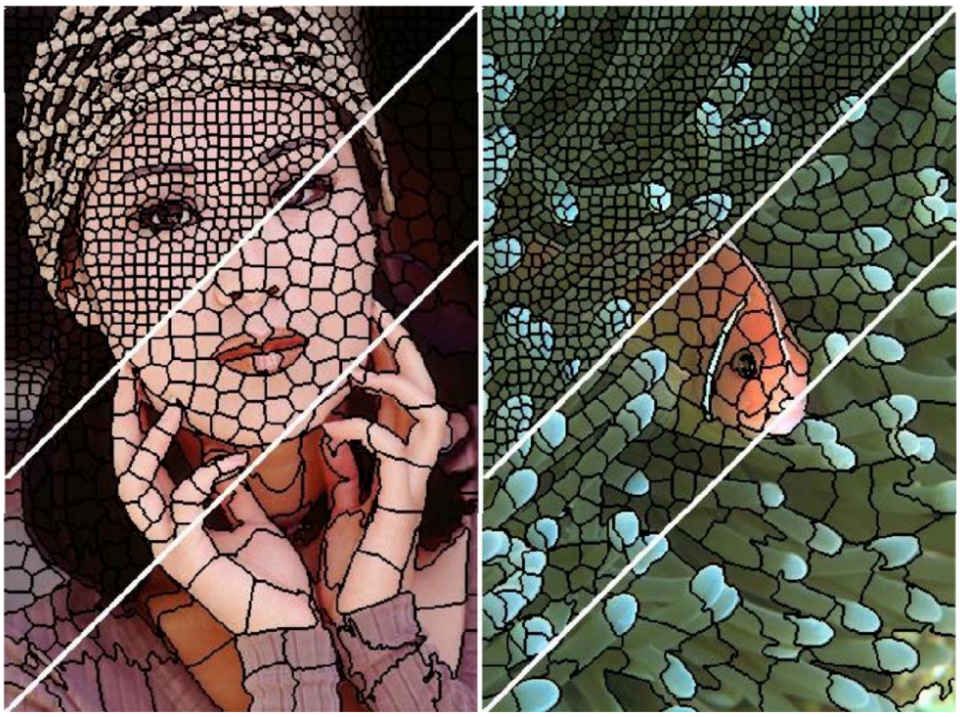


Figure 2.10: An example of 2 images segmented using the SLIC algorithm. Each presented image is segmented with three distinct approximate superpixel sizes: 64, 256, and 1024 pixels. Figure extracted from Achanta et al. (2012).

The operation of the algorithm is simple. It starts with n centers uniformly distributed in the image. Then, it adjusts the centers' positions to the local minimum of the gradient of pixel intensity to avoid centering the superpixel in an edge. Finally, it iterates to every center, reallocating the pixels in a fixed-size window to the closest center. Limiting the search area makes the algorithm faster while still producing good results. The SLIC algorithm produces areas with great homogeneity as it is built on a K-means-like approach, but it may neglect the natural edges of the image as a side effect.

2.6.2 Quickshift

Quickshift is a fast mode-seeking algorithm proposed by Vedaldi and Soatto (2008). In Quickshift, every pixel started as a superpixel then the closest ones are put together within a defined radius distance. For each superpixel, if a pixel is out of the radius limits, a new cluster is defined and populated by the pixels inside the radius of the new centroid. Quickshift is a hierarchical clustering algorithm, and the size of the clusters can be derived from the generated tree of radius values. This method does not force the pixels to be close to each other spatially, producing highly homogeneous superpixels of different sizes and shapes. Figure 2.11 shows an example of an image segmented using the QuickShift algorithm.

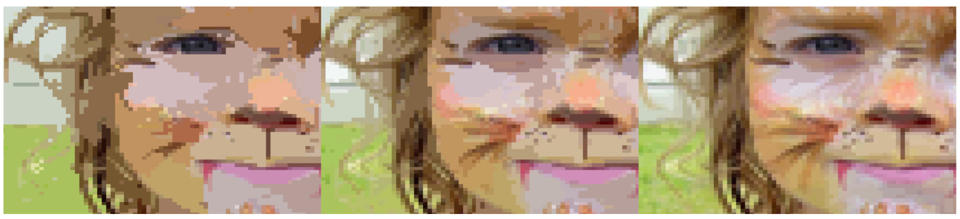


Figure 2.11: An example of an image segmented using the QuickShift algorithm. The image is shown segmented with three distinct superpixel configurations. Figure adapted from Vedaldi and Soatto (2008).

2.6.3 Felzenszwalb

Proposed by Felzenszwalb and Huttenlocher (2004), the algorithm is a graph-based segmentation algorithm where each vertex represents a pixel, and each selected edge has some measure of dissimilarity as its value. Every pixel in the image graph-represented, but only some edges are added to the graph according to a defined criterion (e.g. K-nearest neighbors) to guarantee the intended complexity for the algorithm ($O(m\ log\ n)$, where m is the number of edges and n the number of vertices).

There are two presented strategies to select the edges, the first one uses the notion of a grid and connects each pixel to the 8 closest ones in the grid. The second strategy maps the image into a higher-level feature space and

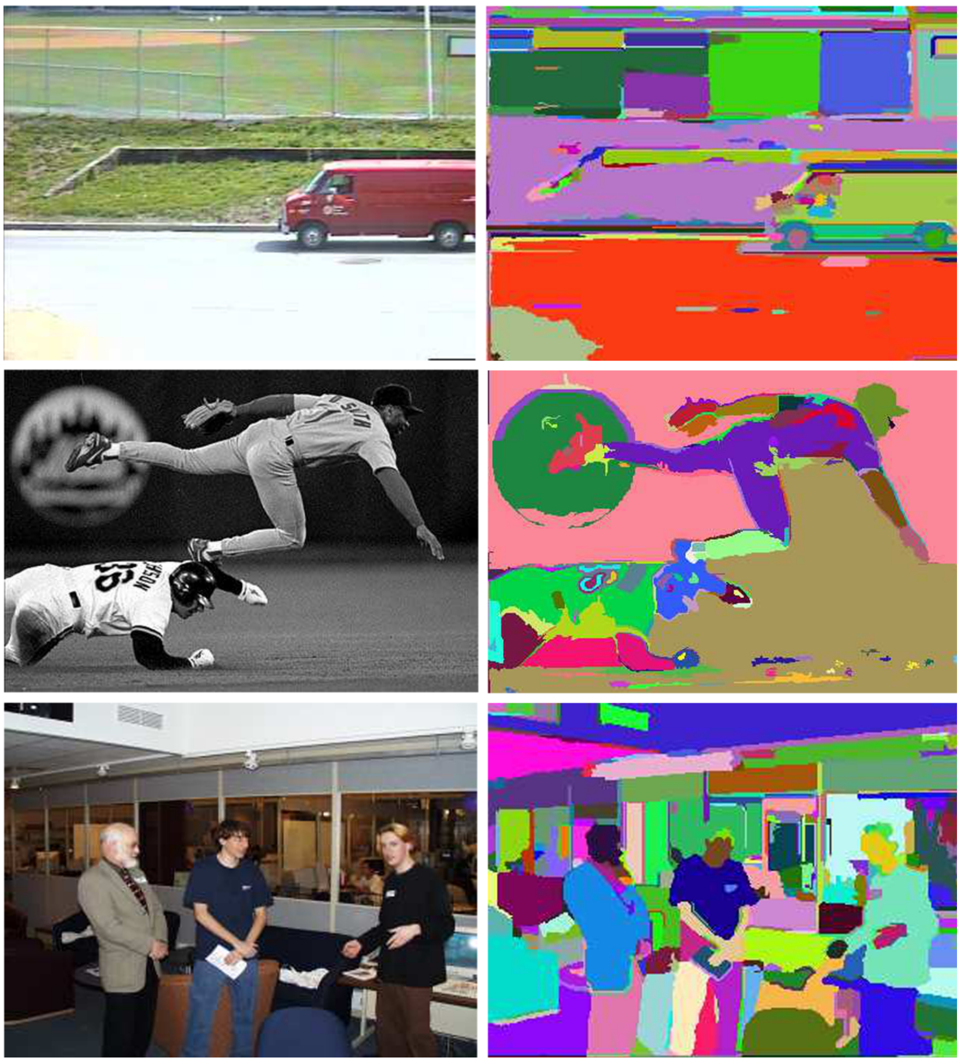


Figure 2.12: Three images segmented using Felzenszwalb algorithm. Figure adapted from Felzenszwalb and Huttenlocher (2004).

connects the m closest points in this new space. For instance, a 5-dimensional space is defined by the spatial position x and y, and the three colors of RGB.

The algorithm explores the same idea that Kruskal presented in Kruskal (1956) used in their classical algorithm to find the minimum spanning tree (MST) on a graph and selects the edges in a non-descending order to generate the clusters. The clusters are generated using the intuition that the intracluster dissimilarities are lower than the dissimilarities in the borders among clusters.

By construction, the Felzenszwalb algorithm generates clusters that vary in shape and size, but strongly respect the borders of the natural objects in the image.

2.7 Open-set Recognition

Traditionally, datasets and methods are designed to deal with a static closed world, knowing all possible categories during training. The same premise applies to most DNNs developed to handle data from closed-sets (Bendale and Boult, 2016).

Adapting recognition systems from controlled laboratory environments to the real world presents many operational challenges. Any recognition system exposed to real-world input needs to identify unseen categories while adding additional categories with little to no downtime. An open-world system has to identify new classes and add the newly identified classes to the learned multiclass open-set recognition algorithm (Bendale and Boult, 2015).

The use of the term "recognition" and not "classification" is the first question to be answered. Classifying something presupposes that all possible classes in the universe are known. Recognizing something is a broader concept, as it assumes that some classes are recognizable in a larger domain with unrecognizable elements Scheirer et al. (2012).

The concept of *openness* introduced by Scheirer et al. (2012) measures the knowledge available in training time from a test time perspective, explaining how effective a model could be. Equation 2-6 shows the original formulation, where "target classes" is the number of classes to be identified. This formulation yields the percentage of openness, where 0 represents a completely closed problem.

$$openness = 1 - \sqrt{\frac{2 \times \text{training classes}}{\text{testing classes} + \text{target classes}}} \tag{2-6}$$

To better understand the open universe Scheirer et al. (2014) defined three recognition categories and Geng et al. (2020) expand adding one more:

1. known known classes (KKCs) - correctly labeled classes with data available at training time;

2. known unknown classes (KUCs) - wrongly labeled classes or grouped meaningful classes (i.e. background)

3. unknown known classes (UKCs) - classes present in data at training time but not labeled;

4. unknown unknown classes (UUCs) - classes with no information on training time.

The survey presented by Geng et al. (2020) adopted a taxonomy for OSR methods. Methods fit into two main categories: Discriminative or Generative. Discriminative methods fit into two sub-categories, traditional machine learning-based and DNN-based. Generative methods fit into instance generation-based and non-instance generation-based. This work focus on DNN-based discriminative methods.

According to Geng et al. (2020), the threshold-based approach is the most common among discriminative OSR recognition models. In this type of approach, the end result of recognition is computed by an empirically established threshold that defines whether to be set as unknown or classified as one of the KKCs. The need for modeling distribution tails makes EVT widely used in OSR methods. The optimal definition of the size of the tail or the value of the threshold is still an open research question.

3
Related Work

Understanding and organizing the literature on any area is a challenging task that can help researchers to place their work among the many existing methods. It may also be useful to provide an overview of the research area for newcomers and the following works.

Considering the lack of a more structured organization for the OSR and OSS literature, in Section 3.1, we perform a systematic mapping of the literature and propose a taxonomy for deep learning open-set recognition and segmentation, helping to organize the literature by classifying existing methods according to their characteristics. Furthermore, the taxonomy allows the identification of the emerging trends that may serve as the base for future approaches and locate our proposal in the literature. To this extent, we focused on deep learning-based methods only.

Based on the result of the systematic mapping, we explore in more depth the articles that are most related to the developed methods in Section 3.2.

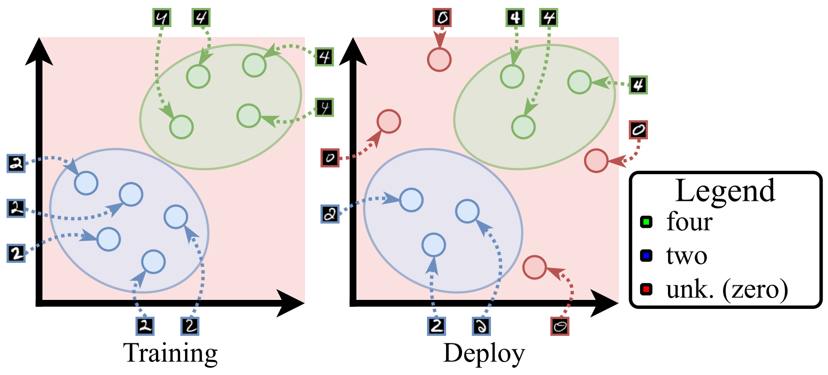
Section 3.3 concludes the chapter by discussing some of the most promising trends in OSR and OSS while also presenting possible future research directions in the field and its relation with the methods developed in this work.

3.1
A Systematic Mapping of Open-set Segmentation in Visual Learning

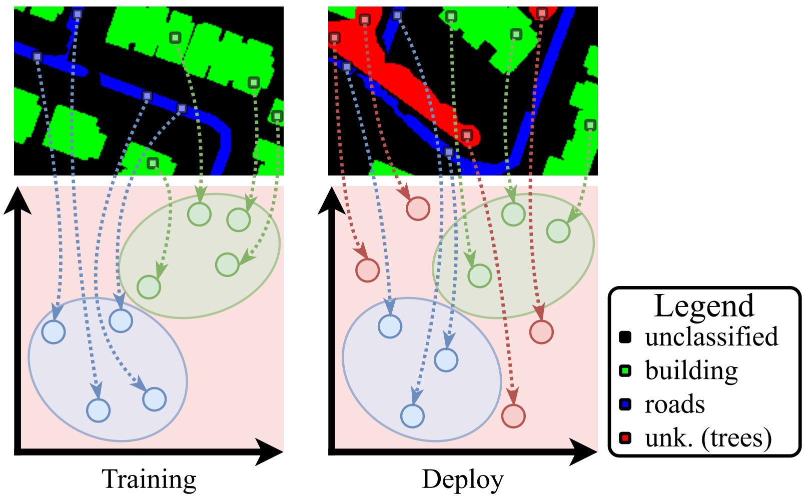
This section presents a systematic literature review and the proposal of novel taxonomy to organize OSR and OSS tasks together. To the extent of the author's knowledge, this is the first taxonomy proposed to tackle both tasks simultaneously.

Since most open-set segmentation works derive from a recognition method, a taxonomy for both tasks allows the identification of the most promising trends for segmentation and the ones not explored.

Section 3.1.1 presents OSR and OSS, and Section 3.1.2 introduces the literature reviewing process and the proposed used taxonomy. Section 3.1.4 discusses the most representative OSR papers according to the proposed taxonomy, while Section 3.1.5 reviews, analyzes, and categorizes the OSS articles.



3.1(a): OSR



3.1(b): OSS

Figure 3.1: Difference between training and deployment phases in OSR (a) and OSS (b) scenarios. Red circle samples (for OSR) or red pixels (for OSS) represent samples unknown in training.

3.1.1 Introduction

During the last decade, the automation of visual recognition tasks has reached human-level standards in many domains (Huang et al., 2017; Zagoruyko and Komodakis, 2016; Zhang et al., 2022; Tao et al., 2020). CNNs (Krizhevsky et al., 2012) shifted the main limitation of visual recognition from the lack of representation capability of shallow features to the amount of labeled training data in a dataset/domain. Closed-set tasks in CNNs and related network architectures such as classification, detection, or segmentation assume that the training and testing label spaces are the same (Sun et al., 2020). This scenario is not compatible with the majority of real-world problems since the tasks are limited due to the difficulty of collecting labeled samples that exhaust all possible classes.

As stated by Scheirer et al. (2012), an open-set scenario happens when unknown samples can appear in the prediction phase. In an open-set scenario, not all possible classes are known during training. Applying this definition to a classification problem, a new task called Open-set Recognition arises. The same definition can also be used for each image pixel, extending the traditional semantic segmentation problem to Open-set Segmentation. OSS refers to the set of algorithms that identifies pixels of unknown or out-of-distribution classes at inference time while correctly classifying pixels of known classes learned in training (Oliveira et al., 2021). Figure 3.1 illustrates the OSR and OSS tasks.

The open-set tasks have caught the research community's interest with multiple recently proposed methods for OSR problems (Bendale and Boult, 2016; Sun et al., 2020; Oza and Patel, 2019; Cui et al., 2020; Guo et al., 2021). However, only a few publications tackle the problem for different visual tasks, such as segmentation or object detection (Hendrycks et al., 2018). OSS is an inherently harder problem due to its dense labeling nature compared to Open-set Recognition or Classification. Thus, in real-world scenarios, it is harder to perform open-set semantic segmentation precisely (Brilhador et al., 2021). The complexity of the problem may explain why there is still a gap in the literature, with only a handful of articles tackling the issue (Cui et al., 2020).

3.1.2 Systematic Review Methodology

Aiming to systematize the choice and analysis of publications on OSS, we followed the methodology from the literature of systematic mapping (Kitchenham et al., 2009) as to how to conduct an organized review process.

We delimited this survey to focus on deep learning methods for OSS. The used search terms were: 1) "*segmentation*"; 2) "(*open-set* OR *open set* OR *openset* OR *open-world* OR *open world*)"; "(*deep learning* OR *neural network*)".

We selected three digital libraries/search engines to gather comprehensive results: Google Scholar[1], Scopus[2], and Web of Science[3]. We defined only one search string for Scopus and Web of Science since they allow for structured search strings, as presented in Table 3.1. Table 3.1 also shows the two less restrictive search strings defined for Google Scholar.

Besides the search results for OSS, multiple relevant OSR publications were manually included in the mapping, as the majority of OSS methods were adapted from the OSR literature. Section 3.1.4 presents an overview of the OSR

[1]`https://scholar.google.com.br/`
[2]`https://www.scopus.com/search/form.uri?display=basic#basic`
[3]`https://www.webofscience.com/wos/woscc/basic-search`

Database	Search	Results
Web of Science	"segmentation" AND ("open set" OR "open-set" OR "openset" OR "open world" OR "open-world") AND ("neural network" OR "deep learning")	16
Scopus	"segmentation" AND ("open set" OR "open-set" OR "openset" OR "open world" OR "open-world") AND ("neural network" OR "deep learning")	36
Google Scholar	open-set segmentation	33
Google Scholar	open-world segmentation	27

Table 3.1: The table presents the used queries for each search engine and the number of results returned.

literature according to the taxonomy presented in Section 3.1.3. We highlight that the review of the OSR literature is intended to be representative rather than extensive, considering that the number of papers on OSR is considerably larger than those on OSS. We aimed to reach the representativity by including the seminal articles for each category from the taxonomy with the ones found during the OSS paper search. Thus, the selection of articles for the OSR task is not necessarily fully complete, differently from the OSS review. Yet, the revision of OSR papers is necessary, as we describe OSR methods to introduce OSS afterward.

Since the total number of articles is relatively small, we considered the union of all results and manually excluded the following types of publications: surveys; thesis; dissertations; submitted and rejected articles; publications describing frameworks used in competitions; articles in which the main task is other than segmentation or recognition; and articles not focusing on images. We refined the search of OSS methods to 71 publications after duplicate removal, further reducing this number to 24 papers after applying the exclusion criteria, with 15 being focused on OSS and 9 dealing with OSR tasks. Figure 3.2 shows the distribution of the publications by year of publication and the growing interest in OSR and OSS from the research community.

As only 24 publications resulted from the combined search and exclusion criteria, all articles were read and further classified into the taxonomy. To better understand research trends in OSS using deep neural networks, we extracted the following complementary data from all final selected articles:

1. Does the article address the open-set scenario?
2. Which is the main task addressed by the article?
3. What kind of data is used?

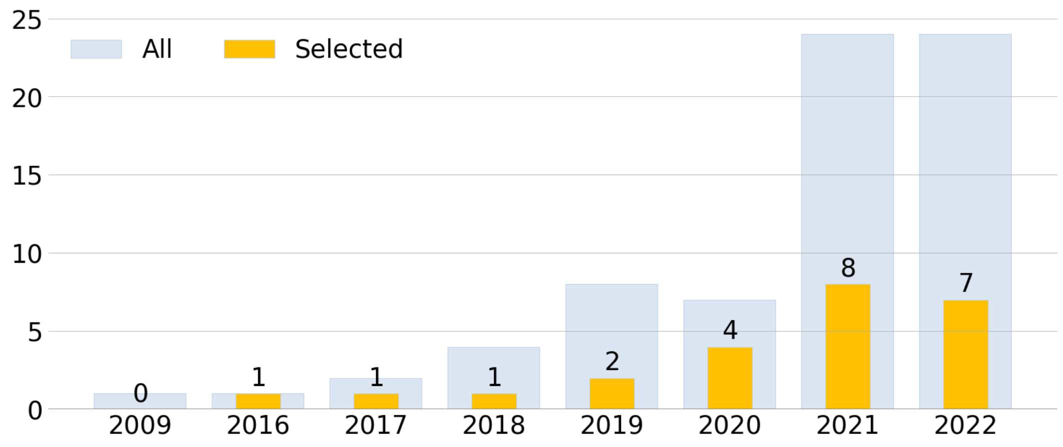


Figure 3.2: The evolution in the number of publications of the combined search results is shown in grey and in yellow is the final number of selected articles.

4. Does the method use reconstruction?
5. Does the method use auxiliary data?
6. Does the method use a generative approach?
7. Does the method use any statistical modeling?
8. Does the method use the intermediate feature space?
9. Can easily adapt the method from the closed-set task or, in short, is the method plug & play?
10. Does the method use extreme value theory (EVT) to model OOD classes?

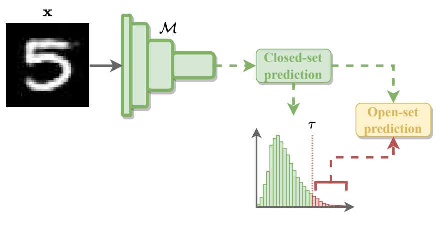
We compiled the Table 3.2 from the proposed questions above. Each column of the table answers one proposed question to map the architectural choices made by the authors. We used the proposed questions to map the emerging trends in literature, assisting in organizing the methods to define an adequate taxonomy.

We further detail the most relevant individual articles presented in Table 3.2 in Sections 3.1.4 and 3.1.5.

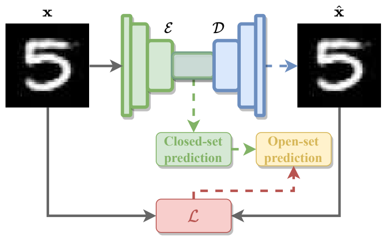
3.1.3 Taxonomy

Aiming to better understand the trends, the selection of articles guided us to the following taxonomy, mapping three identified paradigms that organize the families of methods for OSR and OSS commonly found in the literature:

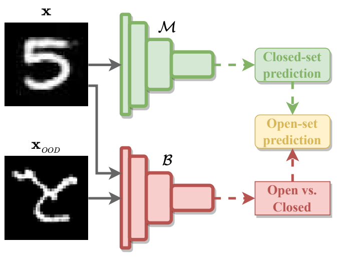
1. *Statistical modeling*: statistics of the intermediary and output activations from $\mathcal{M}$ models are used to define in- and out-of-distribution samples



3.3(a): Statistical Modeling



3.3(b): Reconstruction-based



3.3(c): Auxiliary data

Figure 3.3: Figures present the schematics for the proposed taxonomy: statistical modeling (a); reconstruction-based (b); auxiliary data (c). In all figures, x represents the input data, $\hat{x}$ the reconstructed input, $\mathcal{M}$ the closed-set model, τ the threshold used to identify the OOD pixels, $\mathcal{E}$ the encoder and $\mathcal{D}$ the decoder of the reconstruction auto-encoder, and *beta* a discriminator model.

(Bendale and Boult, 2016; Ge et al., 2017; Hendrycks et al., 2018; Sun et al., 2020; da Silva et al., 2020; Cui et al., 2020; Vendramini et al., 2021; Oliveira et al., 2021; Martinez et al., 2021; Yan et al., 2021; Cen et al., 2021; Grcić et al., 2021; Chan et al., 2021; Gawlikowski et al., 2022; Hong et al., 2022; Dong et al., 2022), as illustrated in Figure 3.3(a). This is a broader category than the next two, and as such, it is possible to further split it into four overlapping subdivisions according to the characteristics of the statistical modeling - which activation layers are used, the employment of EVT, the use of activations to represent known and unknown classes, and the output of an anomaly (entropy or probability) score;

2. *Reconstruction-based*: image reconstruction loss $\mathcal{L}$ is used to model or classify OOD samples (Yoshihashi et al., 2019; Oza and Patel, 2019; Sun et al., 2020; Nunes et al., 2022b), as shown in Figure 3.3(b). This category is split into two subdivisions - Conditional or not. The conditional subdivision is characterized by the employment of class conditioning as a means of reconstructing the input image according to the learned condition. Conditional strategy tends to generate worst reconstructions for the OOD classes due to unknown adequate conditioning;

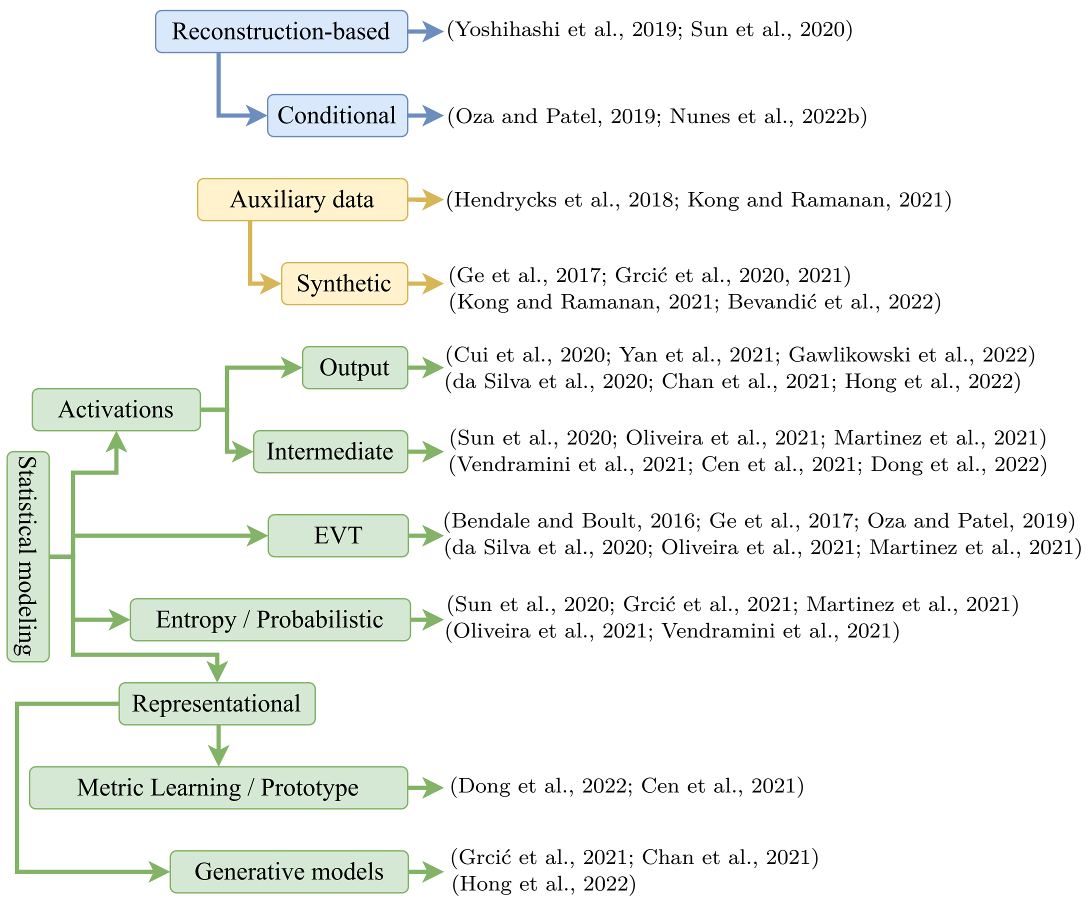

Reconstruction-based
(Yoshihashi et al., 2019; Sun et al., 2020)
Conditional
(Oza and Patel, 2019; Nunes et al., 2022b)
Auxiliary data
(Hendrycks et al., 2018; Kong and Ramanan, 2021)
Synthetic
(Ge et al., 2017; Grcić et al., 2020, 2021)
(Kong and Ramanan, 2021; Bevandić et al., 2022)
Statistical modeling
Activations
Output
(Cui et al., 2020; Yan et al., 2021; Gawlikowski et al., 2022)
(da Silva et al., 2020; Chan et al., 2021; Hong et al., 2022)
Intermediate
(Sun et al., 2020; Oliveira et al., 2021; Martinez et al., 2021)
(Vendramini et al., 2021; Cen et al., 2021; Dong et al., 2022)
EVT
(Bendale and Boult, 2016; Ge et al., 2017; Oza and Patel, 2019)
(da Silva et al., 2020; Oliveira et al., 2021; Martinez et al., 2021)
Entropy / Probabilistic
(Sun et al., 2020; Grcić et al., 2021; Martinez et al., 2021)
(Oliveira et al., 2021; Vendramini et al., 2021)
Representational
Metric Learning / Prototype
(Dong et al., 2022; Cen et al., 2021)
Generative models
(Grcić et al., 2021; Chan et al., 2021)
(Hong et al., 2022)

Figure 3.4: Classification of the selected publications under the proposed categories of the taxonomy presented in Section 3.1.2. Each category can be further divided into more refined groups according to the methods' characteristics. Each method may fall under more than one group, as they are not mutually exclusive.

3. *Auxiliary data*: a discriminative model β trained with KKC and KUC samples can discriminate between known and unknown samples, (Hendrycks et al., 2018; Grcić et al., 2020, 2021; Kong and Ramanan, 2021; Bevandić et al., 2022; Grcić et al., 2022). The pipeline in Figure 3.3(c) shows a closed-set model $\mathcal{M}$ coupled with a discriminative model $\mathcal{B}$ used to identify UUC samples. This category can be split into two subdivisions - Synthetic or not. The Synthetic methods use some type of generative strategy to generate OOD samples, helping to better model in and out-of-distribution samples.

Graphical visualization of all selected papers under the respective category is shown in Figure 3.4.

Ref	T	D	R	A	G	S	F	P	E	SE
Bendale and Boult (2016)	R	I	-	-	-	✓	-	✓	✓	M
Ge et al. (2017)	R	I	-	-	✓	✓	-	-	✓	M
Hendrycks et al. (2018)	R	I	-	✓	-	✓	-	-	-	M
Yoshihashi et al. (2019)	R	I	✓	-	✓	-	-	-	-	M
Oza and Patel (2019)	R	I	✓	-	✓	-	-	-	✓	M
Sun et al. (2020)	R	I	✓	-	✓	✓	✓	-	-	M
Vendramini et al. (2021)	R	I	-	-	✓	✓	✓	✓	-	M
Bharadwaj et al. (2022)	R	I	-	-	-	-	-	-	-	W
Gawlikowski et al. (2022)	R	RS	-	-	-	✓	-	-	-	W
da Silva et al. (2020)	S	RS	-	-	-	✓	-	✓	✓	S
Grcić et al. (2020)	S	I	-	✓	✓	-	-	-	-	G
Cui et al. (2020)	S	I	-	-	-	✓	-	-	-	S,W,G
Oliveira et al. (2021)	S	RS	-	-	✓	✓	✓	✓	✓	S,G
Martinez et al. (2021)	S	RS	-	-	✓	✓	✓	✓	✓	S,G
Yan et al. (2021)	S	I	-	-	-	✓	-	-	-	S
Cen et al. (2021)	S	I	-	-	-	✓	✓	-	-	S,G
Grcić et al. (2021)	S	I	-	✓	✓	✓	-	-	-	G
Chan et al. (2021)	S	I	-	-	-	✓	-	-	-	W
Kong and Ramanan (2021)	S	I	-	✓	✓	-	✓	-	-	G
Nunes et al. (2022b)	S	RS	✓	-	✓	-	-	-	-	G
Bevandić et al. (2022)	S	I	-	✓	-	-	✓	-	-	S,G
Grcić et al. (2022)	S	I	-	-	-	✓	-	-	-	G
Hong et al. (2022)	S	I	-	-	-	✓	-	-	-	G
Dong et al. (2022)	S	I	-	-	-	✓	-	-	-	G

Table 3.2: The table shows systematic review results for OSS and the selected articles of OSR. Data is ordered by task (column T) and by publish year. Columns stand for, respectively: T - main task tackled (S - segmentation, R - recognition); D - data type (I - 2D image, RS - remote sensing image); R - if the model uses image reconstruction somehow; A - if it uses auxiliary data; G - if it uses generative modeling; S - if it uses any statistical modeling; F - if it uses the intermediate feature space to model open-set distributions; P - if the model can be used in a plug & play fashion; E - if the method uses EVT to model open-set distributions; and SE - the source of the article (M - manually included; W - Web of Science; S - Scopus; and G - Google Scholar).

3.1.4 Open-set Recognition

A comprehensive view of the OSR was presented in Section 2.7 using the taxonomy presented by Geng et al. (2020). This section presents the seminal manually chosen articles for each category of the unified taxonomy for OSS and OSR. They represent well the examples of methods that fall upon the proposed categories and can be considered the base of more recent approaches.

Deep statistical models for OSR can operate either solely on the output activations of a model $\mathcal{M}$ (Bendale and Boult, 2016; Gawlikowski et al., 2022; Bharadwaj et al., 2022) or also consider the intermediary feature representations of a closed-set classification network (Vendramini et al., 2021; Sun et al., 2020), as shown in Figure 3.3(a). An important subset of this OSR paradigm specializes in using EVT for detecting OOD samples. An example of this case is the traditional OpenMax algorithm (Bendale and Boult, 2016), which adds an “unknown” output class and estimates the probability of the input images to each of the $C + 1$ classes, where C is the number of known categories. Extreme value theory is a robust theoretical framework to work with long-tailed distributions and anomaly detection, but is usually limited to working directly on logits, not including intermediate feature representations. Vendramini et al. (2021) showed how simple generative models (i.e. principal component analysis or Gaussian mixtures) surpassed the performance of OpenMax considerably in multiple traditional OSR scenarios by introducing information from the middle layers of a CNN $\mathcal{M}$.

Following a rather distinct paradigm, reconstruction-based strategies (Figure 3.3(b)) (Oza and Patel, 2019; Yoshihashi et al., 2019; Sun et al., 2020) leverage reconstruction error from auto-encoding networks (e.g. auto-encoders and their variants) in order to delineate the boundary between known and unknown samples. These strategies rely on the reconstruction error from known classes being smaller than reconstruction errors from unknown classes, since the training is only with known samples. Multiple articles (Oza and Patel, 2019; Yoshihashi et al., 2019) repurpose the closed-set classification encoder $\mathcal{E}$ and attach a trainable decoder $\mathcal{D}$ to try to reconstruct the input image for the known classes. C2AE (Oza and Patel, 2019) exemplify the reconstruction paradigm for OSR quite well by merging a closed-set classification encoder $\mathcal{E}$ pre-trained on the known classes with an upsampling decoder $\mathcal{D}$ for reconstruction. $\mathcal{E}$ works both to classify among KKCs and to compress the representation of the input samples into an embedding that can be reverted to an approximation of the input space by $\mathcal{D}$. In this strategy, wrongly labeled samples are purposely fed to the network to enforce that it is able to only reconstruct samples correctly conditioned to the input label.

Conditional Gaussian Distribution Learning (CGDL) Sun et al. (2020) is a variation of the traditional reconstruction-based pipeline that couples the reconstruction loss with a Kullback Leibler (KL) constraint on network activations – effectively working as a cascaded VAE (Kingma and Welling, 2013). The KL divergence is used during the training phase in order to enforce simpler gaussian bottleneck embeddings before the reconstruction. CGDL is framed as statistical and reconstruction-based due to the use of both reconstruction loss and the KL divergence.

At last, the third OSR strategy uses known unknown samples as auxiliary data to ensure that the model learns to differentiate between known $\mathbf{x}$ and unknown samples $\mathbf{x}_{OOD}$. This OSR paradigm leverages a known set of unknown samples – henceforth known as the support set – to transform the usually unsupervised generative modeling of OSR into a supervised discriminative process. For instance, G-OpenMax (Ge et al., 2017) employs a Generative Adversarial Network (GAN) $\mathcal{B}$ trained on OOD data to learn how to discriminate the known classes (classified through the closed-set branch $\mathcal{M}$) from synthetic samples. In the same direction, the Outlier Exposure (Hendrycks et al., 2018) model uses different datasets as OOD samples in the open *vs.* closed branch $\mathcal{B}$ to learn how to discriminate the known distribution from others. Figure 3.3(c) shows an example of this class of methods.

3.1.5 Open-set Semantic Segmentation

Statistical Modeling is the most common background structure used by OSS methods varying the usage from method to method. PCA, GMM, entropy, or probability produce anomaly scores from intermediate features or final layers of model $\mathcal{M}$ to distinguish and characterize OOD via threshold τ (Oliveira et al., 2021; Martinez et al., 2021; Hong et al., 2022; Chan et al., 2021; Cui et al., 2020; Grcić et al., 2021, 2022). OpenPCS (Oliveira et al., 2021) and OpenPCS++ (Martinez et al., 2021) use PCA to reduce the dimensionality, generating a representation of the stacked intermediate features and the final layers. A threshold is employed in the resulting log-likelihood to identify OOD pixels. An advantage of both OpenPCS and OpenPCS++ is the "plug & play" characteristic, which allows a fast adaptation of the method and the use in either new datasets or different closed-set backbones. Another related work proposed by Cui et al. (2020) applied a statistical test to the produced entropy-uncertainty map to determine if any area is unknown. Other representational strategies employed are Metric Learning and Prototyping, as in Cen et al. (2021) and Dong et al. (2022), using the calculated distance between representations and each sample to define which pixels are OOD. Methods proposed by da Silva et al. (2020); Oliveira et al. (2021) and Martinez et al. (2021) use EVT to model the final score or loss distribution and to separate OOD objects from the known objects. A different approach uses probability sampling to balance sample selection and improve the learning for the method proposed by Yan et al. (2021).

Reconstruction-based strategies are employed in only one method in OSS. In general, reconstruction-based methods use the reconstruction error $\mathcal{L}$ to identify OOD pixels. The only reconstruction method (Nunes et al., 2022b) found in our search uses conditional reconstruction to identify OOD pixels. In training, the method learns to reconstruct pixels conditioned to their class, and in testing, all pixels are conditioned to all known classes. The ones from unknown samples tend to present higher reconstruction loss values, thus being set as OOD by a threshold τ.

Auxiliary data had three different usages mapped in this study. The first uses synthetic images (Kong and Ramanan, 2021; Grcić et al., 2020, 2021). The method proposed by Grcić et al. (2020) employs synthetic negative patches added to images that simultaneously achieve uniform discriminative prediction and high inlier likelihood. Also, the Jensen-Shannon divergence was employed in both training and inference instead of the Kullback–Leibler (KL) divergence. The Jensen-Shannon divergence mildly penalizes high confidence predictions in comparison to KL-divergence. The OpenGAN method (Kong and Ramanan, 2021) learns a robust open-vs-closed discriminator $\mathcal{B}$ that serves as open-set likelihood. $\mathcal{B}$ is trained with fake (synthetic) data from a generator and real open training examples as an outlier exposure strategy. As the GAN objective is not a realistic reconstruction, both generator and discriminator $\mathcal{B}$ use the features of the closed-set model $\mathcal{M}$, which enables readily modifying closed-set systems for open-set recognition.

The combination of synthetic data and OE together is the second mapped usage of auxiliary data. We highlight that OpenGAN was the only work found that used synthetic data and OE together to enhance the discriminative ability of the model.

Finally, the third mapped usage is a strategy that randomly replaces a small crop of the input image with some OOD mini-patch (Grcić et al., 2020, 2021; Bevandić et al., 2022). In Bevandić et al. (2022), the mini-patch is a random crop of a real image of the same size but with a different distribution. In Grcić et al. (2020, 2021), the mini-patch is synthetic. For this sort of approach to work, the ground truths must be equally modified including the unknown class to the added mini-patch area. The model trains to differentiate in-distribution and out-of-distribution and to correctly identify OOD pixels.

3.2
Strongly Related Works

The methods presented in this section were selected among the results of the systematic review in Section 3.1.

3.2.1
OpenPixel

The OpenPixel method proposed by da Silva et al. (2020) was the first CNN for OSS. It uses a patch-wise strategy with a patch or context window of 55x55 pixels to classify the central pixel. The CNN trains to classify each patch by iterating over each image's pixel. If the probability given by the softmax layer is below a threshold, the pixel is set as unknown.

Morph-OpenPixel was proposed by adding a morphological filter at the end of OpenPixel's network. The morphological filter is a post-processing technique to delineate object boundaries more precisely. Figure 3.5 shows the representation of OpenPixel and the morphological post-processing. The identified unknown pixels are subject to the morphological filter to determine if a pixel belongs to a border of an object. The class of the pixel is set to the same class as most of its neighbors whenever the pixel belongs to a boundary. The obtained results show that Morph-OpenPixel improved the overall accuracy and Kappa.

This strategy has the major disadvantage of creating a patch for each pixel, which makes the entire process extremely expensive and unfeasible in real-world scenarios.

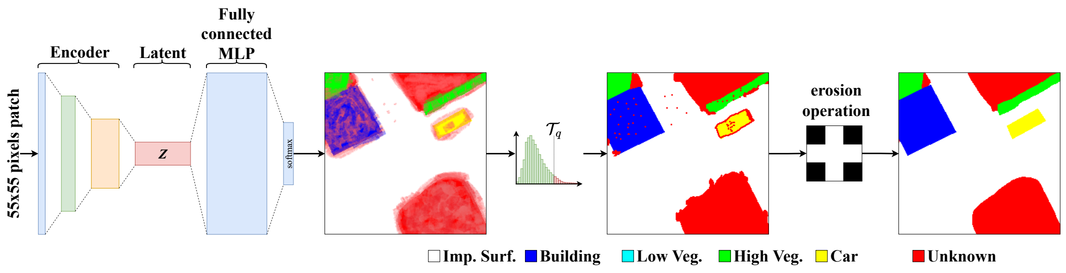


Figure 3.5: OpenPixel and Morph-OpenPixel architectures. The OpenPixel representation goes up to the semantic map. Morph-OpenPixel includes a morphological filter for post-processing the OpenPixel output.

3.2.2 Open Fully Convolutional Network

Open Fully Convolutional Network (OpenFCN) proposed by Oliveira et al. (2021) was the first fully convolutional model proposed to OSS. OpenFCN extends traditional FCN-based architectures for the OSS task. Traditional FCN-based models usually consist of a CNN backbone with inference layers replaced by a spatial expansion strategy and more convolutions. OpenFCN is first trained as closed-set semantic segmentation, using the SoftMax layer to compute prior probabilities for the known classes.

To compute posterior prediction, OpenFCN uses the same protocol as OpenMax Bendale and Boult (2016), which relaxes the requirement that the sum of the prediction probabilities for KKCs equals 1. An additional class is added to the posterior prediction, and the OpenMax function reweights the SoftMax predictions to account for the misclassification probability. During the validation, using the validation data, a Weibull distribution is calculated for each KKC from the correctly classified pixels. A threshold is defined from the quantiles of the Cumulative Distribution Function (CDF) for the Weibull distributions. All pixels below the threshold are unknown. The final OpenFCN segmentation had boundary issues among adjacent objects and, in many cases misclassified pixels within these areas.

3.2.3 Open Principal Component Scoring

OpenPCS works similarly to CGDL (Sun et al., 2020) with three key differences: uses PCA instead of a VAE; training is purely supervised, and the closed-set semantic segmentation is detached from the fitting of the Gaussians that occurs only in the validation phase.

OpenPCS combines feature maps from earlier layers with feature maps from the latest layers of the model, merging low and high-semantic-level information. According to Shwartz-Ziv and Tishby (2017), a supervised DNN can be seen as a Markov chain that gradually transforms the input space into the output space. By adding earlier intermediate activation layers to fit the Gaussian, OpenPCS uses activations with high spatial level information as they approach the output space and high semantic information as the activations approach the input space.

As can be seen in Figure 3.6, to combine the intermediate activation layers they must be upsampled to match the spatial resolution of the output prediction represented as the $\uparrow$ function before the concatenation of the layers. The concatenation of the intermediate layers can produce a high-dimensional

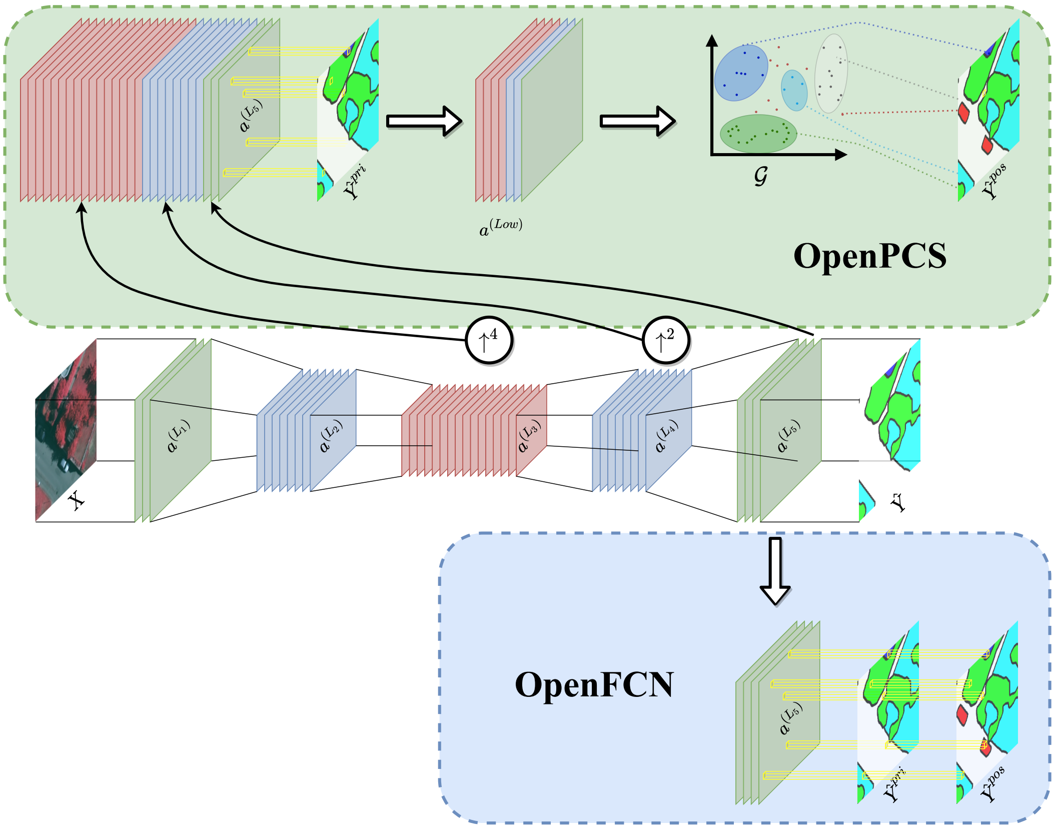


Figure 3.6: During training both OpenFCN and OpenPCS behave like a traditional closed-set FCN for semantic segmentation for the KKCs. The closed-set FCN is shown in the middle of the figure. During validation, OpenFCN computes OpenMax and the Weibull distributions. During testing, the probabilities for OSS are thresholded to predict the unknown pixels. OpenPCS concatenates the activation maps (in this example $a^{(L_3)}$, $a^{(L_4)}$ and $a^{(L_5)}$). $a^{(L_3)}$, $a^{(L_4)}$ are scaled up to the dimensions of $a^{(L_5)}$ to produce a column vector for each predicted KKC pixel. OpenPCS reduces the concatenated high-dimensional feature space to a low-dimensional ($a^{(Low)}$) space using the Principal Components. For each KKC, a multivariate Gaussian is fitted, and an array of log-likelihoods is thresholded to identify the OOD pixels. Adapted from Oliveira et al. (2021).

feature space since modern FCNs may have up to thousands of channels for each layer. OpenPCS compute a low dimensional feature space with PCA before fitting a Generative model G to recognize OOD pixels.

OpenPCS-based approaches have some key advantages: it is simple to adapt new FCN backbones and datasets since the closed-set training is detached from the OOD prediction; PCA can be accelerated through parallelization to compute the log-likelihood scoring; PCA dimensionality reduction is highly effective in identifying the most representative activation channels to compute the scoring function to detect UUCs.

OpenPCS++ proposed by Martinez et al. (2021) is a variation of OpenPCS that uses the statistical whitening transform as a feature normalization for better stability in known class likelihood scoring space. The transformation that enforces data to have an identity covariance matrix is known as the statistical whitening transform. This transformation makes the dimensions statistically independent, and the variance of each dimension equals one. This operation equals the weights of the dimensions of the scoring space used to detect the OOD pixels.

3.2.4
Class Conditioned Auto-Encoder for Open-set Recognition

The method called C2AE proposed by Oza and Patel (2019) divides the OSR task into sub-tasks: closed-set classification, open-set training, and open-set testing. Figure 3.9 shows a simplified schema of C2AE based on the original paper.

The Closed-set classification uses a shallow classifier on top of an Encoder (F) to extract features and perform on the MNIST dataset. After the closed-set training, (F) weights are frozen and used in other parts of the method. The closed-set encoder and classifier are trained with the traditional Cross Entropy loss.

Open-set training shown in Figure 3.7 is split into two parts: conditional decoder training; and EVT modeling of the reconstruction errors. For the conditional decoder training, the frozen encoder F produces a latent representation of the inputs. The latent representation is then conditioned using FiLM (Perez et al., 2018) and feeds the Decoder G. The G reconstruction is expected to be perfect if conditioned to the correct class.

The input is conditioned to its match class and to non-match class. With this training strategy, G learns to output a poor reconstruction when conditioned to a non-matching class and a good/perfect reconstruction when conditioned to the match class, emulating an open-set scenario. EVT models the match and non-match class reconstruction errors. The optimal operating threshold lies between the match and the non-match distributions and minimizes the probability of errors for that given model.

Open-set testing shown in Figure 3.8 condition the input to all KKC classes and compute the reconstruction errors. If the minimum reconstruction error is below the previously calculated threshold, the shallow classifier output is returned. If the minimum reconstruction error is greater than the threshold, the returned class is unknown.

3.3
Discussion and Literature Trends

In general, OSS methods are based on an OSR counterpart. Hence, our proposed taxonomy works for both tasks since the fields share similar strategies. Reconstruction-based methods might be an ongoing trend for OSS since some of the more robust methods in OSR rely on reconstruction (Oza and Patel, 2019; Yoshihashi et al., 2019; Sun et al., 2020). Our systematic review only found one method proposed in this thesis for OSS that uses reconstruction (Nunes et al., 2022b), which means that this type of strategy is still in its earlier

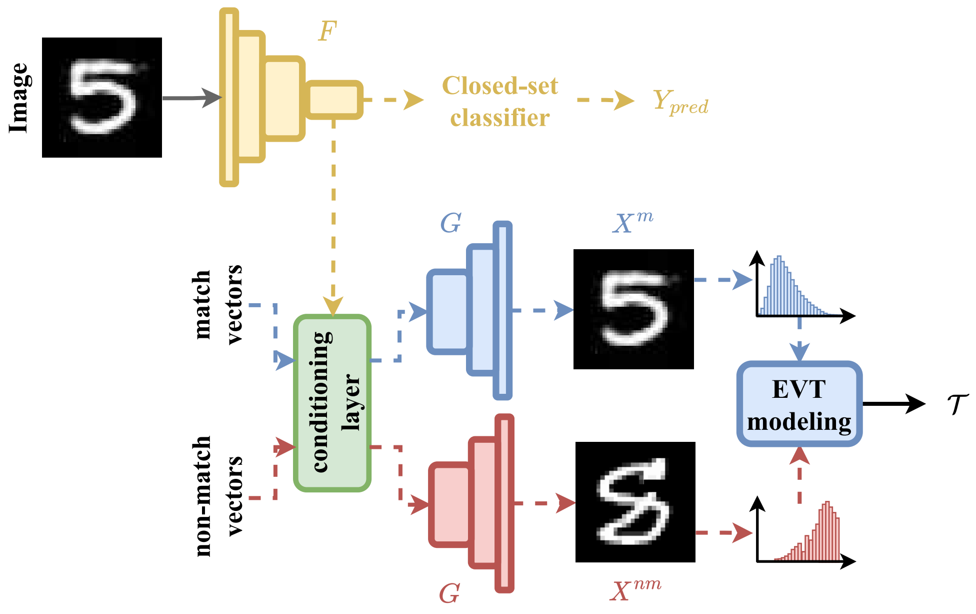


Figure 3.7: Simplified training schematics

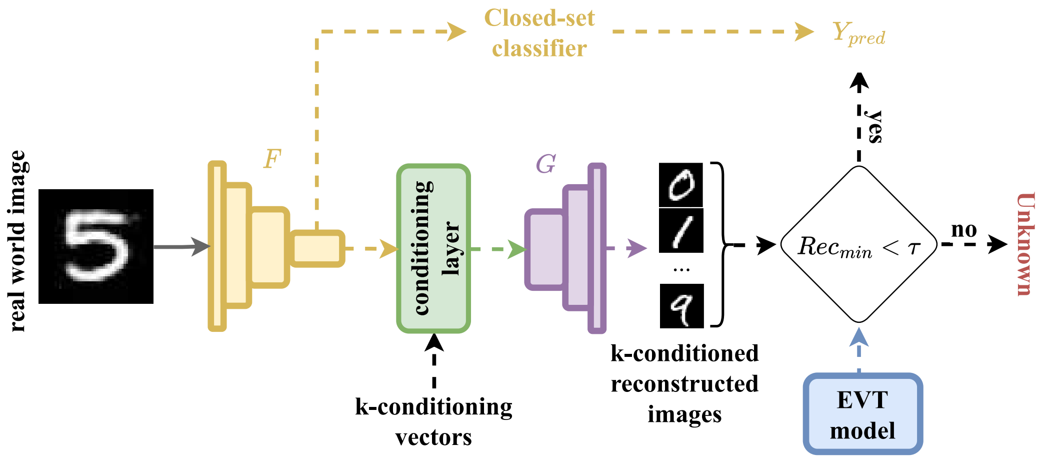


Figure 3.8: Simplified testing schematics

Figure 3.9: The figure shows the C2AE schema divided into three phases. **1) Closed-set pre-training** follows the traditional closed-set training with an Encoder (F) and a shallow classifier. **2) Open-set training** shown in Figure 3.7, uses the pre-trained closed-set encoder F with its weights frozen. F is used to train a decoder to reconstruct the input conditioned to the label. Reconstructions conditioned to the correct class yield a better reconstruction (smaller error value) than reconstructions conditioned to the wrong class (higher error value). In the end, EVT models reconstruction errors defining the operating threshold. **3) Open-set testing** shown in Figure 3.8, each input is conditioned to every KKC getting the minimum error reconstruction. The model yields the classification of the shallow classifier if the minimum reconstruction error is below the threshold, otherwise, it is unknown.

steps and showed strong results compared to the other baseline OSS methods.

A major gap found in the literature that seems relevant for OSS is the lack of methods to improve the semantic consistency of the segmentation, particularly as boundaries across objects from distinct classes tend to present larger segmentation errors. Many open-set strategies employ some confidence or anomaly score to identify OOD pixels. Thus, post-processing schemes capable of mitigating the lack of confidence in border regions between different objects may improve open-set segmentation prediction results. In this direction, techniques like visual attention modules, CRFs, and superpixel post-processing are promising alternatives to be explored in future works. Another approach proposed in this work tackles this gap and uses superpixel post-processing to improve semantic consistency for OSS.

Finally, zero-shot and few-shot tasks overlap open-set tasks since the knowledge in these scenarios is inherently incomplete during training, and the method may need to handle samples of unknown classes during the deployment phase, possibly even using some online learning strategy. Developments in the literature of OSR/OSS and zero-/few-shot learning (Cen et al., 2021; Zhou et al., 2021; Saito et al., 2021) seem to walk towards each other, possibly resulting in future deep Open World (Bendale and Boult, 2015) approaches.

4
Proposed Methods

In this chapter, the three proposed approaches are presented in detail. The Section 4.1 presents Open Gaussian Mixture of Models (OpenGMM) as an extension of the OpenPCS method proposed by Oliveira et al. (2021), which was the first model developed; The Section 4.2 presents the second model developed called Conditional Reconstruction for Open-set Semantic Segmentation (CoReSeg); and Section 4.3 presents the post-processing developed and the novel superpixel generation method called Fusing Superpixels for Semantic Consistency (FuSC) both applicable to any OSS method.

4.1
Open Gaussian Mixture of Models

OpenGMM builds upon the previously proposed OpenPCS (Oliveira et al., 2021) method that uses PCA to compress the representation extracted from the backbones and uses the generated representation to do the OOD pixel detection. OpenGMM replaces PCA with GMM (Rasmussen, 2003). GMM's multimodal representation should be better suited for real-world pixel modeling that may not conform to unimodal representations.

Like OpenPCS, OpenGMM uses intermediate feature maps together with the last layer activation maps. Combining the activations from the earlier layers with the latter layers produces a tensor that fuses low and high-semantic-level information. The concatenated tensor may have hundreds or thousands of channels, which is known to be redundant (Sun et al., 2020; Huang et al., 2017). OpenGMM fits GMM rather than PCA like OpenPCS to deal with the concatenated tensor size and redundancy of activations. The GMM is applied to model each closed-set KKC distribution, generating as many models as KKCs. Each GMM model generates a score tensor with the log-likelihood values for all pixels and generates a final score tensor combining all GMM scores with the closed-set prediction. The produced final score tensor is used to set all pixels below the threshold as unknown or OOD.

The OpenPCS framework proposed by Oliveira et al. (2021) and, as an extension, OpenGMM allow for multiple closed-set backbones. We adopt three backbones for OpenGMM and OpenPCS: DN-121 (Huang et al., 2017), WRN-

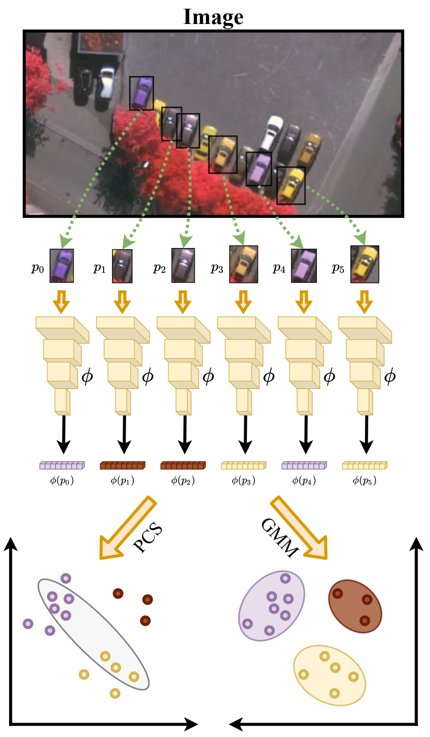


Figure 4.1: The figure shows an example of how different objects can be represented by distinct data distributions. Due to the multimodal representation capability, GMM is better suited for representing real-world data than Open-PCS (Oliveira et al., 2021).

50 (Zagoruyko and Komodakis, 2016) and U-net (Ronneberger et al., 2015). We also used the same three backbones with CBAM layers as described in Section 5.1.

Readers should notice that adapting any closed-set semantic segmentation network to the OpenGMM and OpenPCS frameworks is relatively quick and simple to implement and deploy, without requiring retraining or additional layers to be trained. This promptness contrast to other existing OSR/OSS methods (Hendrycks et al., 2018; Yoshihashi et al., 2019; Sun et al., 2020) or even CoReSeg also proposed in this work (Nunes et al., 2022b) that need to be retrained.

The plug-and-play characteristic of methods such as OpenPCS and OpenGMM is a great advantage when considering the problem of adapting the solution to real-world applications and novel domains.

4.2 Conditional Reconstruction for Open-set Semantic Segmentation

CoReSeg employs a pre-trained closed-set neural network to generate a latent representation of the input image. This latent representation is used to perform a closed-set prediction and as input for the reconstruction decoder. This decoder also receives conditioning information, whose objective is to guide the reconstruction of the input image from its latent features conditioned to the desired class (or classes). An overview of this process is illustrated in Figure 4.2.

Our method is inspired by some design choices proposed by Oza and Patel (2019), wherein the conditioning concerns the entire image, while for CoReSeg the conditioning is performed in a pixel-wise manner. The main idea of the proposed method is that objects from known classes will be better reconstructed when their pixels are conditioned to the correct class, while unknown objects will present poor reconstructions since there is no correct conditioning to these pixels.

The following subsections will detail the architecture choices, the tried variations, the two sequential training steps, and the testing or deployment procedure.

4.2.1 Reconstruction

CoReSeg uses an auto-encoder to reconstruct the input image conditioned to its mask. Auto-encoders learn how to reconstruct images, typically minimizing some measure of reconstruction error as a loss. As used for anomaly detection, CoReSeg uses reconstruction errors to identify pixels of unknown classes.

Auto-encoders detailed in Section 2.3 typically use a bottleneck architecture that contracts the spatial dimension while increasing the number of channels learning general abstract latent representations. Well-trained reconstruction auto-encoders learn good latent representations of the trained distributions and can correctly reconstruct objects of known classes. It is expected that a well-trained network could correctly reconstruct objects of known classes (distributions), while could not represent the unknown or anomalous objects with the learned representations and thus could not reconstruct objects of unknown distributions. Since reconstruction errors are higher for objects of unknown classes, the model can identify anomalous pixels and set them as unknown.

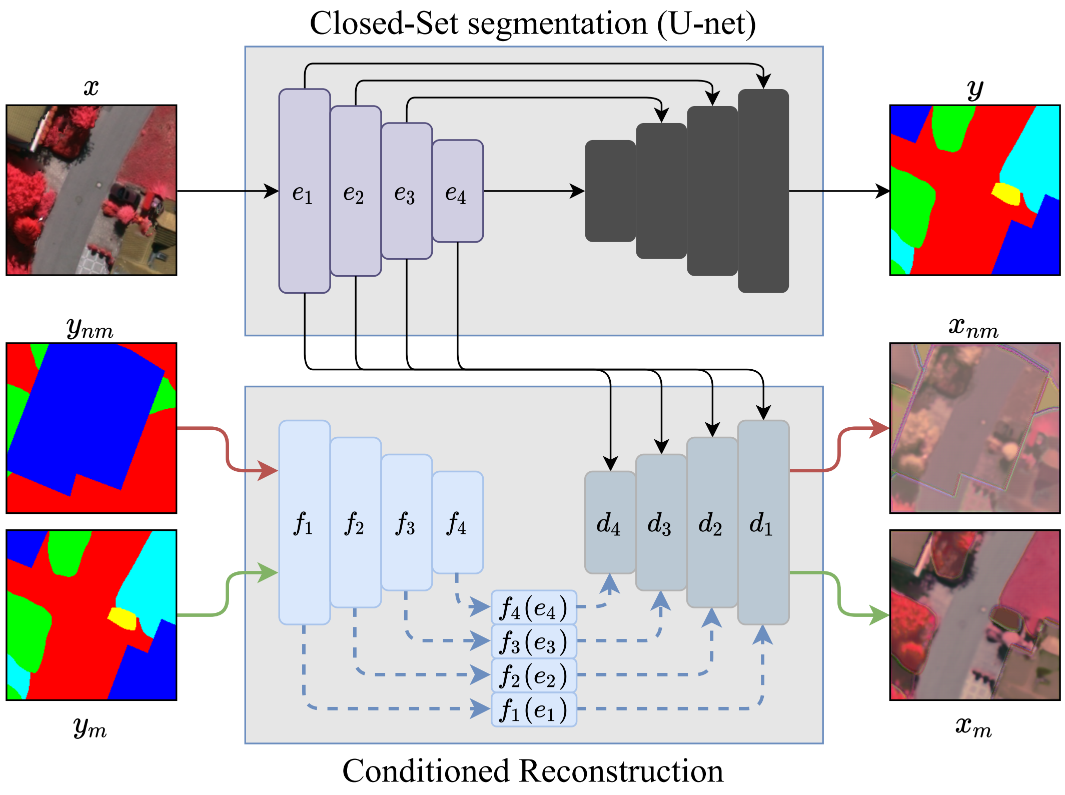


Figure 4.2: Training schema where e_i denotes a layer on the closed-set encoder, d_i denotes a layer on the reconstruction decoder, and f_i denotes a simplified FiLM conditioning layer that has two encoders β and γ. The model is trained to reconstruct each image with matching and non-matching masks as a way of enforcing the conditioning with good (match) and poor (non-match) representations of the original image.

4.2.2 Conditioning

The conditioning mechanism is central for CoReSeg the model must condition the reconstruction decoder to differentiate each known class and better approximate the reconstruction from the input. The proposed model assumes that poorly reconstructed pixels are prone to be unknown. The conditioning mechanism encourages the model to learn shared features, like textures and colors, to represent the known classes.

Our first attempt to condition the features extracted from the closed-set encoder e_x was to concatenate a generated tensor c_x with the same spatial dimensions of e_x for every layer as Figure 4.3 shows. The reconstructed images were very similar to the original and did not make the class characteristics visible.

Inspired by Perez et al. (2018), we tried three conditioning mechanisms: biasing or additive, scaling or multiplicative and affine transformation that combines biasing and scaling.

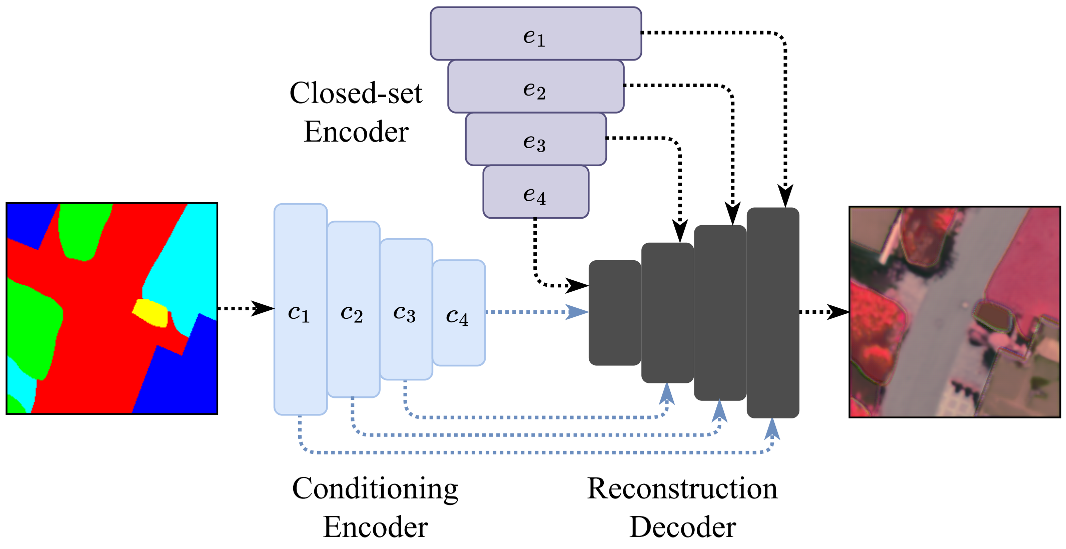


Figure 4.3: Concatenation conditioning - tensors from the c_x conditioning encoder with the e_x closed-set encoder are concatenated and then processed by the reconstruction decoder.

As reported by Perez et al. (2018) affine transformations as a conditioning mechanism were more robust and yielded better results than the three other tested mechanisms. The Feature-wise linear modulation (FiLM) (Perez et al., 2018) was adapted to be a pixel-wise mechanism capable of conditioning the entire encoder respecting the spatial dimensions of each layer. Figures 4.4, 4.5 and 4.6 show the schematics of the three explained mechanisms: additive, multiplicative and affine.

4.2.3 Architectural Variations of CoReSeg

CoReSeg is built on the idea that the result of the reconstructions allows the identification of OOD. With this in mind, we tried some different configurations for the architecture, looking for a final architecture capable of improving the ability to differentiate and identify known classes and unknown classes. The ablation results for the variations presented in this section are presented in Chapter 6.

The first proposed model, referred to as the base model, can be seen in Figure 4.7(a) and used the skip connections from the closed-set encoder conditioned by the chosen conditioning mechanism as input for the reconstruction decoder. The variation proposed, referred to as the full model, in Figure 4.7(b) uses the raw skip connections to concatenate with the conditioned skip connections as input for de reconstruction decoder.

Well-trained reconstruction or segmentation models have higher uncertainty in reconstructing or segmenting objects' borders. We include attention

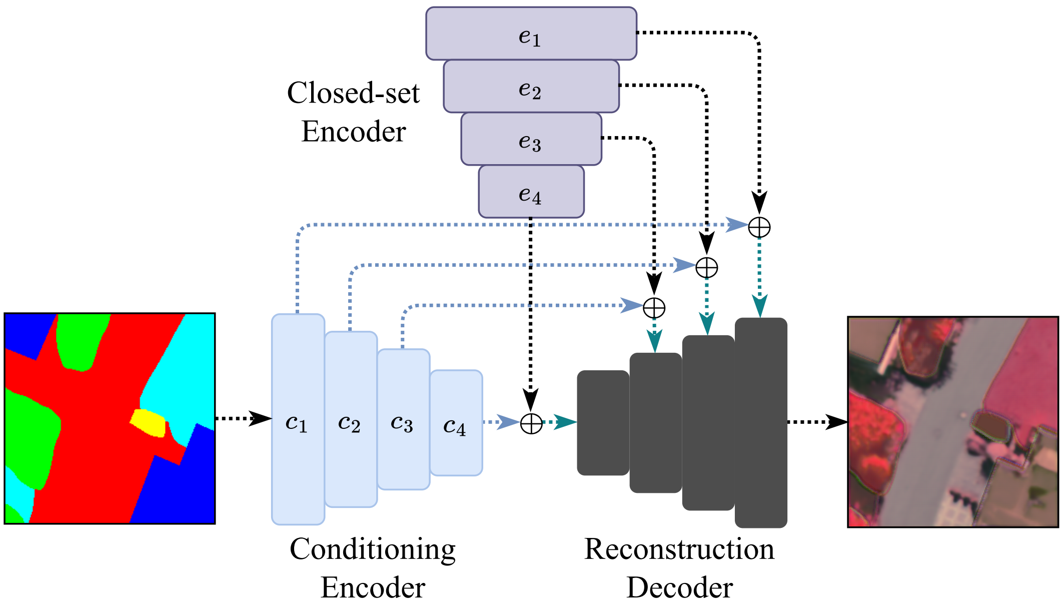


Figure 4.4: Additive conditioning - a pixel-wise (element-wise) summation is computed between the c_x conditioning encoder with the e_x closed-set encoder, and the resultant tensor is then processed by the respective reconstruction decoder.

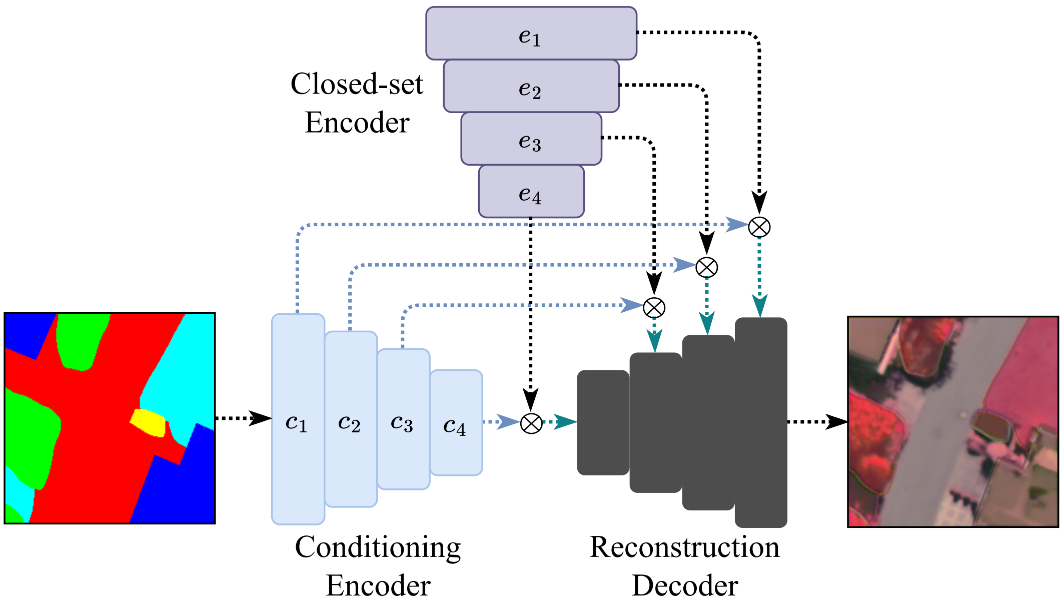


Figure 4.5: Multiplicative conditioning - a pixel-wise (element-wise) multiplication is computed between the c_x conditioning encoder and the e_x closed-set encoder, and the resultant tensor is then processed by the respective reconstruction decoder.

mechanisms just before each decoder layer as a mechanism to help CoReSeg in an attempt to improve the OOD pixels recognition by focusing on the most relevant features while reconstructing the input. Figure 4.8 shows the schematics of the two variations of CoReSeg with the attention mechanism as explained in

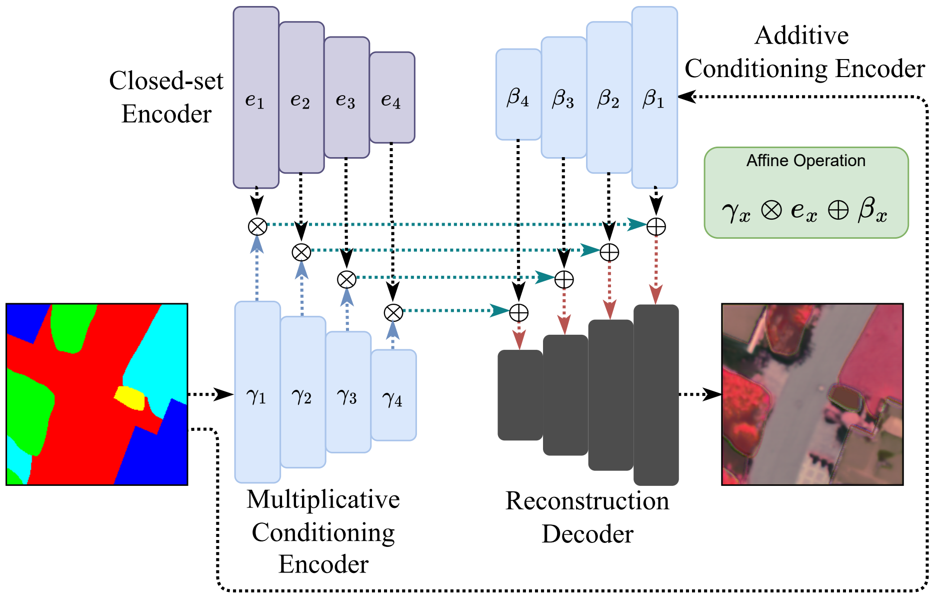
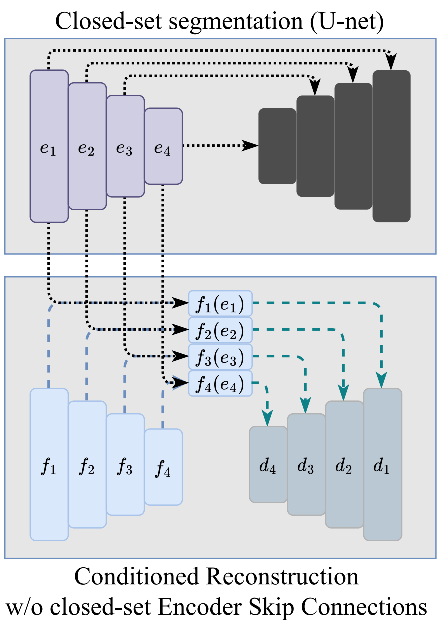


Figure 4.6: Additive-multiplicative conditioning - an affine operation for each element in the closed-set encoder computed with the pixel-wise (element-wise) multiplication between the γ_x conditioning encoder and the e_x closed-set encoder, also computed a pixel-wise (element-wise) summation between β_x conditioning encoder and e_x. The affine operation $f_x = \gamma_x \times e_x + \beta_x$ is computed for every parameter in the closed-set encoder and used as input to the reconstruction decoder.

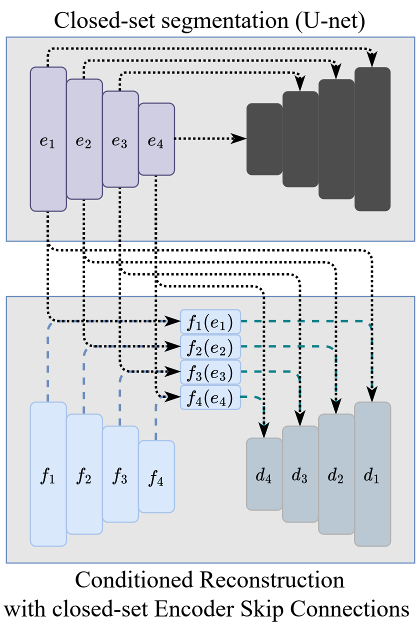
Section 2.4.1. Figure 2.7 summarizes the channel and spatial attention mechanism used. The CBAM mechanism employs 2 distinct attention mechanisms serially. First, a channel attention mechanism refines the input enhancing the most important feature maps, then a spatial attention mechanism enhances the most relevant areas of the image.

4.2.4 Closed-set Training

Regardless of the architectural variations for CoReSeg, the training procedure is the same and requires a pre-trained closed-set segmentation model to generate latent features that will be later used as input to the reconstruction decoder. For this reason, the first necessary step is to train a U-net (Ronneberger et al., 2015) in a traditional closed-set scenario, where only the known classes are learned. We call the encoder of this U-net the Closed Set Encoder, and its output is the desired latent representation for the conditional reconstruction training to be performed later. The resulting model will be used in both *Open-set Training* and *Deploy* phases. Also, the weights of this closed-set U-net will not change after its initial training, which means that the semantic segmentation layers are frozen during the training of the rest



4.7(a): Base model



4.7(b): Full model - closed-set skip connections to reconstruction decoder.

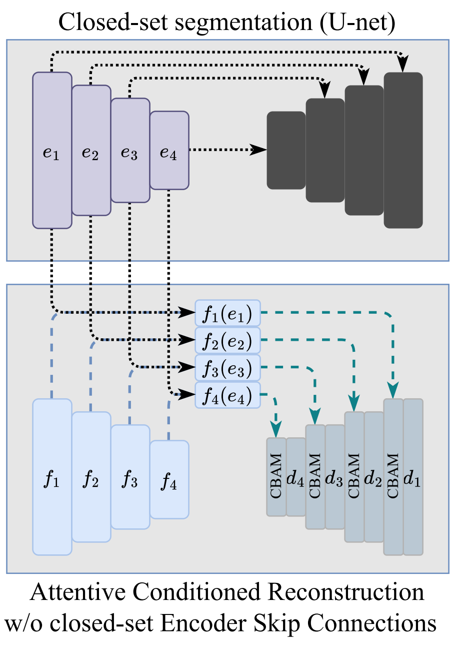
Figure 4.7: This figure shows the two variations for the use of skip connections in the reconstruction process. Figure 4.7(a) shows the proposed *Base model* using the closed-set skip connections only as input for the conditioning mechanism, Figure 4.7(b) shows a variation called the *Full model* that also concatenates the same closed-set skip connections used before with the conditioned tensor to use as input for the respective reconstruction decoder layer.

of CoReSeg's framework.

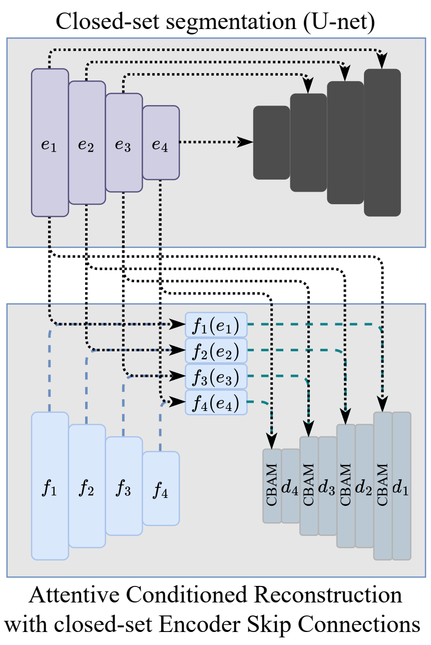
4.2.5
Open-set Training - Conditional Reconstruction

Conditional Reconstruction training aims at reconstructing the image that serves as input to the closed-set semantic segmentation block of the framework from its latent representation. The reconstruction is guided by a conditioning input, which, in the case of a semantic segmentation task, is comprised of a mask providing a class for each pixel. The conditional reconstruction block of the framework can be seen as an auto-encoder where the conditioning layers are the encoder, and the reconstruction layers are the decoder. Figure 4.2 shows how these different parts of the training are connected.

To train the model to reconstruct the image taking into account the conditioning input, CoReSeg uses for each input image 2 different masks to condition the reconstruction. The first mask is the one correctly labeled for the image (match mask, y_m). The second one is the label from a different image


Closed-set segmentation (U-net)
e_1 e_2 e_3 e_4
$f_1(e_1)$ $f_2(e_2)$ $f_3(e_3)$ $f_4(e_4)$
f_1 f_2 f_3 f_4
CBAM d_4 CBAM d_3 CBAM d_2 CBAM d_1
Attentive Conditioned Reconstruction w/o closed-set Encoder Skip Connections

4.8(a): Base model


Closed-set segmentation (U-net)
e_1 e_2 e_3 e_4
$f_1(e_1)$ $f_2(e_2)$ $f_3(e_3)$ $f_4(e_4)$
f_1 f_2 f_3 f_4
CBAM d_4 CBAM d_3 CBAM d_2 CBAM d_1
Attentive Conditioned Reconstruction with closed-set Encoder Skip Connections

4.8(b): Full model - closed-set skip connections to reconstruction decoder

Figure 4.8: This figure shows both variations presented in Figure 4.7 adapted to use the CBAM (Woo et al., 2018) attention mechanism.

(non-match mask, y_{nm}), which means that it incorrectly conditions the pixels from the input image. The use of a non-match mask is important to make sure the network is effectively learning to condition the input while not simply reconstructing the image from the latent representation.

We employ the L1 loss in the reconstruction step. The final reconstruction loss $\mathcal{L}$ is computed as follows:

$$\mathcal{L} = L1(x, \hat{x}_m) + \alpha \times L1(x, \hat{x}_{nm}), \tag{4-1}$$

where x is the input image, $\hat{x}_m$ and $\hat{x}_{nm}$ are the reconstructions conditioned on y_m and y_{nm} respectively, while α weights the importance of each term.

Aiming to enforce the conditioning, the encoder from the conditional reconstruction block applies a transformation to the intermediate features from the frozen Closed Set Encoder layers (e_i). The result of this transformation is then used as input on the corresponding layer of the reconstruction decoder (d_i). In Figure 4.2, this process is represented by the $f_i(e_i)$ blocks and it is performed for both match and non-match conditioning masks. The transformation responsible for the conditioning is the FiLM method proposed by Perez et al. (2018) extended to work in a pixel-wise problem. More specifically, to use the pixel-wise FiLM, the conditional reconstruction decoder is composed of two auxiliary encoders: β and γ. Both encoders have the same shape as e_i and d_i. To apply the transformation, we perform the following operation $\gamma_i \odot e_i + \beta_i$, where β_i and γ_i are the i^{th} blocks of the conditional reconstruction encoder and e_i is the output of the i^{th} block from the Closed Set Encoder. This procedure allows us to perform pixel-wise FiLM conditioning on e_i.

The reconstruction decoder uses as the main input the latent representation of the input image from the Closed Set Encoder. Furthermore, each layer of the reconstruction decoder receives two additional inputs concatenated to the previous layer activation: (i) the corresponding FiLM transformation (f_i) from the conditional reconstruction encoder, and (ii) the raw feature maps from the corresponding Closed Set Encoder (e_i). These concatenations can also be viewed in Figure 4.2.

CoReSeg uses a stochastic training strategy as in Oliveira et al. (2021). For each image in the dataset, a small number of randomized crops were selected to be reconstructed. For each selected crop (and its corresponding match mask), we choose from a randomized buffer of crops the one with the least overlap between its mask and the selected match mask to be the respective non-match image. This process is repeated for each crop inside the batch during training.

4.2.6 Why Use Non-match Masks in Training?

Unsupervised learning produces semantic representations of the distributions to the extent that those representations assist in reconstructing the input as in the training dataset. The network is not encouraged to learn features coupled with class information to reconstruct the input.

The conditional reconstruction adds the conditioning mask to the training procedure to enforce the network to learn features to distinguish among known classes. Using only the correct ground truth for the input image to condition the reconstruction allows the network to generate the needed latent representation ignoring the conditioning mechanism and using mainly (if not only) the closed-set encoder skip connections values. The empirical evaluation confirmed the tendency of the network to undervalue the conditioning mask and use mostly the closed-set encoder features values.

To overcome the tendency to undervalue the conditioning and encourage the network to learn how correctly condition the input to its class, we use a non-match mask in reconstruction training as detailed in Section 4.2.5. The non-match reconstruction error becomes part of the loss calculation, as Equation 4-1 shows.

We tested two non-match strategies: a non-match selection and a synthetic non-match generation. For the selection strategy, we selected a buffer of random crops of the image and compared it with the match mask selecting the one with the smallest intersection area. The synthetic generation strategy changes each class of the match sample mask by a random different class. In exploratory experiments, the synthetic generation strategy yields worse results. We will only report the results using the selection approach.

For this kind of training to produce the desired result, the match and non-match masks should have as few intersecting areas as possible, to force the model to learn better features for the reconstruction. Therefore, the need for distinct overlapping masks makes the selection of non-match crops much more difficult in sparse datasets. The quality of the generated latent representations strongly relies on crop selection and its intersection.

4.2.7
Deploy - Open-set Pixel Recognition

During deploy – shown in Figure 4.9 – we cannot provide match and non-match masks for the conditional encoder, as the labels for these samples are not available. So, to define which pixels are known and unknown CoReSeg tries to condition every pixel for each known class.

The input image is processed by the closed-set semantic segmentation block, generating a closed-set prediction. Then, the reconstruction decoder is conditioned with K masks, with K being the number of known classes, where all pixels from the mask m_k are set as the class k. Each one of these masks will provide a reconstructed output where all pixels were conditioned by the class k, and the corresponding reconstruction loss can be calculated from the input image for all of them.

Then, for each pixel the minimum error for $k \in \{1, 2, \ldots, K\}$ is computed and selected – where $\{1, 2, \ldots, K\}$ is the set of known classes. Pixels that were conditioned to the right class yield a small minimum error, while unknown pixels result in higher error values for each one of the reconstructions, since none of them match the right expected class. At last, a threshold operation defines which pixels are known and unknown. We use error quantiles to set thresholds and find the best performance for the model. If the minimum reconstruction loss of a pixel is below the threshold, its class is deemed as known and set to the closed-set predicted output, and otherwise, it is set as unknown.

Figure 4.10 shows the images produced by the conditioning mechanism proposed with impervious surfaces as UUC, from left to right, the figure

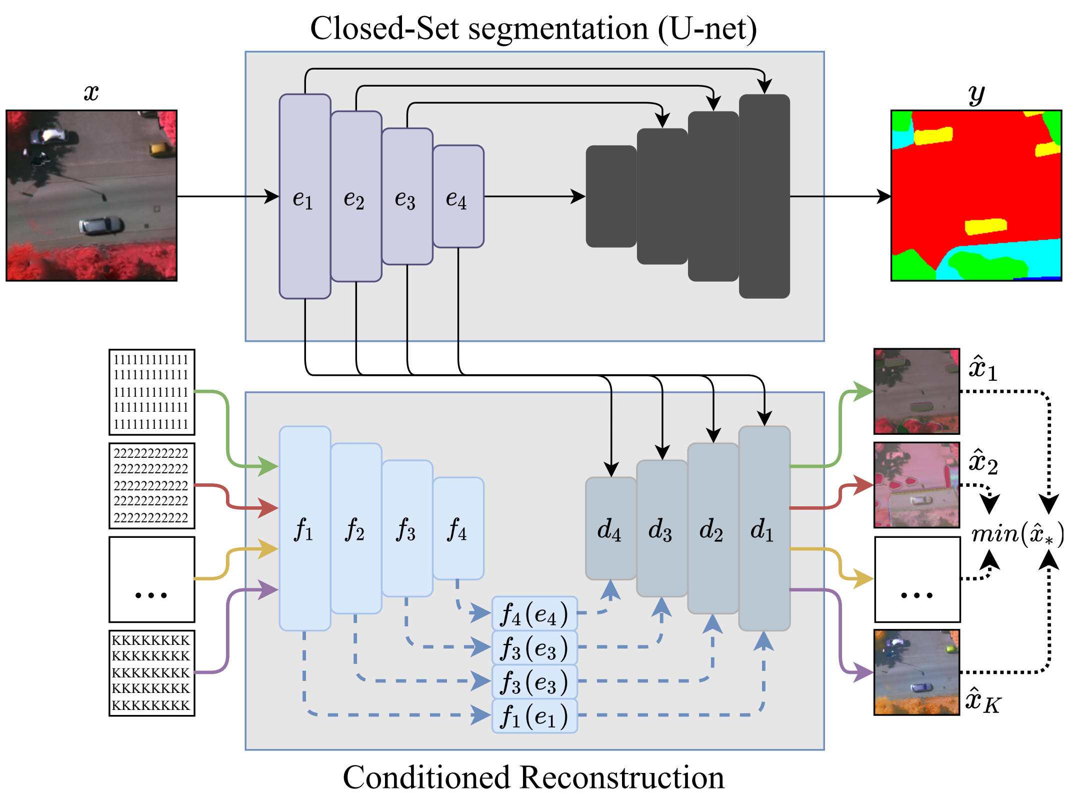


Figure 4.9: The figure shows the “Deploy” schema where e_i denotes a layer on the closed-set encoder, d_i denotes a layer on the reconstruction decoder, and f_i denotes a simplified FiLM conditioning layer that has two encoders β and γ.

shows the input image, the ground truth with the UUC in red, the closed-set prediction, the reconstruction errors conditioned to each of the four KKCs, and the computed minimum error at right.

In Figure 4.10, we can highlight that due to the absence of the class *impervious surfaces* during training, the closed-set model predicted the classes *car*, *low vegetation*, and *building* for most of the area of *impervious surfaces*. The shadows clearly influenced the predictions, and the borders of the cars were not well predicted.

We can also point in Figure 4.10 the influence of the conditioning mechanism on reconstructions, and for the reconstruction error images, the darker shades of gray indicate lower errors. From the left, *building* conditioned reconstruction error image shows the building areas are darker. *High vegetation* conditioned reconstruction image shows the high vegetation areas in a darker shade of gray. *Low vegetation* conditioned reconstruction error image shows cars and buildings in a lighter shade of gray with the original areas of *low vegetation* in a darker shade, highlighting that intra-class and inter-class variations make it harder to differentiate between low and high vegetation.

Car conditioned reconstruction error is the last of the four images, and the model could not reconstruct the image producing smaller errors for the cars.

The computed minimum error shows the impact of shadows on the composite error image. The composite image shows larger errors reconstructing areas in the presence of shadows or originally of low and high vegetation and impermeable surfaces.

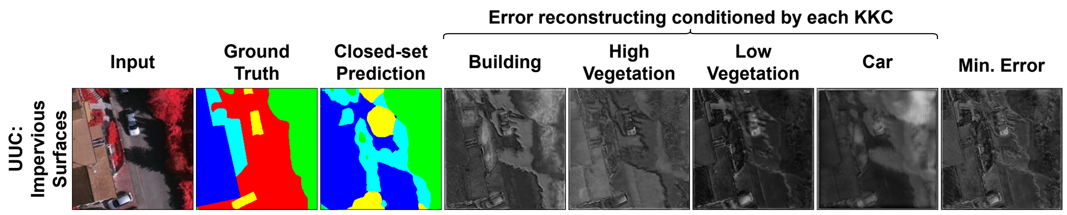


Figure 4.10: From left to right, the figure shows the input image, the ground truth with the UUC in red, the closed-set prediction, the four conditioned reconstruction error images, and the computed minimum error image. The figure shows *impervious surfaces* as UUC, to produce the scenario we used the LOCO protocol. The used colors are: white for *impervious surfaces*; dark blue for *building*; light blue for *low vegetation*; green for *high vegetation*; yellow for *car*; and red for the UUC. Also, the darker the pixel, the smaller the error.

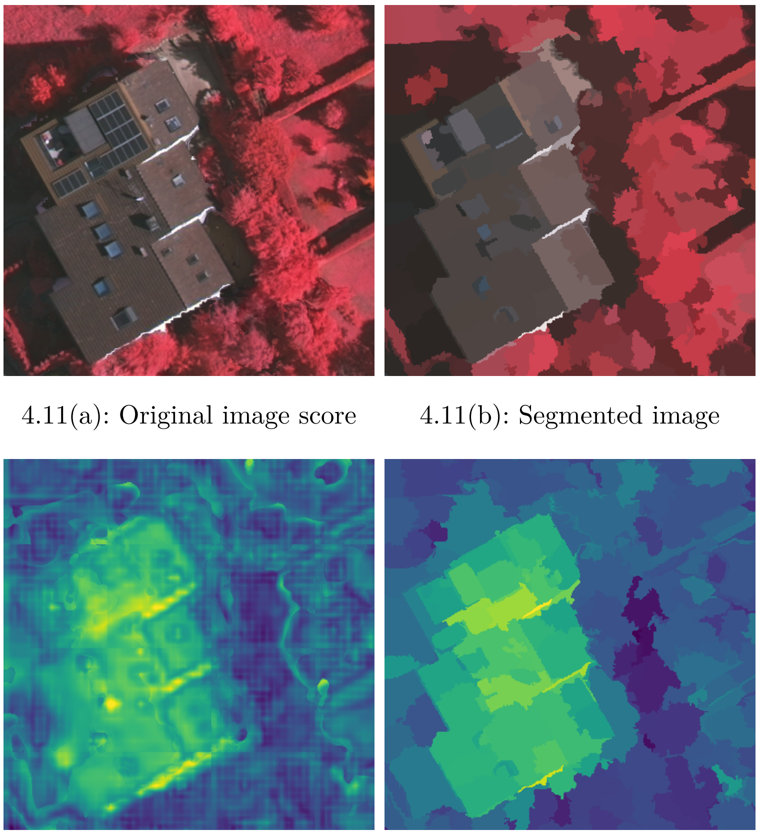
4.3 Improving Semantic Consistency with Superpixels

Superpixels are commonly employed before or during the segmentation process (Ji et al., 2020; Melas-Kyriazi and Manrai, 2021; Kang et al., 2021; Ratajczak et al., 2020; Zhang et al., 2014). In general, when employed as post-processing, the input image is used to generate the SPS and apply it somehow in the output prediction. This procedure produces more consistent borders among objects and tends to improve semantic consistency for the final segmentation. Following the literature, in the present work, we employ superpixels as a post-processing step applied to the scores returned by the OSS algorithms (i.e. reconstruction error, PCA/GMM likelihood, entropy, heat-map, etc.).

The OSS methods used in this work generate an output tensor of the same size as the input image containing the log-likelihood scores or reconstruction errors according to the method. The proposed post-processing computes the average value of each superpixel/segment producing a final score tensor with the segment's average value set to all pixels. Algorithm 1 details the usage of the superpixel over-segmentation in the final step just before the open-set recognition phase. While our post-processing scheme is agnostic to the choice of SPS algorithm, in this study we evaluated SLIC (Achanta et al., 2012), Quickshift (QS) (Vedaldi and Soatto, 2008) and Felzenszwalb (FZ) (Felzenszwalb and Huttenlocher, 2004). The three segmentation algorithms chosen have different generation characteristics and distinct pros and cons, as detailed in Section 2.6.

Figure 4.11(a) shows an example of an input image segmented using a superpixel algorithm in Figure 4.11(b). Figures 4.11(c) and 4.11(d) show the input image's output score tensor produced by an OSS algorithm and the same output tensor segmented using a superpixel algorithm. In a qualitative analysis, the segmented image improved the delineation of the objects and regions in the original image.

The theoretical complexity of the proposed procedure detailed in Algorithm 1 is linear to the number of pixels in the image (O(n) with n the number of pixels). Hence its use as post-processing is not computationally expensive and can be coupled with OSS methods to improve the quality of the final produced segmentation prediction.



4.11(a): Original image score 4.11(b): Segmented image

4.11(c): Base method score 4.11(d): Score post-processed

Figure 4.11: This figure shows the effect of the use of the proposed post-processing. The first line of images shows in Figure 4.11(a) the original image and in Figure 4.11(b) the segmented image produced using the same segmentation used to post-process the scores. The second line shows in Figure 4.11(c) the output score from the OSS method and in Figure 4.11(d) the superpixel post-processed score using the Algorithm 1.

The remaining steps of the open-set recognition process are kept the same for each OSS method. The superpixel over segmentations are homogeneous and tend to respect object borders. Applying the superpixels to the score image smooths the segmented areas, aiding the OSS algorithm in avoiding errors within the segmented objects.

The final superpixel segmentation reflects its generation characteristics. We can see in Figure 4.13 an illustrative example of two SPS that present different characteristics and may represent better different scenarios. The SLIC algorithm could better represent textures, while the FZ algorithm could better identify borders, but none of the single SPS could represent the underlying image properly. Figure 4.13 also compares the single SPSs with the Fusing Superpixels for the Semantic Consistency method proposed in the next section that produces improvement when compared to the single SPSs.

Algorithm 1 The algorithm used to apply superpixel segmentation to the output tensor of the OSS method. The complexity of the procedure is linear with respect to the number of pixels in the image (O(n) for n the number of pixels).

```
Require: image
Require: superpixel segmentation
Ensure: segmentations labels are sequential from 0 to number of segments
 1: $segments\_sum \leftarrow array(fill\_value = 0, size = max\_segment)$
 2: $segments\_px\_count \leftarrow array(fill\_value = 0, size = max\_segment)$
 3: $i \leftarrow 0$    ▷ Two arrays of the size of the number of segments are used to
    compute the mean value of the pixels for each segment. Each pixel of the
    image is visited and its value is added to the segment_sum array and one
    is added to the pixel count array
 4: while $0 <= i < image$ do
 5:     $j \leftarrow 0$
 6:     while $0 <= j < image$ do                    ▷ get the label of the pixel
 7:         $label = segments[i][j]$                   ▷ sum score value
 8:         $segment\_sum[label] += img[i][j]$          ▷ count the pixel
 9:         $segment\_pixel\_count[label] += 1$
10:         $j++$
11:     end while
12:     $i++$
13: end while                    ▷ compute the mean value for each segment
14: $segment\_value = segment\_sum/segment\_pixel\_count$
15: $i \leftarrow 0$
16: $new\_image \leftarrow image.clone()$    ▷ set the mean computed mean value to
    every pixel in the image
17: while $0 <= i < image$ do
18:     $j \leftarrow 0$
19:     while $0 <= j < image$ do
20:         $label = segments[i][j]$
21:         $new\_img[i, j] = segment\_value[label]$
22:         $j++$
23:     end while
24:     $i++$
25: end while
```

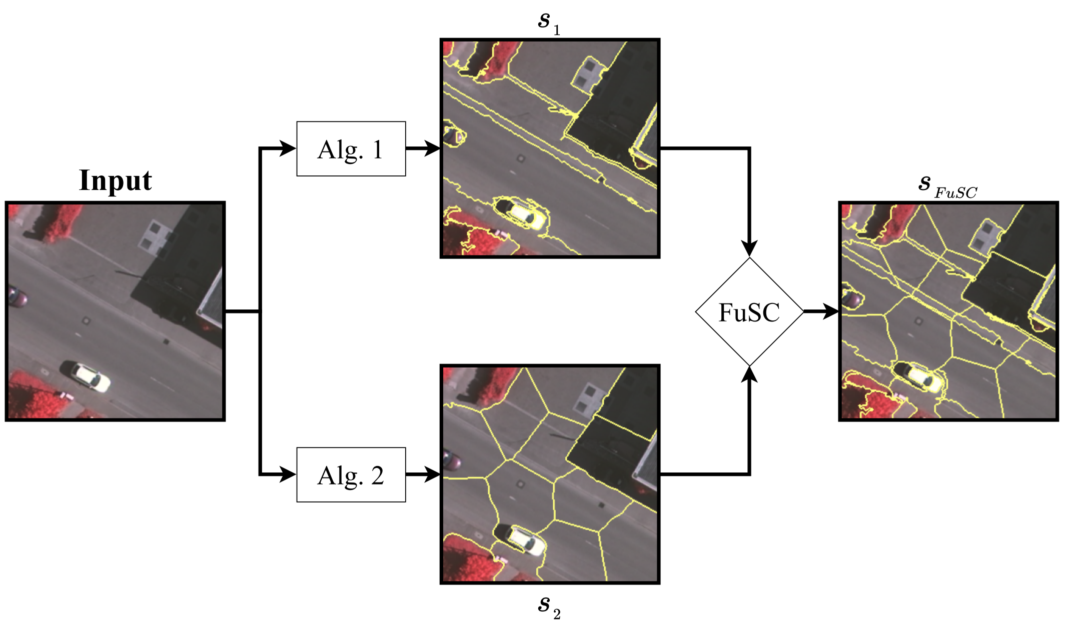


Figure 4.12: The figure shows a toy example illustrating the workflow to merge two different superpixel segmentations. First, the input image x is processed by 2 different superpixel segmentation algorithms (Alg. 1 and Alg. 2). Then the generated segmentations s_1 and s_2 are merged into the final segmentation s_{FuSC} using the merging procedure described in Algorithm 2.

4.3.1 Fusing Superpixels for Semantic Consistency

All SPS algorithms share the same main goals: generate homogeneous areas and respect borders among different objects. SPS algorithms aim to minimize the intracluster/intrasegment variance while maximizing the inter-cluster/intersegment variance.

Different SPS algorithms use distinct procedures and premises to produce the final segmentation. Since the generation process is different from one to another, each over-segmentation produced fails and succeeds in distinct ways to achieve the intended representation.

The FuSC procedure fuses input segmentations from multiple types of superpixel generation algorithms. Using distinct families of SPS methods allows FuSC to take advantage of the different generation characteristics, enhancing the advantages and mitigating the disadvantages of each method. Figure 4.13 shows an example of how each single SPS represents ground truth and compares to FuSC segmentation. We can observe through the qualitative result that the FuSC improves the representation of the mask concerning the ground truth.

The class with the highest pixel count is assigned to the entire segment to generate ground truth that perfectly overlaps with all superpixels.

Figure 4.12 illustrates the merging of two different superpixel segmentations. As shown in the figure, the final SPS respects both segmentations' borders, and each segment represents better the underlying region. FuSC is agnostic to the SPS algorithm, being applicable to any set of distinct superpixel algorithms. However, in practice, using more than two algorithms yields exceedingly small segments, motivating our experiments to focus only on pairs of algorithms.

Algorithm 2 Pseudo-algorithm for the FuSC procedure and the auxiliary procedure of joining segmentations. The complexity of the procedure is pseudo-polynomial with respect to the number of pixels in the image and the minimum size of the superpixel (Appendix A).

```
Require: scores                                                  ▷ pixel-wise array
Require: segments                                                ▷ list of segments
 1: procedure JOIN_SEGMENTATIONS($seg1, seg2$)
 2:     joint = []
 3:     for $s1 \in seg1$ do                 ▷ Selecting s2 ∈ seg2 where s2 ∩ s1 ≠ ∅
 4:         for s2 ∈ seg2.OVERLAP_SEGMENTS(s1) do
 5:             overlap_area = $s1 \cap s2$
 6:             joint.ADD_NEW_SEGMENT(overlap_area)
 7:         end for
 8:     end for            ▷ secure that the labels are connected and sequential
 9:     $joint \leftarrow connected\_sequential\_labels(joint)$
10:     return joint
11: end procedure
12:
13: procedure FuSC($seg1, seg2$)
14:     joint = JOIN_SEGMENTATIONS(seg1, seg2)
15:     for $s \in joint$ do
16:         if $s.size < min\_size$ then
17:             closest = CLOSEST_NEIGHBOR(s, joint)
18:             joint = MERGE_SEGMENTS(joint, s, closest)
19:         end if
20:     end for
21:     return joint
22: end procedure
```

The first step of the fusion procedure is to generate unique segments by superposing two different segmentations. This procedure can be executed by running the following steps:

1. generates a new segmentation from the intersection of the input segmentations;

2. relabel the masks securing that the mask is sequential, that each label is used once and represents a connected area.

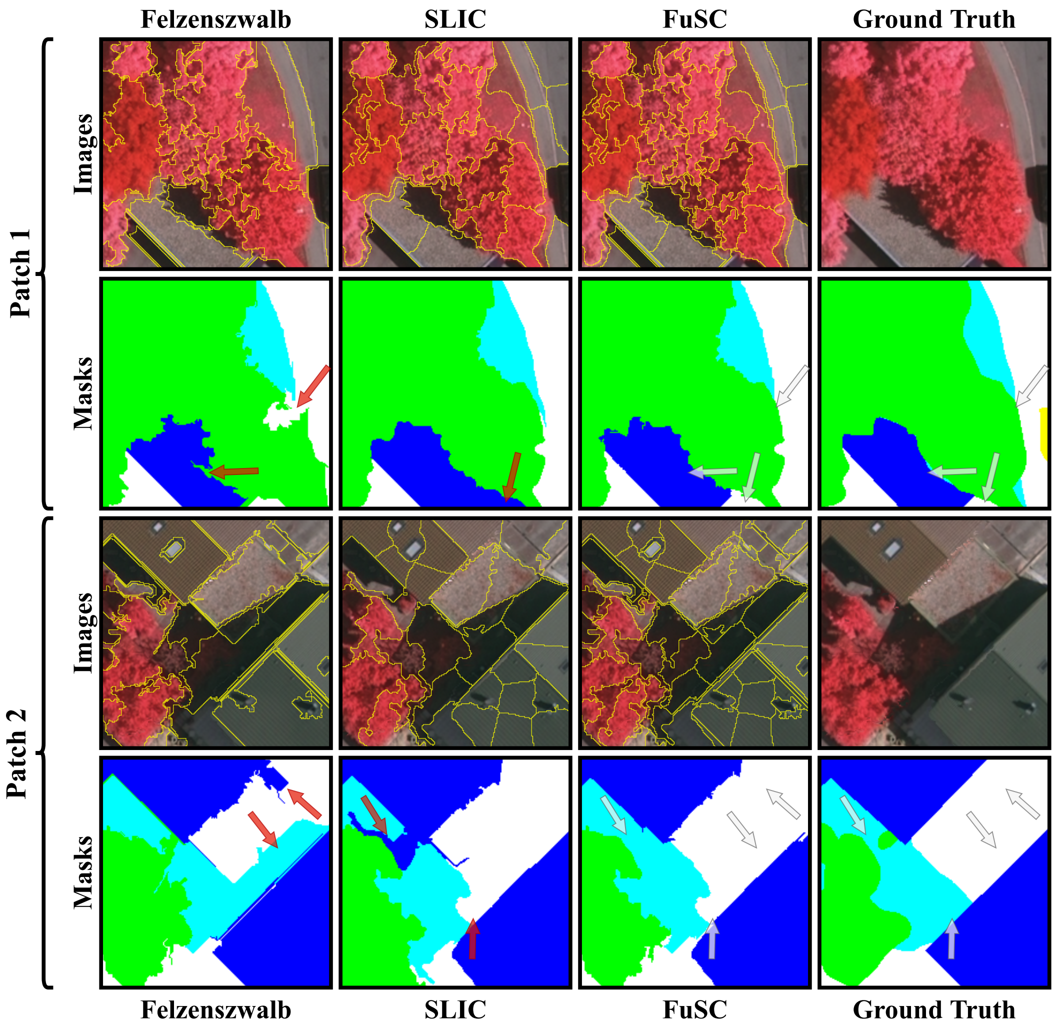


Figure 4.13: The figure shows the comparison of the resulting segmentation from two SPS algorithms (Felzenszwalb and SLIC) and our proposed fusion algorithm, FuSC. The first and third rows show the input image superimposed with the superpixel segments and the second and fourth rows depict the closer class fit of each segment according to the real labels. Red arrows indicate areas where class boundaries failed when using one single SPS algorithm, while gray arrows point to these same regions fixed after applying the FuSC algorithm.

Both steps described above have theoretical complexity linear on the number of pixels of the image (O(n) for n the number of pixels). This initial merging procedure is prone to produce some extremely small segments.

To tackle the unwanted small segments side-effect, we use the Mahalanobis distance (MAHALANOBIS, 1936) to fuse the closest neighbor segments until there are no more segments below the specified pixel minimum size. FuSC is detailed in Algorithm 2.

The theoretical complexity of the FuSC procedure is $O(n \times minimum\_size^2)$, for n the number of pixels and $minimum\_size$ the parameter of the merge procedure. The code in python with the depiction of the complexity analysis can be seen in Appendix A.

4.4 Proposed Methods and Research Questions

OpenGMM method proposed in Section 4.1 is an extension of the baseline methods: OpenPCS (Oliveira et al., 2021) and OpenPCS++ (Martinez et al., 2021). The proposed approach is an improvement of the previous methods replacing the data representation strategy from the PCA with a GMM. The quantitative results of OpenGMM compared to the baseline methods results described in Section 7.1. These results show that OpenGMM can improve the known benchmarks and answer $\mathcal{RQ}_1$.

Section 4.2 described CoReSeg a novel end-to-end fully convolutional approach to OSS. The quantitative results for CoReSeg compared to OpenGMM and the baseline methods are presented in Section 7.2, and the obtained results were even better than the ones obtained by OpenGMM. Section 7.4 shows the results produced by all methods. CoReSeg produced better semantic consistency for both the Vaihingen and the Potsdam datasets. This set of better quantitative results and better semantic consistency produced answers $\mathcal{RQ}_2$.

The last proposed approach, detailed in Section 4.3, is a general post-processing technique that can be used with any OSS method. The quantitative results for the post-processing are presented in Section 7.3 and produced better results in thirty-nine in forty tested scenarios. The qualitative results for all methods are presented in Section 7.4 and show continuous improvements in semantic consistency as the methods were presented, with the best overall results obtained post-processing CoReSeg's results. Therefore, the proposed post-processing can be used with any OSS method and answers the last research question $\mathcal{RQ}_3$.

5 Experimental Setup

All experiments used PyTorch version 1.13 framework (Paszke et al., 2019) to implement neural network models and backbones, with an NVIDIA Titan X with 12GB of memory. All models fit on a single graphics card, filling between 10 GB and 11 GB of memory.

SPS algorithms were implemented using the *scikit-learn*[1] and *scikit-image* (van der Walt et al., 2014) libraries. The official implementations for FuSC, OpenGMM, and CoReSeg are publicly available[2] encouraging reproducibility.

This chapter is organized as follows: Section 5.1 describes the used backbones and presents the closed-set results with and without the attention mechanism; Section 5.2 describes the basic training procedure and hyperparameters for the CoReSeg and OpenGMM; Section 5.3 presents the tested datasets and its key characteristics; Section 5.4 describes the leave one class out (LOCO) protocol used to emulate an open-set scenario and the evaluation metrics; and Section 5.5 describes used configurations to generate the superpixels for post-processing.

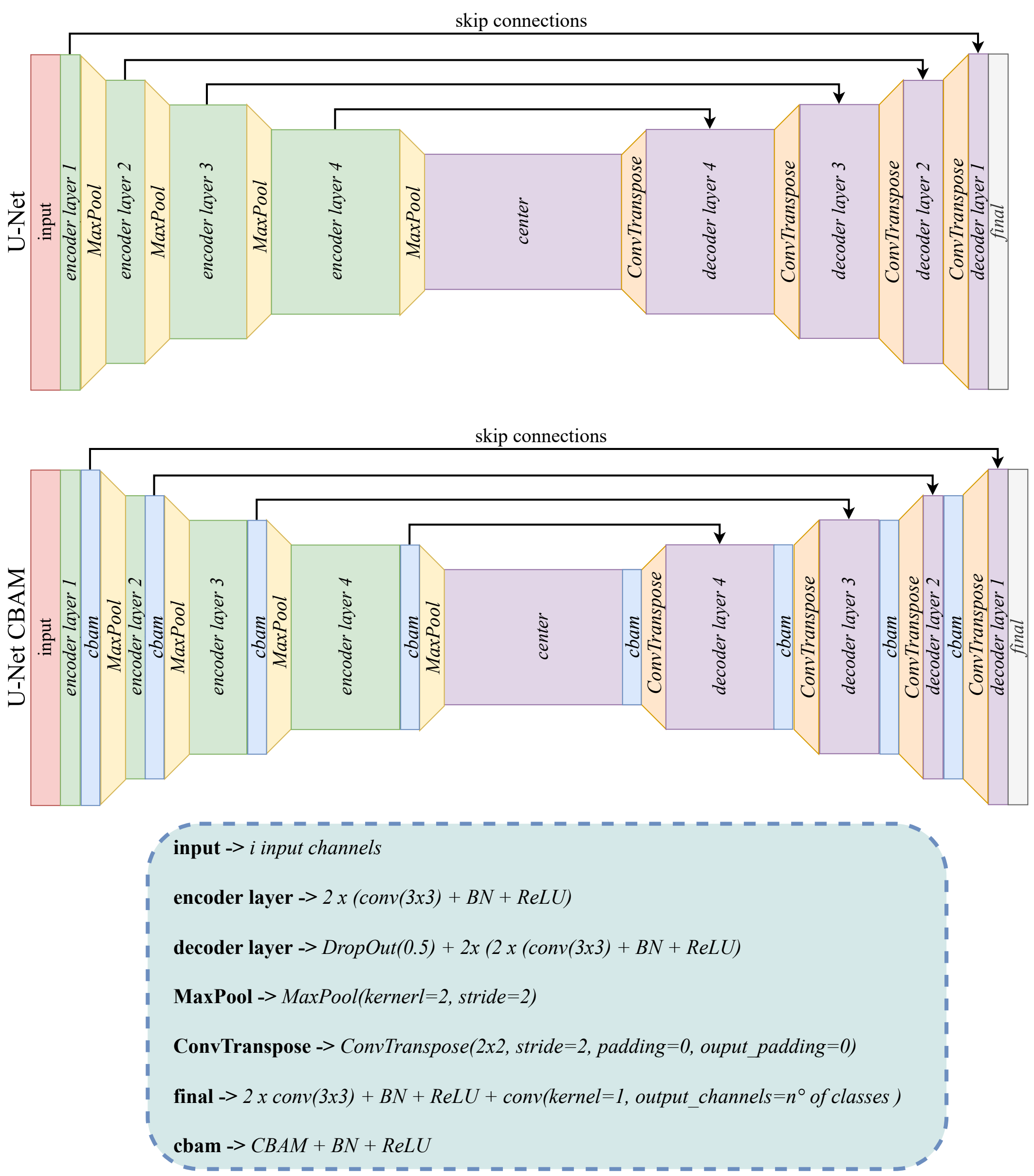
5.1 Closed-set Backbones

Both OpenGMM and CoReSeg use closed-set backbones for semantic segmentation, and the three used backbones are: DN-121 (Huang et al., 2017), WRN-50 (Zagoruyko and Komodakis, 2016) and U-net (Ronneberger et al., 2015). Figures 5.2 and 5.1 show the architectural schematics of all used closed-set models.

All closed-set backbones were trained for 600 epochs for the Potsdam dataset and 1200 epochs for the other datasets. Other training schemas are stochastic batch selection; learning rate of $1x10^{-3}$; weight decay of $2x10^{-1}$ after every 1/3 of the epochs; and batch balanced cross-entropy loss. Also, the Adam solver (Kingma and Ba, 2014) was the used optimizer for all closed-set models.

[1]`https://scikit-learn.org/`

[2]`https://github.com/iannunes`



input -> *i input channels*

encoder layer -> *2 x (conv(3x3) + BN + ReLU)*

decoder layer -> *DropOut(0.5) + 2x (2 x (conv(3x3) + BN + ReLU)*

MaxPool -> *MaxPool(kernerl=2, stride=2)*

ConvTranspose -> *ConvTranspose(2x2, stride=2, padding=0, ouput_padding=0)*

final -> *2 x conv(3x3) + BN + ReLU + conv(kernel=1, output_channels=n° of classes)*

cbam -> *CBAM + BN + ReLU*

Figure 5.1: The figure describes the U-Net (Ronneberger et al., 2015) used as backbones and also shows the variation adding the CBAM (Woo et al., 2018) attention mechanism. The U-Net with CBAM model is the same standard U-Net with the attention mechanism added after the blocks as shown in the figure. All convolution layers use default padding and stride equal to one.

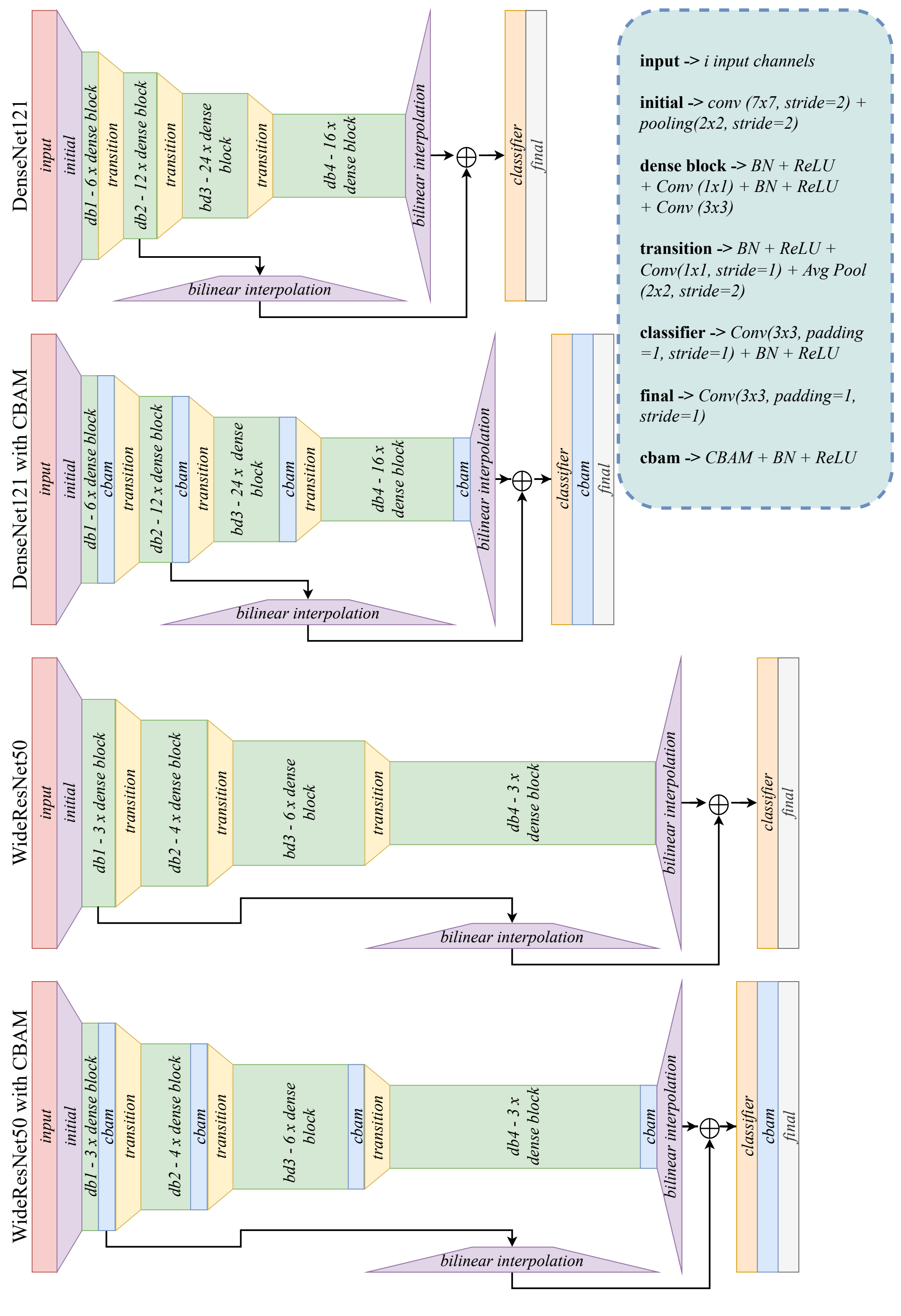


Figure 5.2: The figure describes the two used backbones: DN-121 (Zagoruyko and Komodakis, 2016) and WRN-50 (Huang et al., 2017); and also shows the variations adding the CBAM (Woo et al., 2018) attention mechanism to both models. WideResNet and DenseNet use the same basic *dense block*. While DenseNet adds more sequential blocks making the network deeper, the WideResNet uses fewer blocks but increases the number of channels. For our experiments, WideResNet uses twice more channels as DenseNet. The number of *dense blocks* used are shown inside the layer block.

All closed-set backbones were trained from scratch since the available pre-trained backbones were trained using 3-band RGB images and are heavily distinct in both the image domain and the number of input bands.

Tables 5.1 and 5.2 shows the achieved results for the closed-set methods in all five tested scenarios with and without the CBAM attention mechanism. The best average results are in bold and we can see that the closed-set segmentation with attention outperforms the results of the scenarios without attention in 21 of 24 average metrics, 9 out of 12 for the Vaihingen dataset and all 12 for the Potsdam dataset.

UUC	$\mathcal{A}$	**DN-121**				**WRN-50**				**U-net**			
		j	a	b	κ	j	a	b	κ	j	a	b	κ
0	-	69	83	84	75	69	82	83	73	75	87	83	80
1	-	59	80	79	70	60	80	78	70	63	83	72	75
2	-	72	89	87	85	70	89	85	84	73	93	79	89
3	-	66	89	84	79	64	84	82	77	70	89	77	84
4	-	69	82	81	76	67	80	80	74	73	84	84	79
Avg	-	67	**85**	83	77	66	83	81	76	71	**87**	79	**81**
0	✓	72	84	85	76	69	82	82	74	74	84	85	73
1	✓	62	80	80	71	61	80	79	70	66	82	82	73
2	✓	72	90	86	85	71	89	86	84	78	91	90	87
3	✓	66	86	85	79	67	86	84	79	73	88	87	82
4	✓	69	82	81	75	67	80	80	74	70	82	82	77
Avg	✓	**68**	84	**83**	**77**	**67**	**83**	**82**	**76**	**72**	85	**85**	78

Table 5.1: The table shows the results for the closed-set models used as backbones for the OSS task for the Vaihingen dataset. The *UUC* column shows the hidden class used to emulate an open-set scenario using the LOCO protocol. The $\mathcal{A}$ column indicates the use of the CBAM attention mechanism. The evaluations used four metrics: j for the mean intersection over union (Jaccard distance); a for the global accuracy; b for the balanced accuracy; and κ is Cohen's kappa. For the Vaihingen dataset, five emulated open scenarios using as UUCs: 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; 4 - car. The best-achieved values are in bold. In this table, all results are multiplied by 10^2.

UUC	$\mathcal{A}$	DN-121				WRN-50				U-net			
		j	a	b	κ	j	a	b	κ	j	a	b	κ
0	-	61	76	79	64	55	73	78	61	71	82	85	73
1	-	60	77	79	66	61	76	78	64	68	82	83	73
2	-	62	82	84	73	60	79	80	69	73	86	88	80
3	-	65	82	83	73	56	78	79	67	73	86	87	78
4	-	59	74	75	65	53	70	71	59	64	78	78	71
Avg	-	62	78	80	68	57	75	77	64	70	83	84	75
0	✓	65	77	81	66	59	74	79	62	71	83	85	74
1	✓	65	80	82	70	62	78	79	67	68	82	84	72
2	✓	72	86	87	79	67	83	85	75	76	88	89	83
3	✓	71	85	87	78	62	81	83	72	75	87	88	80
4	✓	63	78	77	70	55	71	72	61	64	78	77	70
Avg	✓	**67**	**81**	**83**	**73**	**61**	**78**	**80**	**67**	**71**	**84**	**85**	**76**

Table 5.2: The table shows the results for the closed-set models used as backbones for the OSS task for the Potsdam dataset. The *UUC* column shows the hidden class used to emulate an open-set scenario using the LOCO protocol. The $\mathcal{A}$ column indicates the use of the CBAM attention mechanism. The evaluations used four metrics: j for the mean intersection over union (Jaccard distance); a for the global accuracy; b for the balanced accuracy; and κ is Cohen's kappa. For the Vaihingen dataset, five emulated open scenarios using as UUCs: 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; 4 - car. The best-achieved values are in bold. In this table, all results are multiplied by 10^2.

5.2 Hyperparameters

For further analysis, all architectural choices, hyperparameters, and methods are available as per the codes and configuration files on the project website. The main ones are described in the two following sections.

5.2.1 OpenGMM

To the extent of this work, we used the GMM implementation of *scikit-learn*[3]. We configured the number of components and the regularization added to the diagonal of the covariance matrix to compute the Gaussian Mixture of Models.

[3]`https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html`

In our exploratory experiments, we used five different numbers of mixture components (2, 4, 8, 16, and 32) as parameters to define the best value applicable in our case. We found that any value above 4 configured as the number of components did not change the final result for the produced OSS. As the GMM is computed faster and with a smaller number of components, we defined the value of 4 as the production parameter.

The *reg_covar* parameter was set to 1×10^{-2}, and this was the smallest regularization value that allowed the GMM algorithm to execute without errors for all hidden scenarios. This regularization value is added to the diagonal of the covariance matrix ensuring that the matrix is all positive.

5.2.2
CoReSeg

Some used hyperparameters are common despite the architectural variations presented in Chapter 4. We defined an initial learning rate of 1×10^{-3}, with a weight decay of 0.2 for every one-third of the total epochs. We also used a stochastic batch selection of size 2 for Vaihingen and 1 for Potsdam and executed the network for forty epochs for Vaihingen and thirty for Potsdam. We used the Adam solver (Kingma and Ba, 2014) as the optimizer to minimize the L1 loss function in the reconstruction module of CoReSeg.

5.3
Vaihingen and Potsdam Datasets

This section describes the used datasets with their available bands or channels, resolutions, known classes, and train-test-validation split. Table 5.3 shows the key characteristics of the datasets.

Feature	**Vaihingen**	**Potsdam**
type of sensor	optical (Near infrared, Red, Green)	
digital surface mode band	Yes	
number of classes	6	
multi-temporal	no	
timestamps	1	
number of bands	4	
KKCs	impervious surfaces, building, low vegetation, high vegetation, car	
KUCs	miscellaneous, segmentation boundaries	
spatial resolution	9 cm	5 cm
size (pixels)	248.798.532	1.368.000.000
labeled pixels	248.798.532	1.368.000.000

Table 5.3: Summary of key characteristics of the datasets.

The experiments used the International Society for Photogrammetry and Remote Sensing (ISPRS) 2D Semantic Labeling[4] datasets of Vaihingen and Potsdam. Both datasets were previously used in OSS (da Silva et al., 2020; Oliveira et al., 2021; Nunes et al., 2022b). Vaihingen images present a 9cm/pixel spatial resolution, varying from 2000 to 2500 pixels per axis, while Potsdam samples have a 5cm/pixel spatial resolution and 6000×6000 pixels each. Both datasets are labeled with the same classes listed in Table 5.4, divided into 5 KKCs: impervious surfaces, buildings, low vegetation, high vegetation, car; and 2 KUCs: segmentation boundaries between objects and miscellaneous. Among KUCs, the miscellaneous class is composed mostly of areas presenting image acquisition noise and objects unimportant to practical remote sensing applications, motivating its removal from the experimental procedure. The experiments used the same four bands employed in previous works on OSS (Oliveira et al., 2021; Nunes et al., 2022b), namely IR-R-G-nDSM.

[4]`https://www.isprs.org/education/benchmarks/UrbanSemLab/default.aspx`

Id	Class
1	Impervious Surfaces
2	Building
3	Low Vegetation
4	High Vegetation
5	Car
6	Miscellaneous
7	Segmentation boundaries

Table 5.4: The table shows all mapped classes for Vaihingen and Potsdam datasets. For this work classes: 6 (miscellaneous) and 7 (segmentation boundaries) are set as unknown.

Table 5.5 shows the dataset's divisions used during this work. The datasets were separated into three sets of images each: training, validating, and testing. The numbers presented in Table 5.5 are the reference used in the original file names downloaded from the official repository of each dataset.

Image set	Vaihingen	Potsdam
Train	1, 3, 5, 7, 13, 17, 21, 26, 32, and 37	2_10, 2_13, 2_14, 3_10, 3_12, 3_13, 3_14, 4_11, 4_12, 4_13, 4_14, 4_15, 5_10, 5_12, 5_13, 5_14, 5_15, 6_8, 6_9, 6_10, 6_11, 6_12, 6_13, 6_15, 7_7, 7_9, 7_11, 7_12, and 7_13
Validation	23	3_11 and 6_14
Test	11, 15, 28, 30, and 34	2_11, 2_12, 4_10, 5_11, 6_7, 7_8 and 7_10

Table 5.5: The table shows the selected patches according to their original nomenclature. Each dataset was divided into three divisions: train, validation, and test.

5.4 Evaluation Protocol

The used datasets were built to perform closed-set semantic segmentation. The LOCO protocol used by Oliveira et al. (2021) was also applied in this work to emulate open-set scenarios. The LOCO protocol splits the known classes and selects a subset of them to be ignored during training, allowing the evaluation of open-set methods on the chosen hidden classes. This protocol allows both the computation of overall performance and class-by-class metrics. Figure 5.3 shows an example of the use of the protocol, presenting the input image on the left, its original ground truth, and all five emulated open-set scenarios on the right.

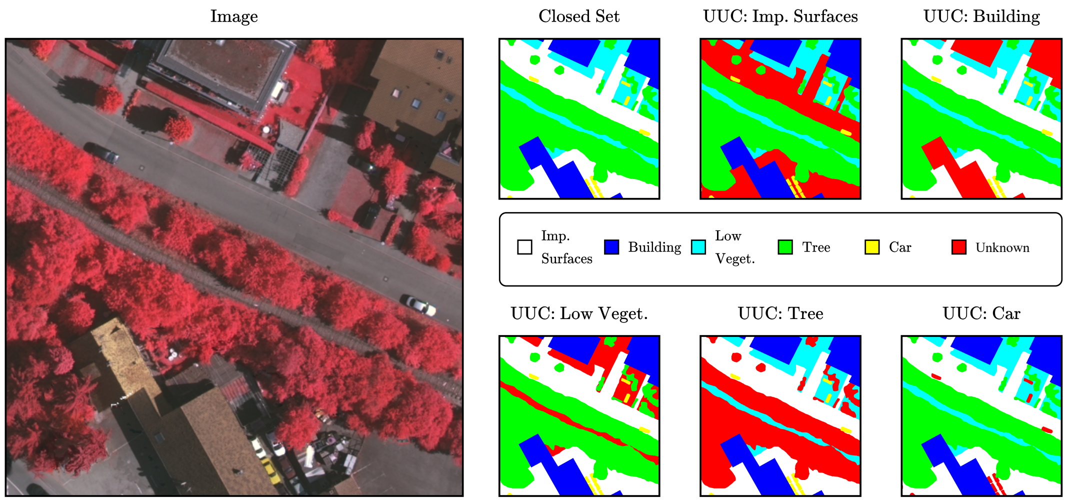


Figure 5.3: An example of LOCO protocol showing a small patch extracted from the Vaihingen dataset as the input image. On the right, one patch presents the original closed-set classes, and the other five patches show the generated labels with the LOCO protocol according to the legend.

To ensure that only information about the KKCs is available for training, we only backpropagate the loss of pixels from known classes, ignoring the background, borders, miscellaneous, and unknown classes.

5.4.1 Evaluation Metrics

For the quantitative assessment of produced open-set segmentations, we used the Receiver Operating Characteristic (ROC) curve and the Area Under the ROC (AUROC) curve to compare the results of the different methods in a threshold-independent manner. The AUROC also provides evidence of the overall performance of each method.

The ROC curve evaluates the entire range of thresholds on a plot of True Positive Rate (TPR) vs. False Positive Rate (FPR), showing the threshold-dependent relationship between the FPR and TPR. The AUROC evaluates the model performance trade-off between known and unknown conditions. To compute the AUROC, KKCs are treated as positives and the UUCs as negatives. The AUROC values range between 0 and 1, and a value of 1 means a perfect detection of the known classes compared to the unknown classes.

The FPR, also called specificity, measures the fraction of unknown classes misclassified as one of the known classes. A high FPR indicates that unknown conditions are conflated with some known class. The TPR, also called sensitivity, is the probability of a positive pixel testing positive. The following equations are used to compute TPR and FPR:

$$TPR = TP/(TP + FN) \tag{5-1}$$

and

$$FPR = FP/(FP + TN), \tag{5-2}$$

where TP, TN, FN, and FP refer to True Positives, True Negatives, False Negatives, and False Positives, respectively.

The AUROC is a suitable measure for evaluating the performance of proposed methods against baselines across the entire threshold spectrum. The definition of a single threshold can result in an incorrect assessment of the performance of KKCs and UUCs. Setting a restrictive threshold may underestimate the UUCs. On the other hand, a broad threshold may overestimate UUCs to the detriment of KKCs.

AUROC: The ROC curve describes the threshold-dependent relationship between the FPR and TPR, and the area under the ROC curve is a metric to evaluate the model performance trade-off between known conditions and unknown conditions. Here, all the known conditions are treated as positives, while the unknown conditions or open classes are treated as negatives. A 100% AUROC means a perfect detection of the known classes compared to the unknown classes.

5.5 Superpixel Configurations

This section presents all superpixels configurations used by the post-processing proposed in Section 4.3 to run the tests of this work with its results in Chapters 6 and 7. An important observation is that in general the hyperparameters used for the single SPS generation algorithms produce smaller superpixels than the base single SPS algorithms merged with FuSC.

We did preliminary exploratory experiments using a grid search to determine the parameters for each superpixel algorithm. In these tests, we used the average size of the produced superpixels to choose the configurations presented in this section. We do not present the results of these experiments as the selected superpixel algorithms are widely used and have plenty of work using them. Table 5.6 shows the average size of produced superpixels for the Vaihingen dataset.

In the list below, FZ stands for the Felzenszwalb algorithm (Felzenszwalb and Huttenlocher, 2004), QS stands for the Quickshift (Vedaldi and Soatto, 2008) algorithm, and SLIC stands for the method with the same (Achanta et al., 2012):

1. Single SPS algorithms:

   (a) Felzenszwalb algorithm:

      i. *fz01*: FZ (scale: 50, σ: 0.5, min_size: 50)
      ii. *fz02*: FZ (scale: 100, σ: 0.5, min_size: 50)
      iii. *fz03*: FZ (scale: 200, σ: 0.5, min_size: 50)
      iv. *fz04*: FZ (scale: 400, σ: 0.5, min_size: 50)
      v. *fz05*: FZ (scale: 50, σ: 0.5, min_size: 100)
      vi. *fz06*: FZ (scale: 100, σ: 0.5, min_size: 100)

   (b) SLIC algorithm:

      i. *slic01*: SLIC (n_segments: n_pixels÷1400, compactness: 5, σ: 1)
      ii. *slic02*: SLIC (n_segments: n_pixels÷700, compactness: 5, σ: 1)
      iii. *slic03*: SLIC (n_segments: n_pixels÷ 500, compactness: 5, σ: 1)
      iv. *slic04*: SLIC (n_segments: n_pixels÷350, compactness: 5, σ: 1)
      v. *slic05*: SLIC (n_segments: n_pixels÷230, compactness: 5, σ: 1)
      vi. *slic06*: SLIC (n_segments: n_pixels÷170, compactness: 5, σ: 1)

   (c) Quickshift algorithm:

      i. *quick01*: QS (kernel_size: 2, max_dist: 50, ratio: 0.5)
      ii. *quick02*: QS (kernel_size: 3, max_dist: 50, ratio: 0.5)
      iii. *quick03*: QS (kernel_size: 4, max_dist: 50, ratio: 0.5)
      iv. *quick04*: QS (kernel_size: 5, max_dist: 50, ratio: 0.5)

2. FuSC:

   (a) combine Felzenszwalb with SLIC:

      i. fz_slic01:

         – SLIC (n_segments: n_pixels÷2000, compactness: 5, σ: 1)
         – FZ (scale: 200, σ: 0.7, min_size: 200)

ii. fz_slic02:
   - SLIC (n_segments: n_pixels÷1500, compactness: 5, σ: 1)
   - FZ (scale: 100, σ: 0.7, min_size: 150)

iii. fz_slic03:
   - SLIC (n_segments: n_pixels÷1000, compactness: 5, σ: 1)
   - FZ (scale: 100, σ: 0.7, min_size: 150)

iv. fz_slic04:
   - SLIC (n_segments: n_pixels÷500, compactness: 5, σ: 1)
   - FZ (scale: 50, σ: 0.7, min_size: 100)

(b) combine Felzenszwalb with Quickshift:

i. fz_quick01:
   - QS (kernel_size: 2, max_dist: 50, ratio: 0.5)
   - FZ (scale: 200, σ: 0.7, min_size: 200)

ii. fz_quick02:
   - QS (kernel_size: 3, max_dist: 50, ratio: 0.5)
   - FZ (scale: 200, σ: 0.7, min_size: 200)

iii. fz_quick03:
   - QS (kernel_size: 4, max_dist: 50, ratio: 0.5)
   - FZ (scale: 200, σ: 0.7, min_size: 200)

iv. fz_quick04:
   - QS (kernel_size: 5, max_dist: 50, ratio: 0.5)
   - FZ (scale: 200, σ: 0.7, min_size: 200)

(c) combine SLIC with Quickshift:

i. quick_slic01:
   - QS (kernel_size: 5, max_dist: 50, ratio: 0.5)
   - SLIC (n_segments: n_pixels÷2000, compactness: 5, σ: 1)

ii. quick_slic02:
   - QS (kernel_size: 5, max_dist: 50, ratio: 0.5)
   - SLIC (n_segments: n_pixels÷1500, compactness: 5, σ: 1)

iii. quick_slic03:
   - QS (kernel_size: 5, max_dist: 50, ratio: 0.5)
   - SLIC (n_segments: n_pixels÷1000, compactness: 5, σ: 1)

iv. quick_slic04:
   - QS (kernel_size: 5, max_dist: 50, ratio: 0.5)
   - SLIC (n_segments: n_pixels÷500, compactness: 5, σ: 1)

SPS configuration	average pixels/superpixel
fz01	322
fz02	491
fz03	849
fz04	1469
fz05	582
fz06	780
slic01	2959
slic02	1317
slic03	897
slic04	639
slic05	366
slic06	261
quick01	367
quick02	824
quick03	1471
quick04	2272
fz_quick01	157
fz_quick02	254
fz_quick03	344
fz_quick04	423
fz_slic01	630
fz_slic02	554
fz_slic03	462
fz_slic04	306
quick_slic01	622
quick_slic02	545
quick_slic03	454
quick_slic04	301

Table 5.6: The table presents the average count of pixels per superpixel for the Vaihingen dataset.

6
Ablation

This chapter will present some experiments executed to define the best hyperparameters for CoReSeg and for the superpixel post-processing. These experiments are also evidence of better FuSC performance over single SPS algorithms.

The ablation chapter will not cover OpenGMM as while testing, the results achieved were similar for all tested parameter combinations. It is worth mentioning that for OpenGMM, we changed the number of components to generate de mixture of Gaussians between 4, 8, and 16, but the experiments yielded the same AUROC. Since the results were the same and fewer components were executed faster, we used 4 Gaussian components in our final experiments.

Vaihingen and Potsdam have many similarities, but Vaihingen is smaller and executes each training round relatively faster. The possibility of running more tests made the Vaihingen dataset the natural choice for the ablation. All results presented in this chapter are for the Vaihingen dataset.

This chapter presents two ablation sets of experiments in detail. Section 6.1 details the tests with parameter and model architecture variations for the CoReSeg, and Section 6.2 shows the post-processing with single SPS algorithms and FuSC with different parameter configurations.

6.1
CoReSeg

We choose to test the model with some variations on hyperparameters to select the model for the other datasets. For the CoReSeg, we ran preliminary tests to define the number of epochs testing from 20 to 100 in steps of size 10, observing better results training for 40 epochs. We established 40 as the number of training epochs for both the ablation and final tests. As a final refinement, our train procedure selects the model with the highest AUC in the validation set and runs extra five epochs on this model.

There are two variations of the model: first, called *full* model in Table 6.1, the input for the reconstruction encoder blocks is the concatenation of the previous layer output with raw *skip connections* from the closed-set encoder and with the same *skip connections* conditioned by the conditioning encoders as explained in Section 4.2, second, called *base* in Table 6.1, the input is formed only by the output of the previous layer and the conditioned *skip connections*. Column *Skip* in Table 6.1 indicates the CoReSeg variation used.

We used Adam optimizer (Kingma and Ba, 2014) with two different initial learning rates 1×10^{-3} and 5×10^{-4}, both learning rates decaying every 1/3 of the epochs with 0.2 as a multiplicative factor.

We also tested adding an extra convolutional block just before the final layer of the model. This block was composed of a Convolutional followed by Normalization and Activation layers. The column *FC* of Table 6.1 shows these configurations.

The most direct observation is that the extra final convolutional block made the model perform worse when compared to the model without this block in all cases. We tested with different learning rates (LR) and achieved mixed results using LR = 0.001 and LR = 0.0005. Training using LR = 0.001 performed better in more cases, and the best model also used LR = 0.001. The LR set as 0.001 was better in all cases for *CoReSeg+Att*. It is worth mentioning that the result was neither final nor conclusive. The reasoning proposed by Smith et al. (2017) stated that changing only the LR without changing the batch size or increasing the number of epochs is not the most suitable, which may explain why the results were not conclusive. Changing LR may require extra fine-tuning in batch size may also be necessary, and according to Smith et al. (2017) bigger batch sizes converge faster.

We also tested two distinct model variations, as detailed in Section 4.2.3 and called *full, base* model, for each model-backbone configuration. Both variations performed close for the best model (CoReSeg+Att), with the variation called *full* model performing slightly better. For the two other tested model configurations without attention module for CoReSeg, the *base* configuration performed better. The CBAM attention mechanism improved the results used in the backbone and the CoReSeg model alone or together.

The *base* model has roughly 72 million parameters, and the *full* model has roughly 102 million parameters. Since the *full* model showed the best results, and due to the computational time needed, we decided to use only the *full* model configuration to run the complete set of tests presented in Chapter 7. The needed running time was the key constraint in not running both model variations.

The ablation results for the *base* were promising since the model is smaller and faster. Further analysis is needed to understand better in which scenarios it should be better.

MA	BA	Skip	FC	LR	UUCs					Avg. AUROC
					0	1	2	3	4	
✓	✓	full	✗	.001	.84	**.90**	.69	.72	**.72**	**.774**
✓	✓	full	✗	.0005	**.87**	.77	.69	.75	.70	.753
✓	✓	full	✓	.001	.84	.85	.65	**.79**	.62	.751
✓	✓	full	✓	.0005	.84	.79	.68	.73	.66	.739
✓	✓	base	✗	.001	.86	.87	**.71**	.74	.67	.769
✓	✓	base	✗	.0005	.86	.84	.67	**.79**	.64	.758
✓	✓	base	✓	.001	.75	.79	.65	.72	.65	.712
✓	✓	base	✓	.0005	.84	.58	.67	.67	.71	.693
✓	✓	*Average*			***.84***	***.80***	*.68*	***.74***	*.67*	***.744***
-	-	full	✗	.001	**.88**	.82	.68	.66	.68	.742
-	-	full	✗	.0005	.82	**.91**	.67	**.75**	.59	.746
-	-	full	✓	.001	.78	.66	.68	.69	.73	.708
-	-	full	✓	.0005	.73	.55	.69	.70	.72	.679
-	-	base	✗	.001	.83	.81	.68	.74	.66	.744
-	-	base	✗	.0005	.87	.78	**.70**	**.75**	.70	**.758**
-	-	base	✓	.001	.78	.73	.67	.67	.74	.717
-	-	base	✓	.0005	.79	.65	.69	.65	**.77**	.712
-	-	*Average*			*.81*	*.74*	*.68*	*.70*	***.70***	*.726*
-	✓	full	✗	.001	**.89**	.77	.69	.74	.63	.742
-	✓	full	✗	.0005	.84	**.78**	.64	**.83**	.59	.733
-	✓	full	✓	.001	.80	.63	.64	.68	.73	.696
-	✓	full	✓	.0005	.77	.55	.72	.67	.70	.682
-	✓	base	✗	.001	.85	**.78**	.69	.78	.64	.748
-	✓	base	✗	.0005	**.89**	.77	.70	.79	.67	**.765**
-	✓	base	✓	.001	.77	.60	.69	.63	.68	.674
-	✓	base	✓	.0005	.79	.65	**.73**	.71	**.74**	.724
-	✓	*Average*			*.82*	*.69*	***.69***	*.73*	*.67*	*.721*

Table 6.1: The table shows results for the CoReSeg method and the variations proposed in section 4.2 tested only with the Vaihingen dataset, varying 3 different parameters (*Skip*, *FC*, and *LR*). The *Skip* column indicates the skip connection strategy, using only before conditioning or concatenating with the conditioned tensor to feed the reconstruction decoder. The *FC* column indicates if a final convolutional layer is added, and the *LR* column indicates the initial learning rate for the reconstruction decoder. Also, the *MA* column indicates if CoReSeg is using the CBAM attention mechanism in the reconstruction module, and the *BA* column if the U-net backbone is using the CBAM attention mechanism. In bold are the best results for each combination of method and backbone, the darkest gray rows are compared between themselves. The UUC numbers are respectively: 0 - impervious surfaces; 1 - building; 2 - high vegetation; 3 - low vegetation; and 4 - car.

6.2 Superpixel Post-processing

This section presents the results of the initial tests executed on the Vaihingen dataset. These tests also defined the hyperparameters used to perform the final tests shown in Chapter 7. For this ablation, we selected the two methods presented in this work to run the tests, CoReSeg and OpenGMM. For the OpenGMM method, we used the two backbones with higher closed-set accuracy.

A total of seventy-two different tests were executed and presented in the Tables 6.2, 6.3 and 6.4. Also, to evaluate post-processing performance using the single superpixel algorithm and FuSC, we tested 24 different settings detailed in Section 5.5.

Table 6.2 shows the results for CoReSeg. FuSC configuration called *fz_slic4* obtained the best average performance. Post-processing using this configuration improved the AUROC by 0.023 or 2.7%, with 24 of the 28 tested FuSC configurations improving the base results. The worst results came from the settings with the highest pixel count per segment. With larger superpixels, the segmentation lost its ability to represent the underlying image.

Table 6.3 shows the results for OpenGMM with DN-121 as the backbone. The post-processing with *fz2* configuration improved the base results by 0.10 or 1.3%. Results achieved by post-processing this family of test scenarios presented the worst performance since only 50% of the superpixel configurations improved the base results. The *fz2* Felzenszwalb's single SPS configuration achieved the best results followed by three other configurations using the same algorithm. The use of SLIC and QuickShift single SPS or merged with FuSC delivered the worst results, especially with the larger superpixels.

Table 6.4 shows the results for OpenGMM with WRN-50 as the backbone. Post-processing with *fz2* configuration improved the base results by 0.14 or 1.9%. The *fz2* Felzenszwalb's single SPS configuration achieved the best results followed by three others using the same algorithm. With this OSS combination of method and backbone, 17 of 28 total SPS configurations improved the base results. As in the other OpenGMM tested scenario, using SLIC and QuickShift single SPS or merged with FuSC delivered the worst results, especially with the larger superpixels.

FuSC configurations achieve much more stable average results than single SPSs configurations. The average results and the standard deviations - grouped by: method, backbone, and SPS algorithm - are presented in Table 6.5. We can observe the much smaller standard deviation of FuSC configurations suggesting that FuSC produces similar results with different base single SPS algorithms

SPS config.	UUCs 0	1	2	3	4	Avg. AUROC	px/seg
-	.884	.934	.710	.867	.854	.850	.0
fz_slic4	.906	.960	.730	.883	.886	**.873**	306
fz_slic3	.906	.960	.730	.881	.883	.872	462
fz1	.905	.959	.731	.882	.880	.871	322
fz_quick1	.906	.958	.729	.882	.881	.871	157
fz_slic2	.906	**.961**	.729	.879	.879	.871	554
fz2	.900	.958	.730	.877	.888	.871	491
fz_quick2	.909	.959	.730	.881	.873	.870	254
fz_quick3	.909	.959	.730	.879	.875	.870	344
fz5	.906	.960	**.732**	.882	.871	.870	582
slic6	.904	.959	.728	**.884**	.876	.870	261
fz_quick4	.908	.960	.731	.878	.873	.870	423
fz6	.901	.959	.731	.877	.879	.869	780
fz_slic1	.905	.960	.728	.877	.876	.869	630
slic5	.904	.959	.729	.883	.868	.869	366
quick1	.910	.959	.730	.881	.861	.868	367
quick_slic4	.908	.960	.729	**.884**	.856	.867	301
slic4	.904	.958	.727	.879	.855	.865	639
fz3	.888	.954	.724	.861	.891	.864	849
quick_slic3	.908	.960	.729	.882	.831	.862	454
slic3	.904	.957	.724	.877	.834	.859	897
quick2	**.912**	.960	.729	.877	.813	.858	824
quick_slic2	.909	.960	.728	.879	.808	.857	545
quick_slic1	.908	.960	.727	.878	.800	.855	622
slic2	.902	.955	.720	.870	.810	.851	1317
fz4	.867	.937	.712	.830	**.898**	.849	1469
quick3	.909	.958	.727	.867	.753	.843	1471
slic1	.894	.948	.713	.857	.741	.831	2959
quick4	.905	.955	.724	.858	.709	.830	2272

Table 6.2: The table shows the results for the Vaihingen dataset using CoReSeg compared with the results obtained from post-processing with different superpixel settings. The first row shows CoReSeg without post-processing, and the rows below present the results for distinct superpixel configurations sorted by average AUROC. The best results achieved are in bold for each column. This table shows AUROC results in all columns. The UUC numbers stand for respectively: 0 - impervious surfaces; 1 - building; 2 - high vegetation; 3 - low vegetation; and 4 - car.

SPS config.	UUCs 0	1	2	3	4	Avg. AUROC	px/seg
-	.872	.936	.646	.688	.687	.766	0
fz2	.880	.954	.650	.696	.701	**.776**	491
fz1	.881	.953	**.654**	.696	.692	.775	322
fz6	.880	**.955**	.650	.697	.690	.774	780
fz5	**.882**	.954	**.654**	.697	.680	.773	582
fz_quick4	**.882**	.954	.651	**.698**	.677	.772	423
fz_quick3	**.882**	.953	.651	**.698**	.676	.772	344
fz_quick2	.881	.952	.652	.697	.677	.772	254
fz_quick1	.879	.949	.652	.696	.682	.772	157
fz3	.874	.952	.637	.687	.704	.771	849
fz_slic2	.881	.954	.646	.695	.678	.771	554
fz_slic1	.880	**.955**	.644	.694	.678	.770	630
fz_slic4	.879	.952	.648	.696	.675	.770	306
fz_slic3	.880	.953	.646	.696	.674	.770	462
quick1	.881	.950	.648	.696	.655	.766	367
slic6	.876	.947	.645	.691	.668	.765	261
quick_slic4	.880	.951	.647	.695	.652	.765	301
slic5	.876	.948	.645	.692	.664	.765	366
quick_slic3	.881	.953	.643	.695	.633	.761	454
slic4	.875	.949	.640	.690	.649	.761	639
quick2	**.882**	.954	.644	**.698**	.614	.758	824
quick_slic2	**.882**	.954	.643	.694	.616	.758	545
quick_slic1	**.882**	.954	.641	.694	.609	.756	622
slic3	.875	.950	.636	.689	.628	.756	897
fz4	.858	.940	.610	.650	**.711**	.754	1469
slic2	.874	.949	.630	.686	.611	.750	1317
quick3	**.882**	.954	.636	.696	.567	.747	1471
quick4	.878	.953	.629	.691	.539	.738	2272
slic1	.869	.943	.612	.674	.546	.729	2959

Table 6.3: The table shows the results for the Vaihingen dataset using OpenGMM with DenseNet-121 as the backbone compared with the results obtained from post-processing with different superpixel settings. The first row shows CoReSeg without post-processing, and the rows below present the results for distinct superpixel configurations sorted by average AUROC. The best results achieved are in bold for each column. This table shows AUROC results in all columns. The UUC numbers stand for respectively: 0 - impervious surfaces; 1 - building; 2 - high vegetation; 3 - low vegetation; and 4 - car.

SPS config.	**UUCs** 0	1	2	3	4	**Avg. AUROC**	**px/seg**
-	.885	.911	.611	.648	.619	.735	0
fz2	.904	.936	.622	.665	.618	**.749**	491
fz1	.906	.934	.624	.665	.608	.747	322
fz6	.906	.937	.622	**.668**	.604	.747	780
fz5	.909	.937	.625	.667	.589	.745	582
fz_quick2	.91	.935	**.626**	.663	.591	.745	254
fz_quick1	.906	.932	.624	.661	.601	.745	157
fz_quick3	.91	.937	.625	.665	.587	.745	344
fz_quick4	.911	.938	.624	.665	.585	.745	423
fz_slic2	.910	.938	.619	.665	.588	.744	554
fz3	.898	.934	.611	.659	.616	.744	849
fz_slic4	.906	.934	.622	.663	.589	.743	306
fz_slic1	.908	.938	.618	.665	.583	.742	630
fz_slic3	.909	.937	.618	.664	.583	.742	462
quick1	.910	.935	.624	.661	.566	.739	367
slic6	.901	.930	.619	.658	.583	.738	261
slic5	.903	.931	.619	.659	.578	.738	366
quick_slic4	.908	.935	.620	.660	.557	.736	301
slic4	.905	.933	.616	.659	.557	.734	639
quick_slic3	.911	.938	.618	.661	.532	.732	454
quick2	**.915**	.939	.620	.662	.514	.730	824
slic3	.906	.933	.614	.659	.534	.729	897
quick_slic2	.911	.939	.620	.660	.515	.729	545
fz4	.878	.924	.592	.630	**.620**	.729	1469
quick_slic1	.911	.940	.619	.659	.515	.729	622
slic2	.906	.934	.611	.657	.508	.723	1317
quick3	.914	**.941**	.618	.660	.463	.719	1471
quick4	.910	.940	.613	.654	.446	.713	2272
slic1	.902	.929	.598	.647	.448	.705	2959

Table 6.4: The table shows the results for the Vaihingen dataset using OpenGMM with WideResNet-50 as the backbone compared with the results obtained from post-processing with different superpixel settings. The first row shows CoReSeg without post-processing, and the rows below present the results for distinct superpixel configurations sorted by average AUROC. The best results achieved are in bold for each column. This table shows AUROC results in all columns. The UUC numbers stand for respectively: 0 - impervious surfaces; 1 - building; 2 - high vegetation; 3 - low vegetation; and 4 - car.

Backbone	**Method**	**Seg. config**	**UUCs**					**Avg. AUROC**
			0	**1**	**2**	**3**	**4**	
U-net	CoReSeg	-	.884	.934	.710	.867	.854	.850
U-net	CoReSeg	fz	.895 ± .0149	.955 ± .0088	.727 ± .0077	.868 ± .0202	.885 ± .0097	.866 ± .0086
U-net	CoReSeg	quick	.909 ± .0029	.958 ± .0022	.728 ± .0026	.871 ± .0103	.784 ± .0667	.850 ± .0167
U-net	CoReSeg	slic	.902 ± .004	.956 ± .0042	.724 ± .0061	.875 ± .0101	.831 ± .0500	.858 ± .0148
U-net	CoReSeg	fz_quick	.908 ± .0014	.959 ± .0008	.73 ± .0008	.88 ± .0018	.876 ± .0038	.870 ± .0005
U-net	CoReSeg	fz_slic	.906 ± .0005	.960 ± .0005	.729 ± .0010	.88 ± .0026	.881 ± .0044	.871 ± .0017
U-net	CoReSeg	quick_slic	.908 ± .0005	.960 ± .0000	.728 ± .0010	.881 ± .0028	.824 ± .0252	.860 ± .0054
DN-121	OpenGMM	-	.872	.936	.646	.688	.687	.766
DN-121	OpenGMM	fz	.876 ± .0092	.951 ± .0056	.643 ± .0171	.687 ± .0186	.696 ± .0111	.771 ± .0083
DN-121	OpenGMM	quick	.881 ± .0019	.953 ± .0019	.639 ± .0085	.695 ± .0030	.594 ± .0512	.752 ± .0123
DN-121	OpenGMM	slic	.874 ± .0026	.948 ± .0025	.635 ± .0125	.687 ± .0067	.628 ± .0455	.754 ± .0137
DN-121	OpenGMM	fz_quick	.881 ± .0014	.952 ± .0022	.652 ± .0006	.697 ± .0010	.678 ± .0027	.772 ± .0000
DN-121	OpenGMM	fz_slic	.880 ± .0008	.954 ± .0013	.646 ± .0016	.695 ± .0010	.676 ± .0021	.770 ± .0005
DN-121	OpenGMM	quick_slic	.881 ± .0010	.953 ± .0014	.644 ± .0025	.695 ± .0006	.628 ± .0192	.760 ± .0039
WRN-50	OpenGMM	-	.885	.911	.611	.648	.619	.735
WRN-50	OpenGMM	fz	.900 ± .0115	.934 ± .0049	.616 ± .0128	.659 ± .0145	.609 ± .0116	.744 ± .0073
WRN-50	OpenGMM	quick	.912 ± .0026	.939 ± .0026	.619 ± .0046	.659 ± .0036	.497 ± .0542	.725 ± .0116
WRN-50	OpenGMM	slic	.904 ± .0021	.932 ± .0020	.613 ± .0079	.657 ± .0047	.535 ± .0509	.728 ± .0126
WRN-50	OpenGMM	fz_quick	.909 ± .0022	.936 ± .0026	.625 ± .0010	.664 ± .0019	.591 ± .0071	.745 ± .0000
WRN-50	OpenGMM	fz_slic	.908 ± .0017	.937 ± .0019	.619 ± .0019	.664 ± .0010	.586 ± .0032	.743 ± .0010
WRN-50	OpenGMM	quick_slic	.910 ± .0015	.938 ± .0022	.619 ± .0010	.660 ± .0008	.530 ± .0199	.732 ± .0033

Table 6.5: The table presents the average results of each execution using the same superpixel algorithm or FuSC configuration. The column *Seg. config* indicates which algorithm is aggregated to present in the *UUCs* columns the average AUROC and the standard deviation for each UUC. The last column also shows the overall average between all UUCs. The dark gray rows are the baseline OSS results of each method and backbone without post-processing.

B	Method	PP	UUCs					Avg.
			0	1	2	3	4	
D	OpenGMM	-	.872	.936	.646	.688	**.687**	.766
D	OpenGMM	FuSC	**.881**	**.953**	**.647**	**.696**	.661	**.767**
D	OpenGMM	SPS	.876	.950	.639	.689	.645	.760
U	CoReSeg	-	.884	.934	.71	.867	.854	.850
U	CoReSeg	FuSC	**.907**	**.960**†	**.729**†	**.880**†	**.860**†	**.867**†
U	CoReSeg	SPS	.901	.956	.726	.871	.839	.859
W	OpenGMM	-	.885	.911	.611	.648	**.619**	.735
W	OpenGMM	FuSC	**.909**†	**.937**	**.621**	**.663**	.569	**.740**
W	OpenGMM	SPS	.905	.934	.616	.658	.553	.733

Table 6.6: The table compares the average AUROC results between all tested scenarios with and without post-processing. In bold are the best-achieved results for the combination of method, backbone, and post-processing. The † symbol points to the best average results overall. The UUC numbers are respectively: 0 - impervious surfaces; 1 - building; 2 - high vegetation; 3 - low vegetation; and 4 - car. The *B* column indicates which backbone was used: “U" for U-net, “D" for DN-121, and “W" for WRN-50.

hyperparameters.

Finding suitable hyperparameters is one of the most time-consuming tasks when using superpixel algorithms. The distance among the achieved results presented in Tables 6.2, 6.3, and 6.4 shows that bad hyperparameters choices can lead the post-processing to worsen the results.

As stated in Section 5.5, the single SPSs merged using FuSC originally produce larger superpixels than the ones generated by the single SPSs configurations used in tests. Despite the larger input superpixels, the FuSC procedure generates medium-sized superpixels performing very homogeneously among tested configurations. This behavior may corroborate that FuSC is more robust to hyperparameter selection.

Table 6.6 shows the average results of FuSC and single SPS algorithms for each combination of method and backbone used for this ablation. The † symbol indicates the best overall average results for each UUC and also for the average. The cells in bold indicate the best average results within the combination of method and backbone. FuSC achieved the best average AUROC results in all tested scenarios. As pointed out before in this chapter, the results obtained by FuSC are more stable and much closer. FuSC configurations performed better on average, even though a single SPS algorithm achieved the best individual performance.

6.3 Conclusion

This ablation study defined the set of configurations used in post-processing for all combinations of methods and backbones and the hyperparameters and final CoReSeg architecture. The selected configurations, hyperparameters, and architecture were used to run all the final tests and motivated the discussion in the next chapter.

Based on Table 6.1, the selected CoReSeg architecture uses the attention mechanism in the closed-set backbone and in the reconstruction module. For the final tests, only the configuration with the best hyperparameters was used to compare with the other methods.

To run the final tests, all used methods were post-processed using the same set of SPS configurations which performed best for each UUC and the average for all scenarios as presented in Tables 6.2, 6.3, and 6.4. The final list of the twelve selected configurations follows fz01, fz02, fz04, fz05, fz06, quick02, fz_quick02_mean, fz_quick04_mean, fz_slic02_mean, fz_slic04_mean and slic06.

To better observe the evolution of the results of this ablation study, Figure 6.1 presents the OSS methods results with and without post-processing.

Some final observations worth mentioning for all post-processing scenarios:

1. in all tested scenarios the post-processing improved the original results;
2. FuSC performance was much more stable among the different configurations;
3. FuSC configurations deliver medium-sized superpixels;
4. the highest the AUROC the most consistent was the improvement obtained post-processing the OSS;
5. the size of the superpixels must be able to properly represent objects in the underlying image. Superpixel configurations with medium size objects achieved the best results ;
6. single or FuSC using the Felzenszwalb algorithm obtained better results in all tested scenarios, suggesting that the generated superpixels may present characteristics that better represent remote sensing images.

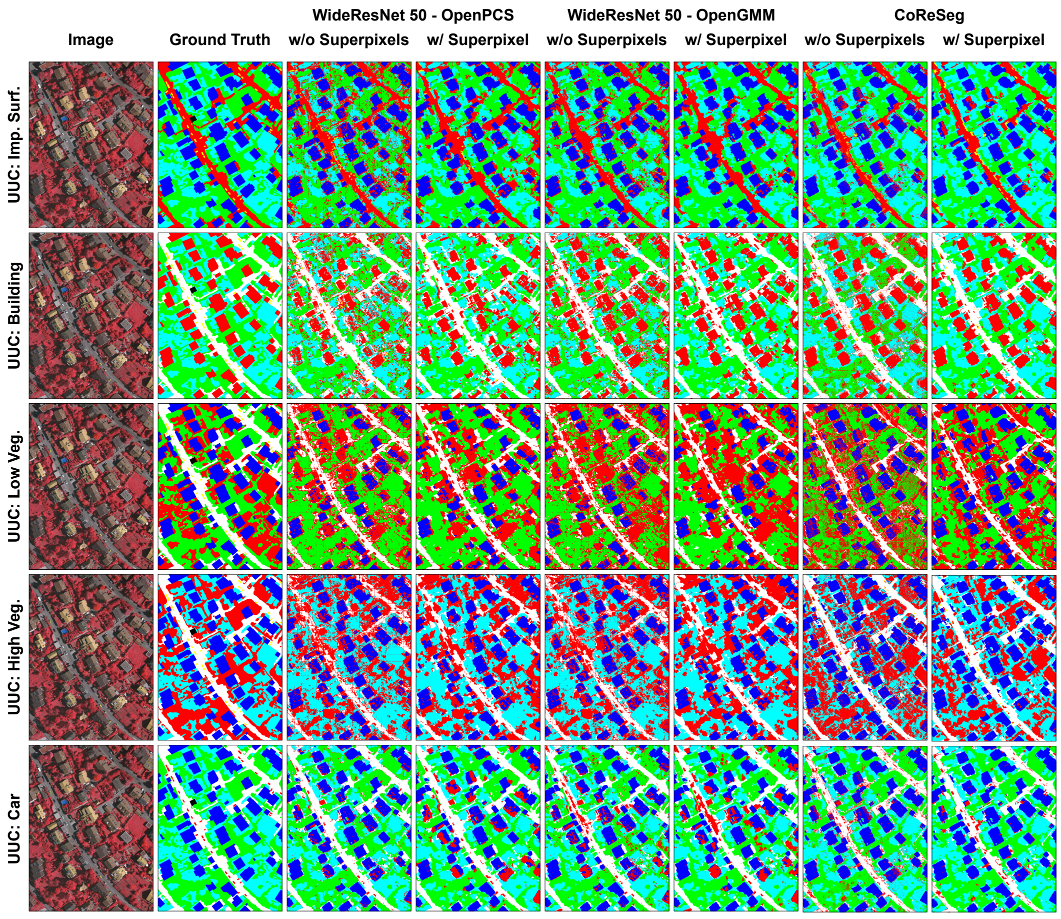


Figure 6.1: The figure shows qualitative results for an image from the Vaihingen dataset under different settings of UUCs and OSS methods. The proposed methods and the superpixel post-processing method generates cleaner segmentation, avoiding the usual mislabeling of unknown pixels.

7 Results and Discussion

This chapter presents and discusses the results of each proposed technique. The subsections show the results starting with the worst results achieved and ending with the best ones. Each subsection compares its results with all previously achieved best results (baseline and proposed).

The last section of the chapter presents a qualitative comparison among the best results for the baseline methods, OpenGMM and CoReSeg. The qualitative analysis also shows results for the post-processing.

It is worth mentioning that our experimental procedure is not the same used for OpenPCS by Oliveira et al. (2021) nor the one used for OpenPCS++ by Martinez et al. (2021), therefore all experiments were re-executed to ensure comparability, and the results presented here are different from the previous works.

Improving semantic consistency for OSS is the main goal pursued with this work. The use of attention mechanisms as a strategy to improve semantic consistency is also tested in this work jointly with all proposed methods as described in Chapter 4.

All discussed OSS methods use pre-trained closed-set backbones trained from scratch using the same stochastic batch selection and hyperparameters described in Chapter 5.

This chapter is organized as follows, Section 7.1 presents the results obtained by the OpenGMM method using all backbones with baseline methods. Section 7.2 shows the CoReSeg model results compared to OpenGMM and the baselines. Section 7.3 shows the results achieved with the post-processing method compared with the versions without post-processing. Finally, Section 7.4 evaluates the best qualitative results obtained.

7.1 Quantitative Results for OpenGMM

Tables 7.1 and 7.2 present the achieved AUROC for the open-set prediction for the Vaihingen and the Potsdam datasets using the OpenGMM method using DN-121 (Huang et al., 2017), WRN-50 (Zagoruyko and Komodakis, 2016) and U-net (Ronneberger et al., 2015) as backbones compared

BB	$\mathcal{A}$	Method	UUCs 0	1	2	3	4	Avg. AUROC
DN-121	✓	OpenGMM	**.90**	**.84**	.62	.50	**.81**	**.735 ± .167**
DN-121	✓	OpenPCS	.89	.82	.62	.54	.80	.735 ± .147
DN-121	✓	OpenPCS++	.69	.65	**.64**	**.61**	.72	.663 ± .046
DN-121	-	OpenGMM	**.85**	**.87**	**.64**	.70	**.71**	**.754 ± .010**
DN-121	-	OpenPCS	.84	**.87**	.61	**.73**	.68	.747 ± .107
DN-121	-	OpenPCS++	.62	.70	.62	.73	.66	.668 ± .048
U-net	✓	OpenGMM	**.84**	**.74**	**.62**	.56	**.68**	**.690 ± .108**
U-net	✓	OpenPCS	.81	.65	.62	**.59**	.56	.649 ± .097
U-net	✓	OpenPCS++	.59	.57	.51	.48	.51	.533 ± .045
U-net	-	OpenGMM	**.84**	**.53**	.64	.71	.55	**.654 ± .129**
U-net	-	OpenPCS	.81	.50	**.67**	**.74**	.45	.634 ± .155
U-net	-	OpenPCS++	.66	.48	.53	.60	**.61**	.575 ± .070
WRN-50	✓	OpenGMM	**.84**	**.79**	.49	.51	**.79**	**.686 ± .170**
WRN-50	✓	OpenPCS	.80	.78	.50	.54	.76	.677 ± .143
WRN-50	✓	OpenPCS++	.56	.56	**.61**	**.64**	.46	.566 ± .071
WRN-50	-	OpenGMM	**.83**	**.86**	.50	.52	**.68**	**.679 ± .171**
WRN-50	-	OpenPCS	.81	.85	.490	.53	.68	.671 ± .159
WRN-50	-	OpenPCS++	.42	.54	**.53**	**.62**	.65	.553 ± .092

Table 7.1: The table presents AUROC results for the Vaihingen dataset with all UUCs and the Average AUROC between all UUCs. The results are ordered, in order, by *BB* column as backbone, $\mathcal{A}$ column that indicates the use of the CBAM attention mechanism, and Average AUROC in descending order. The UUCs numerical notation stands for 0 - impervious surfaces, 1 - building, 2 - low vegetation, 3 - high vegetation, and 4 - car. In bold are highlighted the best AUROC for the combination of Dataset, Backbone, and CBAM usage.

to the baseline methods: OpenPCS and OpenPCS++. The tables also present results grouped by the usage of the CBAM attention mechanism ordering the results descending by the average AUROC.

Potsdam: Table 7.2 shows that amongst the tested methods, OpenGMM performed better using U-net and WRN-50 as backbones regardless of the use of the attention mechanism. In the scenario with DN-121 with CBAM as the backbone, OpenPCS was better in the average results, even with OpenGMM presenting better results for three UUCs (impervious surfaces, building, and car). Using DN-121 without the attention mechanism as the backbone, OpenPCS++ achieved the best performance. In both cases, OpenGMM was the second better method. OpenPCS with DN-121 with CBAM attention mechanism achieved the best overall performance for the Potsdam dataset.

Amongst all test scenarios, we observe that different methods performed better for different UUCs. If UUC was *impermeable surfaces* OpenGMM performed better in four tests and OpenPCS in the other two; if UUC was

BB	$\mathcal{A}$	**Method**	**UUCs**					**Avg. AUROC**
			0	**1**	**2**	**3**	**4**	
DN-121	✓	OpenPCS	.77	.76	**.52**	**.48**	.95	**.697 ± .193**
DN-121	✓	OpenGMM	**.78**	**.77**	.45	.45	**.95**	.681 ± .221
DN-121	✓	OpenPCS++	.52	.71	.49	.46	.91	.619 ± .188
DN-121	-	OpenPCS++	.67	.69	**.54**	**.64**	.90	**.689 ± .134**
DN-121	-	OpenGMM	.72	.76	.42	.52	**.93**	.668 ± .202
DN-121	-	OpenPCS	**.75**	**.79**	.39	.33	.87	.627 ± .247
U-net	✓	OpenGMM	**.77**	**.74**	.43	.40	**.90**	**.650 ± .224**
U-net	✓	OpenPCS	.70	.70	**.46**	.38	.87	.625 ± .2000
U-net	✓	OpenPCS++	.56	.69	.36	**.43**	.69	.553 ± .148
U-net	-	OpenGMM	**.81**	**.72**	.35	.39	**.91**	**.635 ± .252**
U-net	-	OpenPCS++	.56	.71	**.56**	**.57**	.68	.617 ± .075
U-net	-	OpenPCS	.77	.71	.33	.38	.85	.607 ± .236
WRN-50	✓	OpenGMM	**.67**	**.75**	.36	.38	**.91**	**.613 ± .239**
WRN-50	✓	OpenPCS	.66	.74	.39	.38	.90	.612 ± .226
WRN-50	✓	OpenPCS++	.37	.56	**.49**	**.50**	.82	.549 ± .168
WRN-50	-	OpenGMM	.66	**.76**	.30	**.47**	.93	**.626 ± .246**
WRN-50	-	OpenPCS	**.70**	.74	.30	.44	**.93**	.623 ± .250
WRN-50	-	OpenPCS++	.54	.63	**.46**	.46	.85	.588 ± .162

Table 7.2: AUROC results for the Potsdam dataset with all UUCs and the Average AUROC between all UUCs. The results are ordered, in order, by *BB* column as backbone, $\mathcal{A}$ column that indicates the use of CBAM attention mechanism, and Average AUROC in descending order. The UUCs numerical notation stands for 0 - impervious surfaces, 1 - building, 2 - low vegetation, 3 - high vegetation, and 4 - car. In bold are highlighted the best AUROC for the combination of Dataset, Backbone, and CBAM usage.

building, OpenGMM performed better in five test scenarios; if UUC was *low vegetation* OpenPCS++ was better in four and OpenPCS in the other two; when UUC was *high vegetation*, OpenPCS++ performed better in four test scenarios and OpenGMM and OpenPCS in one each; and if UUC was *car*, OpenGMM was better in five test scenarios and OpenPCS in one.

OpenGMM open-set predictions performed better for 14 individual UUCs across the test scenarios, OpenPCS++ predictions were better in 9, and OpenPCS predictions in 7 tests.

Vaihingen: Table 7.1 shows that among the methods tested with all backbones, evaluating the average AUROC, OpenGMM performed best in all six test scenarios. The performance pattern observed for all tested scenarios was the same, with OpenGMM showing the best performance followed by OpenPCS and OpenPCS++.

Comparing OpenGMM's performance between the Vaihingen and the Potsdam datasets, we observe that OpenGMM performed better on average

AUROC in all tested scenarios in the first dataset and for the second only in four out of six.

Comparing individual UUCs for the distinct scenarios, OpenGMM performed better for 15 UUCs, OpenPCS++ in 8, and OpenPCS in 7. For *impervious surfaces* and *building*, the OpenGMM performed better in all scenarios, and for *car* in five out of six test scenarios. For the vegetation classes as UUC, OpenPCS and OpenPCS++ performed better.

The use of the Mixture of Gaussians by OpenGMM seems capable of producing data representations more suitable to model data and identify OOD pixels as expected. Tables 7.2 and 7.1 present results used as evidence supporting the assumption.

7.2 Quantitative Results for CoReSeg

The Conditional Reconstruction for Open-set Segmentation model is a novel end-to-end fully convolutional model proposed in this work. This section presents the results for CoReSeg compared to the two baseline methods, OpenPCS and OpenPCS++, and to the OpenGMM method proposed in this work with its complete results presented in Section 7.1.

D	$\mathcal{A}$	**Method**	**UUCS**					**Avg.**
			0	**1**	**2**	**3**	**4**	**AUROC**
V	✓	CoReSeg+Att	.842	**.900**	.686	.718	**.723**	**.7738 ± .0922**
V	✓	CoReSeg	**.886**	.770	**.688**	**.738**	.627	.7418 ± .0971
V	✓	OpenGMM	.842	.738	.622	.561	.685	.6896 ± .1080
V	✓	OpenPCS	.812	.652	.621	.595	.563	.6486 ± .0970
V	✓	OpenPCS++	.590	.571	.507	.485	.513	.5332 ± .0449
V	-	CoReSeg	**.880**	**.817**	**.680**	.658	**.676**	**.7422 ± .0999**
V	-	OpenGMM	.841	.527	.641	.712	.547	.6536 ± .1285
V	-	OpenPCS	.815	.493	.672	**.737**	.455	.6344 ± .1555
V	-	OpenPCS++	.656	.476	.533	.600	.609	.5748 ± .0705
P	✓	CoReSeg+Att	.749	**.876**	**.646**	**.544**	.772	**.7174 ± .1268**
P	✓	OpenGMM	**.773**	.744	.432	.397	**.904**	.6500 ± .2236
P	✓	OpenPCS	.704	.704	.460	.384	.874	.6252 ± .1999
P	✓	OpenPCS++	.586	.692	.365	.435	.688	.5532 ± .1482

Table 7.3: In this table, the U-net is fixed and the $\mathcal{A}$ columns indicate if the backbone uses the CBAM attention mechanism within the U-net. In bold are the best results for each combination of the dataset and the use of the CBAM attention mechanism within the backbone. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car. The *D* column stands for the datasets with "V" for the Vaihingen and "P" for the Potsdam dataset.

The ablation study performed over the Vaihingen dataset and presented in Chapter 6 has its best results compiled and compared with the achieved results for the Potsdam dataset using the model and backbone selected after the ablation.

The configuration that performed best in ablation was the formulation of CoReSeg with the CBAM attention mechanism added to the reconstruction block and to the U-net backbone (Section 4.2), called CoReSeg+Att.

We present the results in two sequential blocks. The first one shows all results using the same backbone in Table 7.3. Comparing the results of the methods using the same close-set backbone is fairer since they all used the same closed-set output to identify the OOD pixels. Then CoReSeg+Att will be compared to the best results obtained by all combinations of method and backbone in Section 7.1.

CoReSeg uses only the U-net and its variation with attention as the backbones. Table 7.3 shows only results using U-net as the backbone. When comparing the methods using the same closed-set backbone CoReSeg+Att achieved the best overall performance with much stabler results for both Vaihingen and Potsdam datasets.

We tested CoReSeg using U-net+Att and U-net, and CoReSeg+Att for the Vaihingen dataset. The CoReSeg+Att method achieved better results than the other methods with the same backbone, and the three variations of CoReSeg achieved the best three overall results. Table 7.3 shows in bold the best AUROC results for each closed-set backbone used.

Evaluating individual UUC results in Table 7.3, CoReSeg and CoReSeg+Att also performed best in 12 UUCs of the 15 UUCs possible, 5 for each of the three tested scenarios.

In all comparisons, including Vaihingen and Potsdam, the CoReSeg method performed better than the other methods by a large AUROC margin between 0.0674 and 0.0886 compared to OpenGMM. The greatest difference obtained for Vaihingen used U-net and CoReSeg without the CBAM attention mechanism.

For all three tested scenarios with the same closed-set backbone, OpenPCS was the best-performing baseline method. Since OpenGMM overperformed the baseline methods, the performance gain achieved by CoReSeg using the U-net was even better if compared to the best baseline method results. Comparing the best CoReSeg result with the best OpenPCS result for each scenario the AUROC gap observed was between 0.0922 and 0.1252. For the Vaihingen dataset, CoReSeg improved baseline AUROC results by 19,30% and by 14,75% for the Potsdam dataset.

The second block presented in Table 7.4 shows the CoReSeg method and its variations compared with the best results for the baseline methods and OpenGMM regardless of the closed-set backbone.

For the methods without the use of attention in the backbone, OpenGMM overperformed OpenPCS and CoReSeg for Vaihingen by 0.0114 AUROC, equivalent to 1.54%.

Comparing the models with the use of attention in the backbone, CoReSeg+Att achieved the best overall AUROC performance for the Vaihingen dataset by 0.0386, and for the Potsdam dataset by 0.0200, the relative gain was 5.25% and 2.89% respectively. CoReSeg+Att improved results even when compared to other methods using closed-set backbones that performed much better compared to the U-net as shown in Table 7.3.

D	B	$\mathcal{A}$	Method	UUCs					Avg.
				0	1	2	3	4	AUROC
V	U	✓	CoReSeg+Att	.842	**.900**	.686	.718	.723	*.774 ± .092*
V	U	✓	CoReSeg	.886	.770	**.688**	**.738**	.627	.742 ± .097
V	D	✓	OpenGMM	**.900**	.845	.621	.502	**.808**	.735 ± .167
V	D	✓	OpenPCS	.894	.820	.622	.541	.796	.735 ± .147
V	D	✓	OpenPCS++	.694	.647	.638	.611	.725	.663 ± .046
V	D	-	OpenGMM	.851	**.866**	.641	.697	**.713**	.754 ± .100
V	D	-	OpenPCS	.842	**.866**	.614	**.733**	.678	.747 ± .107
V	U	-	CoReSeg	**.880**	.817	**.680**	.658	.676	.742 ± .100
V	D	-	OpenPCS++	.618	.703	.625	.729	.663	.668 ± .048
P	U	✓	CoReSeg+Att	.749	**.876**	**.646**	**.544**	.772	*.717 ± .127*
P	D	✓	OpenPCS	.771	.763	.525	.481	.947	.697 ± .193
P	D	✓	OpenGMM	**.783**	.773	.448	.455	**.948**	.681 ± .221
P	D	✓	OpenPCS++	.520	.712	.492	.463	.907	.619 ± .188

Table 7.4: The table shows the best results for the combination of the method, the use of attention in the backbone, and the dataset. The B column indicates the best performing backbone: “U" for U-net, “D" for DN-121, and W for WRN-50. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car. For the D columns “V" stands for the Vaihingen and “P" for the Potsdam dataset. The $\mathcal{A}$ column indicates the use of the CBAM attention mechanism in the backbone. The best results are in bold for each combination of the dataset and the use of the attention mechanism within the backbone, and the best average result for each dataset is in italic.

In both presented comparisons, CoReSeg+Att overperformed other methods and backbones in five out of six scenarios. The only scenario in which CoReSeg was not the best, OpenGMM, the other proposed method of this work, performed best. For both tested datasets, CoReSeg+Att obtained the best overall results.

Table 7.4 shows the CoReSeg method and its variations compared with the best results for the baseline methods and OpenGMM regardless of the closed-set backbone. CoReSeg+Att (U-net+Att) achieved the best overall AUROC performance for Vaihingen by 0.0386 and for Potsdam by 0.0200.

Without the CBAM attention mechanism, OpenGMM and OpenPCS overperformed CoReSeg for the Vaihingen dataset by 0.0114 AUROC. CoReSeg+Att improved results even when compared to methods using other closed-set backbones.

In both presented comparisons, CoReSeg overperformed the other meth-

Avg. AUROC	Vaihingen			Potsdam		
	Base	Post-processing		Base	Post-processing	
		FuSC	SPS		FuSC	SPS
all	1	14	6	0	10	9
> 0.6	0	13	5	0	9	7
> 0.7	0	9	1	0	2	1

Table 7.5: A scoreboard of the OSS best AUROC results shows which performed better in 3 distinct conditions: counting all scenarios, counting only scenarios with AUROC greater than 0.6, and counting only scenarios with AUROC greater than 0.7.

ods and backbones in five out of six scenarios. In the only scenario in which CoReSeg was not better, OpenGMM performed best. For both tested datasets, CoReSeg+Att obtained the best overall results.

7.3 Quantitative Results with Post-processing

This section presents the best post-processing results compared to results obtained without post-processing. Tables B.1 to B.40 in Appendix B presents the complete set of obtained results.

For Vaihingen, we executed twenty-one test scenarios combining distinct methods, backbone, and use of the CBAM attention mechanism (denoted by the suffix *+Att*), and for Potsdam a total of nineteen scenarios.

For each scenario, we tested eleven superpixel configurations. Four tested configurations used FuSC, and seven used a single superpixel generation algorithm. In this section, we also compare the average results for FuSC and Single SPS for each of the forty scenarios.

Tables 7.7 to 7.12 presents the baseline results compared to the best and worst post-processing results for both datasets.

Tables 7.5 and 7.6 show scoreboards comparing the overall results for post-processing with baseline. These tables show how many test scenarios each proposed strategy performed best.

For all results evaluated together, the proposed post-processing procedure improved the quantitative results for 39 in 40 of the tested scenarios. We could observe that the better the base result for the open-set segmentation, the better the improvement obtained. In the same direction, post-processing baseline results with better semantic consistency produced better open-set segmentations.

CoReSeg produces open-set segmentations with fewer artifacts and better semantic consistency compared to OpenPCS, OpenPCS++, and OpenGMM.

Avg. AUROC	Vaihingen			Potsdam		
	Base	Post-processing		Base	Post-processing	
		FuSC	SPS		FuSC	SPS
all	1	18	2	2	15	2
> 0.6	0	16	2	1	14	0
> 0.7	0	7	0	0	1	0

Table 7.6: The table shows a scoreboard comparing the average of FuSC and Single SPS configurations. The table presents the score of each superpixel strategy that performed better in 3 distinct conditions: counting all scenarios, counting only scenarios with AUROC greater than 0.6, and counting only scenarios with AUROC greater than 0.7.

Therefore, post-processing CoReSeg achieved the greatest AUROC improvements, and when using the CBAM attention mechanism the improvement was even greater. This result suggests that attention mechanisms may play an important role in open-set segmentation studies.

The results worsened after post-processing only when the segmentation method was the OpenPCS++ method with U-net+Att as the backbone. In this scenario, the baseline average AUROC was 0.5332. An AUROC value of 0.5 ranks a random positive sample higher than an aleatory negative sample half the time. The classification/segmentation, in this case, is not informative, and one can say that its predictive ability is likely random guessing.

Results may suggest that open-set segmentations with better semantic consistency and higher AUROC may benefit more from post-processing. The highest pixel count defines the semantic class for each superpixel. Thus, misclassified pixels within well-defined and correctly classified objects are corrected.

Table 7.5 shows that the post-processing improved the AUROC results for thirty-nine out of forty total baseline results, with FuSC producing the best results for twenty-four tested scenarios.

Table 7.6 shows the scoreboard of the average results of all FuSC and Single SPS configurations compared to baseline scenarios. This comparison suggests that the performance achieved by FuSC varies less and delivers a better segmentation on average when compared to the Single SPS configurations. This observed behavior corroborates the ablation observation that FuSC produced closer results than Single SPS. FuSC produced closer final results even with very distinct hyperparameters, suggesting that the hyperparameter selection is less relevant when using FuSC.

We should highlight that individual superpixel algorithm configurations used as input for FuSC produce larger superpixels on average than the ones produced by single SPS configurations. Even though the observed performance of FuSC was consistently better, suggesting that the merge procedure can produce more reliable superpixels to represent the objects in the image.

Using the average values presented in Table 7.6, FuSC produced better results for thirty-three of the forty different tested scenarios. The baseline result was better in three scenarios and the Single SPS algorithms in four. Post-processing using FuSC produces better results on average than using Single SPS.

The results of post-processing using FuSC stands out as an even better option with segmentations with high AUROC values. FuSC seems to benefit from higher AUROC value segmentations. FuSC produces all the best results with AUROC greater than 0.7, and the great majority in other threshold scenarios.

To corroborate the apparent greater stability and reliability of the post-processing results using FuSC, we can compare the standard deviations between the means of FuSC and Single SPS. The standard deviations of the FuSC averages are smaller in most cases, as can be seen in Tables 7.13 to 7.18.

For the Vaihingen dataset segmented using the DN-121 as the backbone, only post-processing OpenGMM worsen the results in Table 7.7. Using U-net as the backbone, Table 7.8 shows two scenarios that the post-processing could deteriorate the results depending on the configurations and the only case that the post-processing could not improve the results with any configuration. With the use of WRN-50 as the backbone presented in Table 7.9 we can see the two scenarios that the choice of superpixel configuration could deteriorate the result.

As the AUROC results for the Potsdam dataset with DN-121 as the backbone presented in Table 7.10 are lower and only in one tested scenario the post-processing improved the results for all configurations. Using U-net (Table 7.11) and WRN-50 (Table 7.12) as the backbone the same behavior identified for DN-121 repeated and in only one tested scenario the post-processing could improve results for all configurations.

Post-processing better base open-set segmentations with FuSC performed better as observed before in this section. Post-processing base open-set segmentations with lower AUROC values with single SPS or FuSC showed mixed results, and the configurations of single SPS performed better in some scenarios.

$\mathcal{A}$	Method	UUCs					Avg.	SPS
		0	1	2	3	4	AUROC	config.
-	OpenGMM	.87	.87	.65	.72	.73	.767 ± .100	fz_slic02
-	OpenGMM	.84	.85	.63	.72	.73	.754 ± .091	fz04
-	OpenGMM	.85	.87	.64	.70	.71	.754 ± .100	-
-	OpenPCS	.87	.88	.62	.76	.69	.764 ± .112	fz_quick04
-	OpenPCS	.85	.86	.62	.76	.70	.757 ± .101	fz04
-	OpenPCS	.84	.87	.61	.73	.68	.747 ± .107	-
-	OpenPCS++	.64	.72	.63	.75	.67	.682 ± .053	fz_quick04
-	OpenPCS++	.63	.70	.62	.75	.66	.672 ± .053	fz04
-	OpenPCS++	.62	.70	.62	.73	.66	.668 ± .048	-
✓	OpenGMM	.91	.86	.64	.50	.83	.747 ± .173	fz_quick04
✓	OpenGMM	.90	.84	.62	.50	.81	.735 ± .167	-
✓	OpenGMM	.89	.83	.62	.50	.82	.734 ± .166	fz04
✓	OpenPCS	.92	.84	.64	.54	.82	.752 ± .156	fz_quick04
✓	OpenPCS	.89	.81	.63	.53	.83	.740 ± .152	fz04
✓	OpenPCS	.89	.82	.62	.54	.80	.735 ± .147	-
✓	OpenPCS++	.72	.66	.65	.63	.73	.678 ± .048	fz_quick04
✓	OpenPCS++	.69	.63	.63	.62	.74	.663 ± .053	fz04
✓	OpenPCS++	.69	.65	.64	.61	.72	.663 ± .046	-

Table 7.7: The table shows the AUROC results for the base open-set prediction obtained by combining DN-121 with or without attention to the method for the Vaihingen dataset. Each backbone-method pair compares the performance of the base open-set prediction with the best and the worst post-processing configuration results. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car. The $\mathcal{A}$ (attention) column indicates if the backbone uses the CBAM attention mechanism as presented in section 5.1.

$\mathcal{A}$	Method	UUCs					Avg.	SPS
		0	1	2	3	4	AUROC	config.
-	CoReSeg	.92	.85	.68	.688	.71	.769 ± .107	fz_quick04
-	CoReSeg	.89	.84	.65	.67	.71	.751 ± .106	fz04
-	CoReSeg	.88	.82	.68	.66	.68	.742 ± .100	-
-	OpenGMM	.88	.52	.66	.71	.55	.663 ± .144	fz02
-	OpenGMM	.84	.53	.64	.71	.55	.654 ± .128	-
-	OpenGMM	.87	.53	.64	.70	.52	.652 ± .145	quick02
-	OpenPCS	.86	.48	.69	.75	.45	.645 ± .177	fz02
-	OpenPCS	.81	.49	.67	.74	.45	.634 ± .155	-
-	OpenPCS	.85	.48	.67	.74	.42	.634 ± .180	quick02
-	OpenPCS++	.70	.47	.53	.60	.62	.585 ± .091	fz_slic02
-	OpenPCS++	.69	.47	.53	.60	.61	.579 ± .083	slic06
-	OpenPCS++	.66	.48	.53	.60	.61	.575 ± .070	-
✓	CoReSeg	.91	.81	.72	.79	.65	.777 ± .097	fz_quick04
✓	CoReSeg	.87	.81	.70	.77	.64	.758 ± .089	fz04
✓	CoReSeg	.89	.77	.69	.74	.63	.742 ± .097	-
✓	CoReSeg+Att	.87	.94	.73	.75	.79	.815 ± .086	fz_quick04
✓	CoReSeg+Att	.83	.93	.72	.70	.79	.795 ± .089	fz04
✓	CoReSeg+Att	.84	.90	.69	.72	.72	.774 ± .092	-
✓	OpenGMM	.88	.74	.63	.59	.72	.713 ± .112	fz04
✓	OpenGMM	.86	.74	.63	.57	.69	.698 ± .112	slic06
✓	OpenGMM	.84	.74	.62	.56	.68	.690 ± .108	-
✓	OpenPCS	.86	.67	.63	.63	.59	.675 ± .106	fz04
✓	OpenPCS	.83	.66	.63	.60	.55	.656 ± .107	slic06
✓	OpenPCS	.81	.65	.62	.59	.56	.649 ± .097	-
✓	OpenPCS++	.59	.57	.51	.48	.51	.533 ± .045	-
✓	OpenPCS++	.59	.57	.50	.48	.52	.532 ± .047	fz_slic04
✓	OpenPCS++	.57	.56	.49	.48	.51	.523 ± .042	fz04

Table 7.8: The table shows the AUROC for the base open-set prediction obtained by the combination of U-net with or without attention to the method for the Vaihingen dataset. Each backbone-method pair compares the performance of the base open-set prediction with the best and the worst post-processing configuration results. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car. The $\mathcal{A}$ (attention) column indicates if the backbone uses the CBAM attention mechanism as presented in section 5.1.

$\mathcal{A}$	Method	UUCs					Avg.	SPS
		0	1	2	3	4	AUROC	config.
-	OpenGMM	.87	.88	.52	.52	.72	.702 ± .177	fz_quick04
-	OpenGMM	.87	.88	.51	.52	.68	.693 ± .183	quick02
-	OpenGMM	.83	.86	.50	.52	.68	.679 ± .171	-
-	OpenPCS	.84	.86	.51	.55	.70	.693 ± .161	fz_quick04
-	OpenPCS	.83	.86	.50	.54	.69	.684 ± .163	slic06
-	OpenPCS	.81	.85	.49	.53	.68	.671 ± .159	-
-	OpenPCS++	.41	.55	.52	.65	.66	.556 ± .103	fz_quick02
-	OpenPCS++	.42	.54	.53	.62	.65	.553 ± .092	-
-	OpenPCS++	.39	.53	.53	.59	.65	.540 ± .096	fz04
✓	OpenGMM	.87	.81	.53	.53	.83	.713 ± .169	fz_quick04
✓	OpenGMM	.86	.81	.51	.52	.82	.702 ± .173	slic06
✓	OpenGMM	.84	.80	.49	.51	.79	.686 ± .170	-
✓	OpenPCS	.81	.79	.54	.59	.82	.711 ± .133	fz04
✓	OpenPCS	.82	.80	.52	.55	.80	.697 ± .148	slic06
✓	OpenPCS	.80	.78	.50	.54	.76	.677 ± .143	-
✓	OpenPCS++	.55	.56	.60	.66	.50	.572 ± .058	quick02
✓	OpenPCS++	.56	.56	.61	.64	.46	.566 ± .071	-
✓	OpenPCS++	.54	.53	.60	.65	.47	.558 ± .070	fz04

Table 7.9: The table shows the AUROC for the base open-set prediction obtained by the combination of WRN-50 with or without attention to the method for the Vaihingen dataset. Each backbone-method pair compares the performance of the base open-set prediction with the best and the worst post-processing configuration results. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car. The $\mathcal{A}$ (attention) column indicates if the backbone uses the CBAM attention mechanism as presented in section 5.1.

$\mathcal{A}$	Method	UUCs					Avg.	SPS
		0	1	2	3	4	AUROC	config.
-	OpenGMM	.75	.77	.41	.52	.93	.677 ± .209	fz06
-	OpenGMM	.72	.76	.42	.52	.93	.668 ± .205	-
-	OpenGMM	.63	.78	.36	.49	.87	.626 ± .209	fz04
-	OpenPCS	.77	.796	.38	.33	.87	.630 ± .253	quick02
-	OpenPCS	.75	.79	.39	.33	.87	.627 ± .247	-
-	OpenPCS	.54	.796	.38	.35	.88	.589 ± .239	fz04
-	OpenPCS++	.75	.74	.52	.66	.91	.716 ± .140	fz06
-	OpenPCS++	.67	.69	.54	.64	.90	.686 ± .131	-
-	OpenPCS++	.696	.66	.50	.60	.87	.666 ± .133	fz04
✓	OpenGMM	.80	.80	.45	.46	.95	.691 ± .227	fz_quick04
✓	OpenGMM	.78	.77	.45	.45	.95	.681 ± .221	-
✓	OpenGMM	.69	.79	.41	.45	.89	.646 ± .208	fz04
✓	OpenPCS	.79	.78	.52	.48	.95	.706 ± .200	fz_quick04
✓	OpenPCS	.77	.76	.52	.48	.95	.697 ± .193	-
✓	OpenPCS	.68	.77	.46	.47	.89	.656 ± .189	fz04
✓	OpenPCS++	.55	.76	.46	.45	.93	.632 ± .209	fz06
✓	OpenPCS++	.54	.75	.41	.50	.90	.621 ± .199	fz04
✓	OpenPCS++	.52	.71	.49	.46	.91	.619 ± .188	-

Table 7.10: The table shows the AUROC for the base open-set prediction obtained by the combination of DN-121 with or without attention to the method for the Potsdam dataset. Each backbone-method pair compares the performance of the base open-set prediction with the best and the worst post-processing configuration results. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car. The $\mathcal{A}$ (attention) column indicates if the backbone uses the CBAM attention mechanism as presented in section 5.1.

$\mathcal{A}$	Method	UUCs					Avg.	SPS
		0	1	2	3	4	AUROC	config.
-	OpenGMM	.84	.75	.34	.39	.93	.649 ± .267	fz_slic02
-	OpenGMM	.81	.72	.35	.39	.91	.635 ± .252	-
-	OpenGMM	.74	.72	.32	.41	.87	.610 ± .237	fz04
-	OpenPCS	.80	.73	.32	.38	.88	.625 ± .257	fz_slic02
-	OpenPCS	.77	.71	.33	.38	.85	.607 ± .236	-
-	OpenPCS	.70	.73	.29	.42	.84	.596 ± .230	fz04
-	OpenPCS++	.58	.76	.56	.58	.72	.640 ± .092	fz_slic02
-	OpenPCS++	.56	.71	.56	.57	.68	.617 ± .0746	-
-	OpenPCS++	.55	.75	.45	.57	.67	.598 ± .117	fz04
✓	CoReSeg+Att	.77	.89	.67	.55	.82	.741 ± .112	fz_quick04
✓	CoReSeg+Att	.75	.88	.65	.54	.77	.717 ± .127	-
✓	CoReSeg+Att	.64	.87	.58	.50	.79	.676 ± .150	fz04
✓	OpenGMM	.84	.75	.43	.40	.92	.669 ± .237	fz04
✓	OpenGMM	.77	.74	.43	.40	.90	.650 ± .224	-
✓	OpenGMM	.76	.72	.45	.43	.88	.649 ± .199	fz04
✓	OpenPCS	.80	.71	.44	.39	.89	.640 ± .222	fz04
✓	OpenPCS	.70	.70	.46	.38	.87	.620 ± .200	-
✓	OpenPCS	.74	.69	.40	.40	.85	.619 ± .207	fz04
✓	OpenPCS++	.64	.73	.34	.44	.72	.574 ± .175	fz04
✓	OpenPCS++	.61	.71	.35	.42	.72	.562 ± .154	slic06
✓	OpenPCS++	.59	.69	.36	.43	.69	.552 ± .148	-

Table 7.11: The table shows the AUROC for the base open-set prediction obtained by the combination of U-net with or without attention to the method for the Potsdam dataset. Each backbone-method pair compares the performance of the base open-set prediction with the best and the worst post-processing configuration results. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car. The $\mathcal{A}$ (attention) column indicates if the backbone uses the CBAM attention mechanism as presented in section 5.1.

$\mathcal{A}$	Method	UUCs					Avg.	SPS
		0	1	2	3	4	AUROC	config.
-	OpenGMM	.67	.80	.30	.47	.95	$.639 \pm .261$	fz_slic02
-	OpenGMM	.66	.76	.30	.47	.93	$.626 \pm .246$	-
-	OpenGMM	.57	.799	.30	.46	.89	$.605 \pm .242$	fz04
-	OpenPCS	.72	.79	.28	.45	.95	$.640 \pm .268$	fz_quick04
-	OpenPCS	.70	.74	.30	.44	.93	$.622 \pm .250$	-
-	OpenPCS	.61	.79	.29	.47	.89	$.601 \pm .240$	fz04
-	OpenPCS++	.58	.69	.47	.46	.91	$.622 \pm .184$	fz04
-	OpenPCS++	.55	.66	.46	.46	.91	$.608 \pm .186$	slic06
-	OpenPCS++	.54	.63	.46	.46	.85	$.588 \pm .162$	-
✓	OpenGMM	.69	.78	.35	.38	.93	$.625 \pm .252$	fz_slic02
✓	OpenGMM	.67	.75	.36	.38	.91	$.613 \pm .239$	-
✓	OpenGMM	.58	.76	.33	.39	.86	$.586 \pm .228$	fz04
✓	OpenPCS	.67	.77	.36	.38	.92	$.622 \pm .245$	fz_slic02
✓	OpenPCS	.66	.74	.39	.38	.90	$.612 \pm .226$	-
✓	OpenPCS	.56	.77	.33	.40	.86	$.584 \pm .226$	fz04
✓	OpenPCS++	.43	.60	.47	.51	.81	$.563 \pm .151$	fz04
✓	OpenPCS++	.37	.56	.49	.50	.82	$.549 \pm .168$	-
✓	OpenPCS++	.31	.63	.41	.48	.86	$.538 \pm .210$	fz04

Table 7.12: The table shows the AUROC for the base open-set prediction obtained by the combination of WRN-50 with or without attention to the method for the Potsdam dataset. Each backbone-method pair compares the performance of the base open-set prediction with the best and the worst post-processing configuration results. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car. The $\mathcal{A}$ (attention) column indicates if the backbone uses the CBAM attention mechanism as presented in section 5.1.

Backbone	Method	SPS	UUCs					Avg. AUROC
			0	1	2	3	4	
DN-121	OpenGMM	-	.851	.866	.641	.697	.713	.754 ± .089
DN-121	OpenGMM	FuSC	.867 ± .001	.875 ± .001	.647 ± .001	.715 ± .003	.726 ± .001	.766 ± .090
DN-121	OpenGMM	Single	.862 ± .009	.869 ± .008	.645 ± .007	.712 ± .004	.720 ± .007	.761 ± .089
DN-121	OpenPCS	-	.842	.866	.614	.733	.678	.747 ± .096
DN-121	OpenPCS	FuSC	.867 ± .003	.879 ± .001	.625 ± .001	.757 ± .003	.684 ± .002	.762 ± .100
DN-121	OpenPCS	Single	.865 ± .007	.874 ± .007	.624 ± .005	.756 ± .003	.678 ± .010	.759 ± .099
DN-121	OpenPCS++	-	.618	.703	.625	.729	.663	.668 ± .043
DN-121	OpenPCS++	FuSC	.633 ± .003	.722 ± .002	.632 ± .002	.748 ± .007	.672 ± .002	.681 ± .047
DN-121	OpenPCS++	Single	.633 ± .004	.718 ± .009	.629 ± .005	.747 ± .004	.667 ± .004	.679 ± .047
DN-121+Att	OpenGMM	-	.900	.845	.621	.502	.808	.735 ± .150
DN-121+Att	OpenGMM	FuSC	.914 ± .001	.857 ± .001	.636 ± .002	.499 ± .002	.825 ± .001	.746 ± .155
DN-121+Att	OpenGMM	Single	.909 ± .008	.852 ± .008	.634 ± .006	.495 ± .002	.820 ± .005	.742 ± .154
DN-121+Att	OpenPCS	-	.894	.820	.622	.541	.796	.735 ± .132
DN-121+Att	OpenPCS	FuSC	.914 ± .002	.837 ± .001	.639 ± .002	.536 ± .002	.822 ± .003	.750 ± .140
DN-121+Att	OpenPCS	Single	.910 ± .008	.833 ± .008	.638 ± .006	.534 ± .001	.818 ± .006	.747 ± .139
DN-121+Att	OpenPCS++	-	.694	.647	.638	.611	.725	.663 ± .041
DN-121+Att	OpenPCS++	FuSC	.724 ± .003	.655 ± .001	.647 ± .001	.621 ± .005	.733 ± .001	.676 ± .045
DN-121+Att	OpenPCS++	Single	.718 ± .012	.650 ± .007	.644 ± .007	.621 ± .003	.732 ± .008	.673 ± .044

Table 7.13: The table compares the base open-set prediction obtained by the method using DN-121 and DN-121+Att and presents the resulting scenarios for the Vaihingen dataset. The AUROC value of the base open-set prediction is compared to the average AUROC for post-processing with a FuSC or a Single SPS configuration. For each UUC the average AUROC values are presented with the standard deviation of the results. The last column shows the overall average between all UUCs. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Backbone	**Method**	**SPS**	**UUCs**					**Avg. AUROC**
			[illegible]	[illegible]	[illegible]	**3**	**4**	
U-net	CoReSeg	-	.880	.817	.680	.658	.676	.742 ± .089
U-net	CoReSeg	FuSC	.915 ± .002	.847 ± .003	.686 ± .002	.681 ± .005	.704 ± .001	.767 ± .096
U-net	CoReSeg	Single	.909 ± .010	.848 ± .004	.679 ± .014	.679 ± .006	.695 ± .017	.762 ± .098
U-net	OpenGMM	-	.841	.527	.641	.712	.547	.654 ± .115
U-net	OpenGMM	FuSC	.870 ± .006	.525 ± .002	.654 ± .002	.714 ± .001	.543 ± .002	.661 ± .126
U-net	OpenGMM	Single	.874 ± .008	.521 ± .008	.652 ± .007	.708 ± .005	.543 ± .012	.659 ± .128
U-net	OpenPCS	-	.815	.493	.672	.737	.455	.634 ± .139
U-net	OpenPCS	FuSC	.848 ± .006	.485 ± .003	.686 ± .002	.747 ± .001	.443 ± .003	.642 ± .155
U-net	OpenPCS	Single	.853 ± .009	.478 ± .009	.685 ± .008	.744 ± .005	.443 ± .012	.641 ± .157
U-net	OpenPCS++	-	.656	.476	.533	.600	.609	.575 ± .063
U-net	OpenPCS++	FuSC	.701 ± .006	.467 ± .003	.526 ± .002	.599 ± .003	.621 ± .002	.583 ± .080
U-net	OpenPCS++	Single	.715 ± .013	.464 ± .005	.524 ± .003	.590 ± .011	.621 ± .008	.583 ± .086
U-net+Att	CoReSeg	-	.886	.770	.688	.738	.627	.742 ± .087
U-net+Att	CoReSeg	FuSC	.910 ± .001	.803 ± .005	.723 ± .001	.783 ± .006	.651 ± .001	.774 ± .086
U-net+Att	CoReSeg	Single	.902 ± .015	.808 ± .007	.720 ± .009	.782 ± .008	.637 ± .017	.770 ± .089
U-net+Att	CoReSeg+Att	-	.842	.900	.686	.718	.723	.774 ± .082
U-net+Att	CoReSeg+Att	FuSC	.867 ± .001	.933 ± .003	.728 ± .003	.746 ± .001	.790 ± .004	.813 ± .077
U-net+Att	CoReSeg+Att	Single	.859 ± .012	.933 ± .004	.728 ± .004	.740 ± .015	.781 ± .007	.808 ± .078
U-net+Att	OpenGMM	-	.842	.738	.622	.561	.685	.690 ± .097
U-net+Att	OpenGMM	FuSC	.878 ± .006	.752 ± .003	.637 ± .001	.570 ± .004	.704 ± .003	.708 ± .105
U-net+Att	OpenGMM	Single	.882 ± .010	.753 ± .007	.635 ± .008	.566 ± .012	.701 ± .010	.707 ± .108
U-net+Att	OpenPCS	-	.812	.652	.621	.595	.563	.649 ± .087
U-net+Att	OpenPCS	FuSC	.851 ± .006	.668 ± .002	.638 ± .003	.608 ± .005	.559 ± .002	.665 ± .100
U-net+Att	OpenPCS	Single	.856 ± .009	.671 ± .005	.638 ± .010	.608 ± .011	.561 ± .014	.667 ± .102
U-net+Att	OpenPCS++	-	.590	.571	.507	.485	.513	.533 ± .040
U-net+Att	OpenPCS++	FuSC	.587 ± .001	.573 ± .000	.495 ± .002	.481 ± .001	.522 ± .002	.532 ± .042
U-net+Att	OpenPCS++	Single	.583 ± .005	.571 ± .005	.493 ± .003	.478 ± .003	.520 ± .003	.529 ± .042

Table 7.14: The table compares the base open-set prediction obtained by the method using U-net and U-net+Att and presents the resulting scenarios for the Vaihingen dataset. The AUROC value of the base open-set prediction is compared to the average AUROC for post-processing with a FuSC or a Single SPS configuration. For each UUC the average AUROC values are presented with the standard deviation of the results. The last column shows the overall average between all UUCs. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Backbone	Method	SPS	UUCs					Avg. AUROC
			0	1	2	3	4	
WRN-50	OpenGMM	-	.834	.864	.498	.517	.681	.679 ± .153
WRN-50	OpenGMM	FuSC	.867 ± .004	.879 ± .002	.514 ± .003	.522 ± .002	.713 ± .003	.699 ± .159
WRN-50	OpenGMM	Single	.865 ± .006	.876 ± .007	.516 ± .006	.514 ± .005	.711 ± .016	.696 ± .159
WRN-50	OpenPCS	-	.808	.846	.490	.531	.678	.671 ± .143
WRN-50	OpenPCS	FuSC	.835 ± .003	.863 ± .002	.506 ± .003	.547 ± .003	.699 ± .001	.690 ± .145
WRN-50	OpenPCS	Single	.832 ± .007	.861 ± .006	.510 ± .008	.542 ± .006	.697 ± .019	.688 ± .145
WRN-50	OpenPCS++	-	.419	.543	.527	.625	.653	.553 ± .082
WRN-50	OpenPCS++	FuSC	.405 ± .003	.543 ± .002	.524 ± .001	.645 ± .008	.659 ± .003	.555 ± .092
WRN-50	OpenPCS++	Single	.402 ± .006	.541 ± .005	.525 ± .004	.633 ± .017	.652 ± .007	.551 ± .090
WRN-50+Att	OpenGMM	-	.845	.794	.491	.511	.789	.686 ± .152
WRN-50+Att	OpenGMM	FuSC	.867 ± .003	.810 ± .001	.525 ± .006	.520 ± .004	.822 ± .003	.709 ± .153
WRN-50+Att	OpenGMM	Single	.863 ± .007	.809 ± .007	.529 ± .009	.517 ± .005	.820 ± .007	.707 ± .152
WRN-50+Att	OpenPCS	-	.796	.784	.505	.537	.761	.677 ± .128
WRN-50+Att	OpenPCS	FuSC	.823 ± .002	.804 ± .002	.533 ± .006	.560 ± .007	.802 ± .004	.705 ± .129
WRN-50+Att	OpenPCS	Single	.820 ± .006	.802 ± .007	.539 ± .010	.565 ± .014	.802 ± .009	.706 ± .126
WRN-50+Att	OpenPCS++	-	.557	.562	.612	.644	.457	.566 ± .064
WRN-50+Att	OpenPCS++	FuSC	.556 ± .001	.557 ± .003	.610 ± .001	.654 ± .006	.468 ± .001	.569 ± .063
WRN-50+Att	OpenPCS++	Single	.551 ± .007	.553 ± .010	.609 ± .006	.655 ± .005	.470 ± .014	.567 ± .063

Table 7.15: The table compares the base open-set prediction obtained by the method using WRN-50 and WRN-50+Att and presents the resulting scenarios for the Vaihingen dataset. The AUROC value of the base open-set prediction is compared to the average AUROC for post-processing with a FuSC or a Single SPS configuration. For each UUC the average AUROC values are presented with the standard deviation of the results. The last column shows the overall average between all UUCs. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Backbone	**Method**	**SPS**	**UUCs**					**Avg. AUROC**
			0	**1**	**2**	**3**	**4**	
DN-121	OpenGMM	-	.717	.760	.417	.518	.928	.668 ± .181
DN-121	OpenGMM	FuSC	.740 ± .004	.774 ± .002	.412 ± .001	.517 ± .001	.930 ± .004	.675 ± .186
DN-121	OpenGMM	Single	.726 ± .040	.777 ± .005	.400 ± .019	.511 ± .009	.916 ± .019	.666 ± .188
DN-121	OpenPCS	-	.751	.786	.388	.335	.873	.627 ± .221
DN-121	OpenPCS	FuSC	.647 ± .070	.796 ± .002	.386 ± .002	.334 ± .001	.874 ± .001	.607 ± .218
DN-121	OpenPCS	Single	.644 ± .071	.792 ± .007	.380 ± .009	.357 ± .039	.875 ± .003	.610 ± .214
DN-121	OpenPCS++	-	.666	.692	.537	.645	.903	.689 ± .119
DN-121	OpenPCS++	FuSC	.709 ± .012	.726 ± .006	.509 ± .017	.653 ± .007	.920 ± .003	.703 ± .133
DN-121	OpenPCS++	Single	.723 ± .025	.722 ± .027	.515 ± .014	.643 ± .019	.906 ± .018	.702 ± .129
DN-121+Att	OpenGMM	-	.783	.773	.448	.455	.948	.681 ± .198
DN-121+Att	OpenGMM	FuSC	.798 ± .003	.792 ± .004	.447 ± .002	.456 ± .001	.956 ± .003	.690 ± .203
DN-121+Att	OpenGMM	Single	.783 ± .040	.796 ± .007	.439 ± .012	.450 ± .003	.941 ± .021	.682 ± .203
DN-121+Att	OpenPCS	-	.771	.763	.525	.481	.947	.697 ± .172
DN-121+Att	OpenPCS	FuSC	.788 ± .004	.780 ± .004	.519 ± .002	.482 ± .001	.956 ± .003	.705 ± .179
DN-121+Att	OpenPCS	Single	.773 ± .037	.784 ± .008	.507 ± .019	.476 ± .004	.941 ± .020	.696 ± .179
DN-121+Att	OpenPCS++	-	.520	.712	.492	.463	.907	.619 ± .168
DN-121+Att	OpenPCS++	FuSC	.529 ± .004	.739 ± .008	.477 ± .013	.464 ± .001	.941 ± .002	.630 ± .184
DN-121+Att	OpenPCS++	Single	.544 ± .011	.754 ± .013	.447 ± .024	.465 ± .014	.929 ± .013	.628 ± .187

Table 7.16: The table compares the base open-set prediction obtained by the method using DN-121 and DN-121+Att and presents the resulting scenarios for the Potsdam dataset. The AUROC value of the base open-set prediction is compared to the average AUROC for post-processing with a FuSC or a Single SPS configuration. For each UUC the average AUROC values are presented with the standard deviation of the results. The last column shows the overall average between all UUCs. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Backbone	Method	SPS	UUCs					Avg.
			0	1	2	3	4	AUROC
U-net	OpenGMM	-	.807	.725	.349	.387	.906	.635 ± .226
U-net	OpenGMM	FuSC	.835 ± .006	.746 ± .006	.343 ± .003	.386 ± .001	.924 ± .004	.647 ± .238
U-net	OpenGMM	Single	.828 ± .036	.745 ± .011	.331 ± .011	.392 ± .010	.908 ± .016	.641 ± .235
U-net	OpenPCS	-	.771	.707	.326	.384	.848	.607 ± .212
U-net	OpenPCS	FuSC	.801 ± .006	.732 ± .006	.318 ± .004	.383 ± .001	.879 ± .004	.623 ± .228
U-net	OpenPCS	Single	.793 ± .039	.736 ± .009	.305 ± .012	.391 ± .013	.865 ± .013	.618 ± .227
U-net	OpenPCS++	-	.561	.714	.557	.572	.681	.617 ± .067
U-net	OpenPCS++	FuSC	.578 ± .007	.751 ± .005	.551 ± .011	.581 ± .007	.720 ± .003	.636 ± .083
U-net	OpenPCS++	Single	.579 ± .017	.764 ± .014	.508 ± .033	.580 ± .011	.711 ± .019	.628 ± .097
U-net+Att	CoReSeg+Att	-	.749	.876	.646	.544	.772	.717 ± .113
U-net+Att	CoReSeg+Att	FuSC	.770 ± .001	.889 ± .002	.665 ± .003	.552 ± .001	.821 ± .003	.739 ± .119
U-net+Att	CoReSeg+Att	Single	.745 ± .045	.888 ± .007	.650 ± .027	.541 ± .017	.817 ± .011	.728 ± .125
U-net+Att	OpenGMM	-	.773	.744	.432	.397	.904	.650 ± .200
U-net+Att	OpenGMM	FuSC	.810 ± .009	.755 ± .003	.437 ± .001	.395 ± .001	.918 ± .003	.663 ± .209
U-net+Att	OpenGMM	Single	.817 ± .029	.751 ± .011	.438 ± .004	.402 ± .014	.906 ± .013	.662 ± .205
U-net+Att	OpenPCS	-	.704	.704	.460	.384	.874	.625 ± .179
U-net+Att	OpenPCS	FuSC	.747 ± .010	.716 ± .003	.453 ± .003	.383 ± .001	.889 ± .003	.638 ± .190
U-net+Att	OpenPCS	Single	.770 ± .029	.711 ± .008	.439 ± .018	.386 ± .008	.876 ± .012	.637 ± .192
U-net+Att	OpenPCS++	-	.586	.692	.365	.435	.688	.553 ± .133
U-net+Att	OpenPCS++	FuSC	.627 ± .010	.711 ± .005	.355 ± .011	.423 ± .004	.721 ± .004	.568 ± .151
U-net+Att	OpenPCS++	Single	.628 ± .016	.726 ± .007	.354 ± .013	.429 ± .009	.714 ± .010	.570 ± .152

Table 7.17: The table compares the base open-set prediction obtained by the method using U-net and U-net+Att and presents the resulting scenarios for the Potsdam dataset. The AUROC value of the base open-set prediction is compared to the average AUROC for post-processing with a FuSC or a Single SPS configuration. For each UUC the average AUROC values are presented with the standard deviation of the results. The last column shows the overall average between all UUCs. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Backbone	Method	SPS	UUCs					Avg. AUROC
			0	1	2	3	4	
WRN-50	OpenGMM	-	.665	.763	.304	.468	.932	.626 ± .220
WRN-50	OpenGMM	FuSC	.671 ± .001	.800 ± .008	.297 ± .003	.468 ± .002	.951 ± .002	.637 ± .233
WRN-50	OpenGMM	Single	.654 ± .033	.805 ± .011	.290 ± .010	.464 ± .003	.939 ± .020	.630 ± .233
WRN-50	OpenPCS	-	.697	.738	.301	.445	.934	.623 ± .224
WRN-50	OpenPCS	FuSC	.719 ± .003	.777 ± .008	.290 ± .004	.454 ± .000	.954 ± .002	.639 ± .237
WRN-50	OpenPCS	Single	.703 ± .039	.785 ± .011	.282 ± .010	.457 ± .006	.940 ± .019	.633 ± .236
WRN-50	OpenPCS++	-	.538	.628	.458	.465	.851	.588 ± .145
WRN-50	OpenPCS++	FuSC	.552 ± .007	.678 ± .013	.463 ± .011	.465 ± .002	.912 ± .003	.614 ± .169
WRN-50	OpenPCS++	Single	.561 ± .012	.697 ± .017	.461 ± .011	.475 ± .016	.900 ± .014	.619 ± .164
WRN-50+Att	OpenGMM	-	.673	.748	.356	.382	.907	.613 ± .213
WRN-50+Att	OpenGMM	FuSC	.688 ± .003	.777 ± .005	.352 ± .004	.378 ± .003	.924 ± .004	.624 ± .224
WRN-50+Att	OpenGMM	Single	.672 ± .038	.777 ± .011	.345 ± .007	.376 ± .008	.908 ± .019	.616 ± .223
WRN-50+Att	OpenPCS	-	.660	.739	.386	.381	.896	.612 ± .202
WRN-50+Att	OpenPCS	FuSC	.667 ± .001	.775 ± .006	.364 ± .006	.381 ± .001	.918 ± .003	.621 ± .218
WRN-50+Att	OpenPCS	Single	.650 ± .038	.778 ± .009	.352 ± .013	.382 ± .010	.902 ± .019	.613 ± .217
WRN-50+Att	OpenPCS++	-	.367	.565	.492	.501	.821	.549 ± .150
WRN-50+Att	OpenPCS++	FuSC	.294 ± .010	.610 ± .015	.444 ± .012	.488 ± .008	.876 ± .003	.542 ± .195
WRN-50+Att	OpenPCS++	Single	.326 ± .045	.621 ± .018	.436 ± .024	.492 ± .011	.860 ± .021	.547 ± .185

Table 7.18: The table compares the base open-set prediction obtained by the method using WRN-50 and WRN-50+Att and presents the resulting scenarios for the Potsdam dataset. The AUROC value of the base open-set prediction is compared to the average AUROC for post-processing with a FuSC or a Single SPS configuration. For each UUC the average AUROC values are presented with the standard deviation of the results. The last column shows the overall average between all UUCs. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

7.4 Qualitative Results

Let's assess CoReSeg's composition of the conditioned reconstruction's minimum error for each UUC scenario with the sample presented in Figure 7.1. The figure indicates the scenario in the columns, and in the rows, from up to bottom, the input image, the ground truth with UUC in red, the closed-set prediction, the four conditioned reconstruction errors, and the minimum error in the last row. The darker shades of gray indicate smaller reconstruction error values.

Impervious surfaces conditioning reconstruction error row shows that for all four conditioned scenarios the reconstruction of the *impervious surfaces* areas are darker and therefore produced smaller reconstruction error values. The similarity between *impervious surfaces* and *building* confounded the reconstruction notably when *low vegetation* is the UUC. Also, the presence of shadows has a remarkable impact when reconstructing the images.

For the *building* conditioning reconstruction error row, in all scenarios, the *vegetation* and *car* classes are poorly reconstructed, and the *building* and *impervious surfaces* show smaller reconstruction error values. Shadows play a relevant role in this scenario as observed in the *impervious surfaces* reconstruction error row.

The *high vegetation* conditioning reconstruction error row produced smaller reconstruction error values when conditioning *impervious surfaces*, *building*, and *low vegetation* as UUC. With the *car* as UUC, the model produced a mildly conditioned reconstruction, with high error values inside a continuous area of the *high vegetation* and low error values at the borders.

The *low vegetation* conditioning reconstruction error row produced low error values to all vegetation and shadow areas, except when *building* is the UUC with high error value areas for some *high vegetation* and shadow areas.

The *car* conditioning reconstruction error row produced high error values for most of the pixels. It could reconstruct the *car* class objects with *building* and *low vegetation* as UUC, but in the other cases, the conditioning failed to deliver low reconstruction error values for the *car* class objects.

Analyzing all reconstructions, the presence of shadows may tamper the final result. The vegetation classes can mislead the reconstruction model, possibly due to its intra-class and inter-class variability. The morphological and color similarities also interfere with the reconstruction process, producing equally good or poor-quality reconstructions of similar classes.

The dataset is highly unbalanced, and the car class has few pixels compared to others. The stochastic training adopted is sensitive to unbalanced

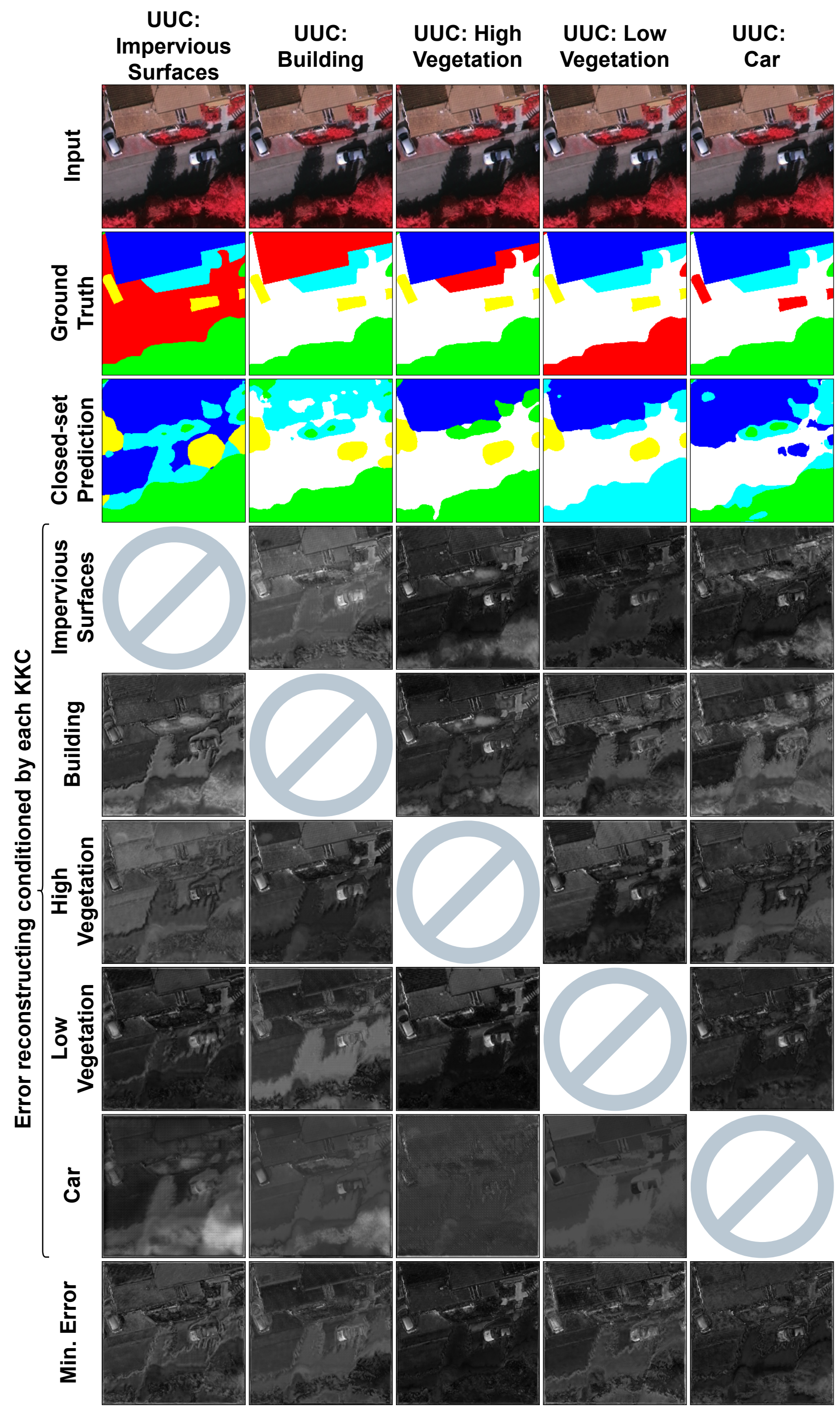


Figure 7.1: From up to bottom, the figure shows the input image, the ground truth with the UUC in red, the closed-set prediction, the four conditioned reconstruction error images, and the computed minimum error image. The crossed circle indicates that the reconstruction is not conditioned to that class since it is the UUC. Each column of the figure shows a distinct UUC scenario produced using the LOCO protocol. The used colors are: white for *impervious surfaces*; dark blue for *building*; light blue for *low vegetation*; green for *high vegetation*; yellow for *car*; and red for the UUC. Also, the darker the pixel, the smaller the error.

data, as while training, the model could see little or no data of the class with fewer pixels.

The minimum error row shows the final composition using all conditioned reconstruction error values produced during the conditioning. The resultant reconstruction error values used to identify the OOD pixels use the lower error values for each pixel found among all reconstructions computing the minimum reconstruction error.

The minimum reconstruction error computed for *impervious sufaces* and *building* as UUC shows higher reconstruction error values (lighter shades of gray) for areas of *impervious sufaces*, *building*, and some areas of *high vegetation*. The presence of shadows tampers the reconstruction producing mixed error values in continuous regions.

The areas of *high vegetation* as UUC are better reconstructed and showed lower error values than the other classes' areas. However, for *low vegetation* as UUC, the reconstruction mixed the vegetation areas and poorly reconstructed the areas with shadows and cars.

With *car* as UUC, the minimum reconstruction error showed higher error values for the areas with cars and some objects' borders. The expected behavior for this scenario. In this case, the closed-set and reconstruction models trained with the classes with more pixels and much less unbalanced, producing the expected poorer reconstructions for the *car* class.

Models trained in all scenarios could not condition the reconstruction with *car* class and produce well-conditioned reconstructions. Conditioning the reconstruction by the *car* class tampers the computation of the final minimum reconstruction error, making the error value higher and the OOD identification less effective. Higher reconstruction error values for *car* class areas in all scenarios indicate that the trained models could not learn good representations for this class. Besides that, CoReSeg presented better results compared to the baseline methods.

Even struggling to learn good representations and properly conditionally reconstruct the images in some scenarios, Figures 7.2 and 7.3 present the produced open-set predictions by the proposed OpenGMM and CoReSeg compared to the baseline method OpenPCS.

Both Figures 7.2 and 7.3 use the same structure. Each row shows a distinct emulated open-set scenario using the LOCO protocol. The columns of both figures show, from left to right, the input image, the ground truth, the three methods OpenPCS, OpenGMM, and CoReSeg in that order, and for each method, the raw predictions and the post-processed predictions. The red color represents the OOD pixels, white for *impervious surfaces*; dark blue for *building*; light blue for *low vegetation*; green for *high vegetation*; and yellow for *car*.

Figures 7.2 and 7.3 present the best-achieved results for each method among all tested post-processing configurations for all three and backbones for OpenPCS and OpenGMM.

Figure 7.2 presents the predictions for the Vaihingen dataset. OpenPCS and OpenGMM achieved the best results using DN-121 (Section 5.1) as the backbone for the Vaihingen dataset. CoReSeg using the CBAM attention mechanism and U-net+Att (Section 5.1) as the backbone achieved the best results. The post-processing configurations (Section 5.5) with the best AUROC results presented used fz_quick04 configuration for OpenPCS, fz_slic02 for OpenGMM; and fz_quick04 for CoReSeg+Att.

Comparing the achieved results of the methods without post-processing, Figure 7.2 shows improvement of OpenGMM over OpenPCS. A comparison of qualitative results between OpenGMM and OpenPCS shows more preserved edges and fewer artifacts in open-set prediction, even with close quantitative results.

A noticeable improvement is visible comparing CoReSeg+Att with the first two methods for the Vaihingen dataset. CoReSeg results are cleaner with small-sized artifacts, and borders are better preserved making objects easier to delimit and observe. It is less common to find a big block of pixels wrongly classified.

Post-processing qualitative results noticeably improved the open-set prediction quality. Comparing the post-processed results with the base prediction for each method shows that the post-processing could drastically reduce artifacts and preserve edges. The delimitation of the natural objects is more observable, and the open-set prediction is more consistent with the ground truth quantitative and qualitatively.

The base open-set prediction produced by CoReSeg+Att already showed smaller artifacts and better edge delineation. Therefore, the improvement in post-processing prediction was more apparent compared to that obtained with OpenPCS and OpenGMM.

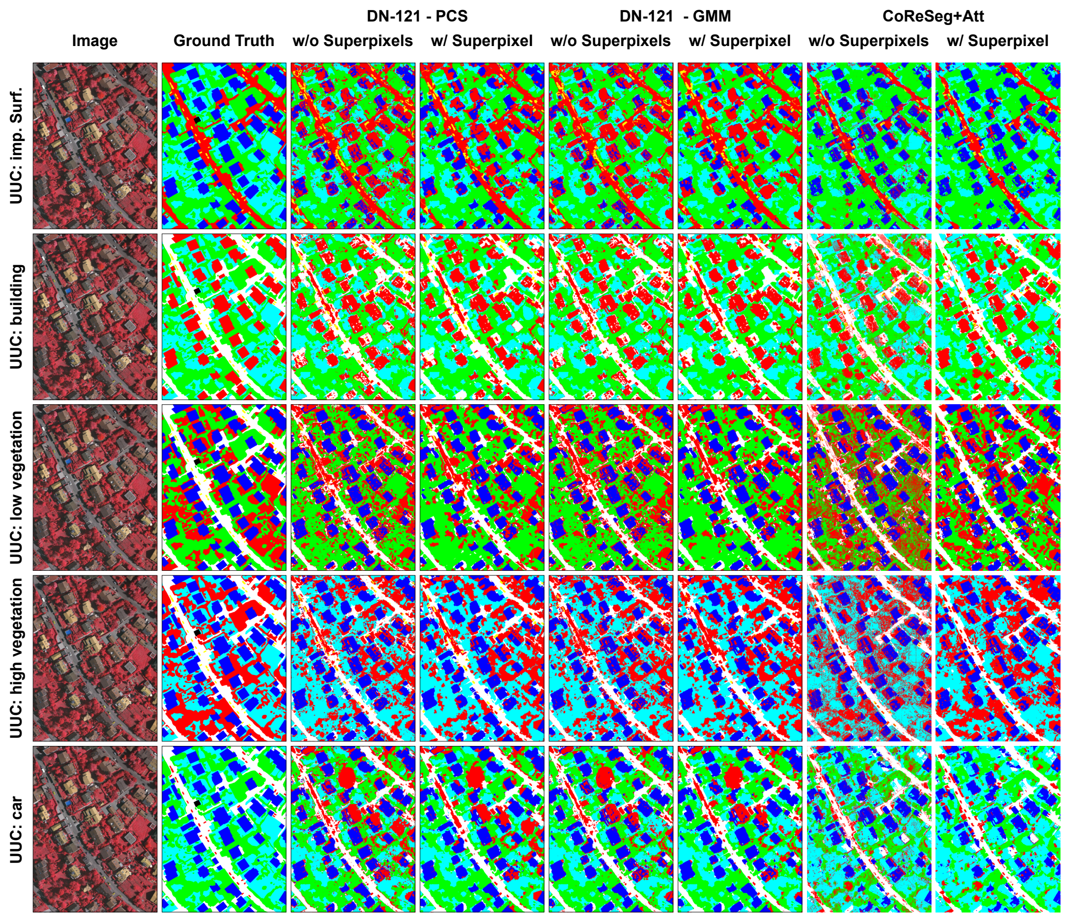


Figure 7.2: The figure shows the open-set segmentation predictions obtained using the best hyperparameter configuration for OpenPCS, OpenGMM, and CoReSeg+Att for one test image of the Vaihingen dataset with all tested UUCs. Also, results with and without post-processing are presented on the right of the base prediction. The exhibited SPS configuration used for post-processing is the best one for each method on average. The used colors are: white for *impervious surfaces*; dark blue for *building*; light blue for *low vegetation*; green for *high vegetation*; yellow for *car*; and red for the OOD pixels.

Figure 7.3 presents the open-set predictions for the Potsdam dataset. OpenPCS and OpenGMM methods achieved the best results using DN-121+Att (Section 5.1) as the backbone for the Potsdam dataset. CoReSeg+Att which uses the CBAM attention mechanism and U-net+Att as the backbone obtained the best results. The post-processing fz_quick04 configuration (Section 5.5) got the best AUROC results with all methods.

The quality of the closed-set backbones segmentation prediction obtained for Potsdam is worse than the closed-set predictions for Vaihingen (Section 5.1). According to Vendramini et al. (2021), the quality of the closed-set segmentation relates to the quality of the open-set prediction. Producing better closed-set segmentation predictions imply in better open-set predictions.

Closed-set predictions for the Potsdam dataset using U-net and U-net+Att presented worse semantic segmentation and quantitative results compared to the other backbones.

Despite U-net+Att's lack of semantic consistency, Figure 7.3 shows that CoReSeg+Att managed to achieve the best results. CoReSeg+Att open-set predictions show better-preserved borders and have fewer artifacts. Also, for all methods, the post-processing presented the same consistent behavior observed for Vaihingen and improved the base results in all cases.

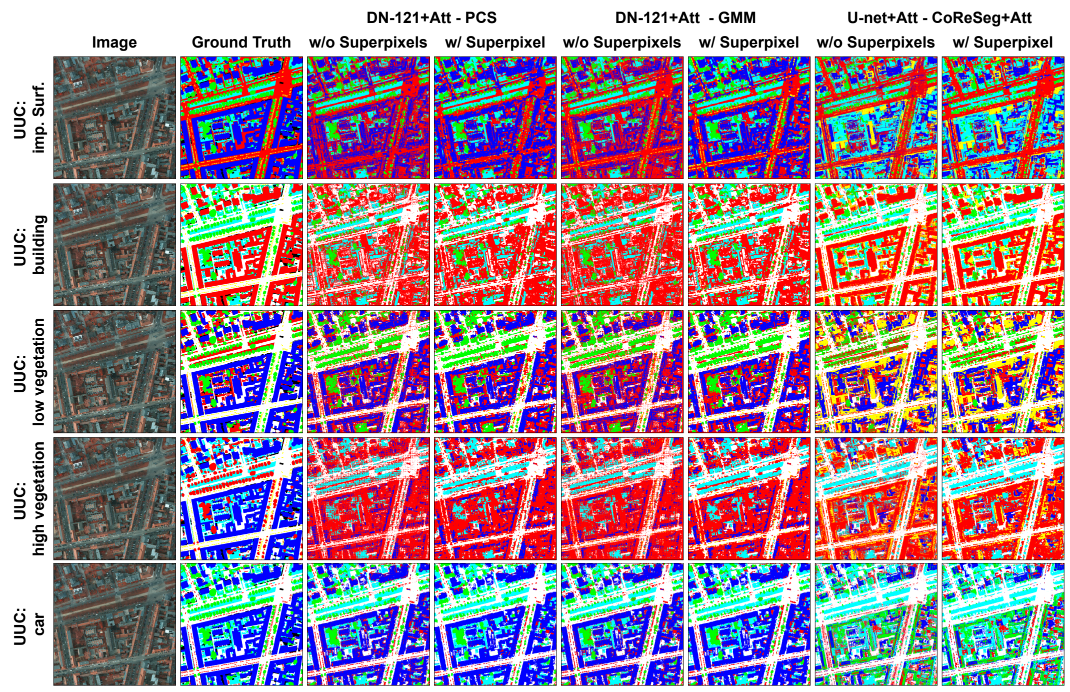


Figure 7.3: The figure shows the open-set segmentation predictions obtained using the best hyperparameter configuration for OpenPCS, OpenGMM, and CoReSeg+Att for one test image of the Potsdam dataset with all tested UUCs. Also, results with and without post-processing are presented on the right of the base prediction. The exhibited SPS configuration used for post-processing is the best one for each method on average. The used colors are: white for *impervious surfaces*; dark blue for *building*; light blue for *low vegetation*; green for *high vegetation*; yellow for *car*; and red for the OOD pixels.

8 Conclusion and Future Work

This work proposed and described two distinct methods for open-set semantic segmentation: OpenGMM as an extension of a known baseline method called OpenPCS (Oliveira et al., 2021) and a novel end-to-end fully convolutional model called CoReSeg (Nunes et al., 2022b). Besides that, this work proposed a general post-processing technique with a new superpixel merging procedure called FuSC.

To answer the three research questions proposed in Section 1.2, we performed exploratory tests on remote sensing image datasets (Section 5.3) and extensive quantitative and qualitative experimental evaluation comparing the proposed approaches with established literature baselines.

The two proposed methods for OSS improved the baseline results and showed better semantic consistency. Output scores of four distinct OSS methods - OpenPCS (Oliveira et al., 2021), OpenPCS++ (Martinez et al., 2021), OpenGMM, and CoReSeg - were post-processed using FuSC producing a refined open-set prediction that consistently improved the quantitative results and semantic consistency.

The proposed OpenGMM method improved the results for all six tested scenarios in the Vaihingen dataset and four out of six tested scenarios for the Potsdam dataset compared to baseline results. Complete results in Appendix B and compiled results in Section 7.1 present pieces of evidence that validate the proposal of OpenGMM as the answer for $\mathcal{RQ}_1$. We attribute the improvement shown by OpenGMM over OpenPCS and OpenPCS++ to its multimodal modeling capability for real-world data. The worse results in Potsdam are attributed mainly to OpenGMM's poorer performances on two UUCs: Low Vegetation and High Vegetation. For the two scenarios in that OpenGMM did not improve the results, the backbone used was DN-121 with and without the CBAM attention mechanism suggesting that DN-121 pre-trained backbones could not produce a good representation for Low and High Vegetation classes as UUC. It is worth mentioning that previous works already identified the instability of OSS algorithms in these two particular classes, possibly due to the large semantic intra-class variabilities (Oliveira et al., 2021; Vendramini et al., 2021).

CoReSeg covers the second research question $\mathcal{RQ}_2$ with its compiled results presented in Section 7.2 validating that assumption. CoReSeg was tested exclusively with U-net as the backbone with or without the CBAM attention mechanism. Due to the computational cost, we tested only CoReSeg+Att with U-net+Att for the Potsdam dataset. For the Vaihingen dataset, we also performed the CoReSeg variations with the CBAM attention mechanism used as part of the backbone and the model itself.

For the Potsdam dataset, CoReSeg+Att average improvement using U-net+Att as backbone was 0.0674 AUROC, which is a 10.37% improvement over the OpenGMM method results, and 0.0922 AUROC or 14.74% improvement over OpenPCS baseline result.

For the Vaihingen dataset, the results obtained by CoReSeg and its variations using the same backbone compared to OpenGMM showed improvements on average results between 0.0522 (7.57%) and 0.0886 (13.55%) AUROC. Comparing CoReSeg's best average results with the best baseline method resulted in a higher AUROC between 0.0932 (14.37%) and 0.1252 (19.30%). Comparing OSS methods under the same backbone is fairer and allows us to evaluate CoReSeg's prediction improvement over the baseline methods.

Among all methods, CoReSeg+Att performed better even with the not-so-fair comparison among its quantitative results with the best-performing combinations of other methods and backbone. In the comparison presented in Table 7.4 in Section 7.2, CoReSeg+Att outperformed OpenGMM in 0.0202 AUROC (2.68%) and OpenPCS in 0.0272 AUROC (3.64%) for the Vaihingen dataset. For the Potsdam dataset, CoReSeg+Att outperformed OpenPCS at 0.0200 AUROC (2.87%) and OpenGMM at 0.0360 AUROC (5.28%).

Qualitative results for CoReSeg (Section 7.4) showed a reduction of artifacts, better border preservation, and improved object delimitation or identification. In short, the open-set predictions produced by CoReSeg are more consistent and closer to ground truths.

Segmentation artifacts and ill-defined borders and objects are common issues in all tested OSS methods, both baseline and proposed. We proposed general superpixel post-processing in conjunction with the FuSC to answer the third research question $\mathcal{RQ}_3$. The post-processing produces refined open-set predictions with more semantic consistency and closer to the ground truths. Results presented in Section 7.3 corroborate the proposal as an answer to $\mathcal{RQ}_3$.

Appendix B presents the results for all forty test scenarios with eleven distinct post-processing configurations used to evaluate the possible impact of the post-processing on final results. The performed tests showed that post-processing improved results in thirty-nine of the forty performed tests and improved the results coupled with all tested methods. The size of the superpixels tampers with the quality of the results. In the only test case in which the post-processed results got worse, the superpixels were large to represent the underlying image.

For the Vaihingen dataset, post-processing produced improvements of 0.0408 AUROC (5,03%) when applied to CoReSeg+Att. For the Potsdam dataset, the largest post-processing improvement produced was over OpenPCS++ with WRN-50 as the backbone, 0.0360 AUROC (6.12%). Post-processed predictions reduce the identified issues and produced segmentations with more semantic consistency.

Defining the best superpixel algorithm and calibrating its hyperparameters is difficult and time-consuming. Superpixels' ability to represent the underlying image varies immensely with the selected hyperparameters. FuSC combines distinct superpixel algorithm outputs producing a final superpixel segmentation capable of representing the underlying image. Results suggest that FuSC final segmentation is a better representation than each individual input segmentation.

Post-processing using FuSC produced more consistent and stabler results, varying less among the different tested configurations. Suggesting that FuSC is less sensitive to hyperparameter selection, the final results for FuSC performed better on average than individual superpixel algorithms. Within the final results, post-processing with FuSC configuration produced the best overall results.

Essentially, we proposed one distinct approach to handle each identified research question. OpenGMM extended the base OpenPCS framework and achieved better results showing that it was possible to improve the state-of-the-art results for the datasets. A remarkable characteristic of OpenGMM is that, like OpenPCS, it could be adapted effortlessly into new backbones and frameworks.

CoReSeg established new state-of-the-art results for both datasets. Using the CBAM attention mechanism coupled CoReSeg allowed the method to improve its performance by roughly 4%. To the author's knowledge, CoReSeg is the first fully convolutional end-to-end method used to perform open-set segmentation in remote sensing images in literature.

As a final relevant collateral contribution of this work, we propose a novel taxonomy (Nunes et al., 2022a) to Open-set Recognition and Semantic Segmentation aiming to organize the literature and provide an understanding of the theoretical trends that guided the existing approaches that may influence future methods.

It is worth mentioning that Appendix C lists the articles published as results of this research. The appendix also lists the articles in progress or already submitted.

8.1 Limitations Found and Future Work

The proposed approaches could benefit from using different backbones adapting to other imaging datasets and domains. The proposed OSS methods could also segment time-series datasets. During this work, we tried some variations of the proposed techniques with little success.

We tried to use CoReSeg with two rural time-series datasets. We tried to pile the bands of all timestamps and handle the time-series datasets with a standard convolutional network. The first step for CoReSeg is to train the backbone, but closed-set results using the U-net were much lower than the baseline results (Martinez et al., 2021). In addition to the poor performance obtained by closed-set methods, during training of CoReSeg conditional reconstruction, we observed an issue due to CoReSeg stochastic batch and patch selection. The contrastive reconstruction training could not find enough overlapping areas with distinct closed-set classes due to the sparseness of the datasets.

The attempt to handle time-series datasets as convolutional image datasets seemed to be inadequate and the preliminary results were much worse than the baseline presented by Martinez et al. (2021). Likely, a model designed to handle time-series datasets works better. In this sense, CoReSeg could be modified to properly handle time-series datasets.

We also tried to use CoReSeg in a sparse urban dataset. In this experiment, CoReSeg performed better than in the time-series experiments, but CoReSeg's results could not match baseline results (Oliveira et al., 2021). In this case, we attribute the bad result to the U-net backbone that had many issues across all presented results. Adapting new closed-set backbones to work with CoReSeg may address the identified performance issues. Sparse datasets are more challenging for architectures like CoReSeg, requiring different strategies to be studied to deal with this type of data.

For OpenGMM, we tried to adapt HRNet Wang et al. (2020) with no success. All achieved results were worse than the ones obtained with the presented backbones. Adapting new backbones is needed to understand what characteristics are relevant to improve the models for OSS.

This work presents results for the Vaihingen and the Potsdam datasets. The used datasets are urban, densely labeled, and share the same set of

KKCs with the same available bands. The images have high definition and are from highly organized cities. Future works must extend the model to segment datasets with distinct characteristics.

Bibliography

Achanta, R., Márquez-Neila, P., Fua, P., and Süsstrunk, S. (2018). Scale-adaptive superpixels. In *Color and Imaging Conference*, volume 2018, pages 1–6. Society for Imaging Science and Technology.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. (2012). Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282.

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495.

Bendale, A. and Boult, T. (2015). Towards open world recognition. In *CVPR*, pages 1893–1902.

Bendale, A. and Boult, T. E. (2016). Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1563–1572.

Bentley, R. A., O'Brien, M. J., and Brock, W. A. (2014). Mapping collective behavior in the big-data era. *Behavioral and Brain Sciences*, 37(1):63.

Bergh, M. V. d., Boix, X., Roig, G., Capitani, B. d., and Gool, L. V. (2012). Seeds: Superpixels extracted via energy-driven sampling. In *European conference on computer vision*, pages 13–26. Springer.

Bevandić, P., Krešo, I., Oršić, M., and Šegvić, S. (2022). Dense open-set recognition based on training with noisy negative images. *Image and Vision Computing*, page 104490.

Bharadwaj, R., Jaswal, G., Nigam, A., and Tiwari, K. (2022). Mobile based human identification using forehead creases: Application and assessment under covid-19 masked face scenarios. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3693–3701.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brilhador, A., Gutoski, M., Lazzaretti, A. E., and Lopes, H. S. . (2021). A comparative study for open set semantic segmentation methods. In Filho, C. J. A. B., Siqueira, H. V., Ferreira, D. D., Bertol, D. W., and ao de Oliveira, R. C. L., editors, *Anais do 15 Congresso Brasileiro de Inteligência Computacional*, pages 1–8, Joinville, SC. SBIC.

Bruce, L. M., Koger, C. H., and Li, J. (2002). Dimensionality reduction of hyperspectral data using discrete wavelet transform feature extraction. *IEEE Transactions on geoscience and remote sensing*, 40(10):2331–2338.

Buyssens, P., Toutain, M., Elmoataz, A., and Lézoray, O. (2014). Eikonal-based vertices growing and iterative seeding for efficient graph-based segmentation. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 4368–4372. IEEE.

Cen, J., Yun, P., Cai, J., Wang, M. Y., and Liu, M. (2021). Deep metric learning for open world semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15333–15342.

Chan, R., Rottmann, M., and Gottschalk, H. (2021). Entropy maximization and meta classification for out-of-distribution detection in semantic segmentation. In *Proceedings of the ieee/cvf international conference on computer vision*, pages 5128–5137.

Chaudhari, S., Mithal, V., Polatkan, G., and Ramanath, R. (2021). An attentive survey of attention models. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 12(5):1–32.

Cheng, G., Xie, X., Han, J., Guo, L., and Xia, G.-S. (2020). Remote sensing image scene classification meets deep learning: Challenges, methods, benchmarks, and opportunities. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:3735–3756.

Cortes, C. and Vapnik, V. (1995). Support vector machine. *Machine learning*, 20(3):273–297.

Cui, Z., Longshi, W., and Wang, R. (2020). Open set semantic segmentation with statistical test and adaptive threshold. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE.

da Silva, C. C., Nogueira, K., Oliveira, H. N., and dos Santos, J. A. (2020). Towards open-set semantic segmentation of aerial images. In *2020 IEEE Latin American GRSS & ISPRS Remote Sensing Conference (LAGIRS)*, pages 16–21. IEEE.

de Santana Correia, A. and Colombini, E. L. (2022). Attention, please! a survey of neural attention models in deep learning. *Artificial Intelligence Review*, pages 1–88.

Dong, H., Chen, Z., Yuan, M., Xie, Y., Zhao, J., Yu, F., Dong, B., and Zhang, L. (2022). Region-aware metric learning for open world semantic segmentation via meta-channel aggregation. *arXiv preprint arXiv:2205.08083*.

Drozdzal, M., Vorontsov, E., Chartrand, G., Kadoury, S., and Pal, C. (2016). The importance of skip connections in biomedical image segmentation. In *Deep learning and data labeling for medical applications*, pages 179–187. Springer.

Duda, R. O., Hart, P. E., and Stork, D. G. (1973). *Pattern classification and scene analysis*, volume 3. Wiley New York.

Dumoulin, V., Perez, E., Schucher, N., Strub, F., Vries, H. d., Courville, A., and Bengio, Y. (2018). Feature-wise transformations. *Distill*, 3(7):e11.

Elkhateeb, E., Soliman, H., Atwan, A., Elmogy, M., Kwak, K.-S., and Mekky, N. (2021). A novel coarse-to-fine sea-land segmentation technique based on superpixel fuzzy c-means clustering and modified chan-vese model. *IEEE Access*, 9:53902–53919.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181.

Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., Martinez-Gonzalez, P., and Garcia-Rodriguez, J. (2018). A survey on deep learning techniques for image and video semantic segmentation. *Applied Soft Computing*, 70:41–65.

Gawlikowski, J., Saha, S., Kruspe, A., and Zhu, X. X. (2022). An advanced dirichlet prior network for out-of-distribution detection in remote sensing. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–19.

Ge, Z., Demyanov, S., Chen, Z., and Garnavi, R. (2017). Generative openmax for multi-class open set classification. *arXiv preprint arXiv:1707.07418*.

Geng, C., Huang, S.-j., and Chen, S. (2020). Recent advances in open set recognition: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3614–3631.

Gondara, L. (2016). Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th international conference on data mining workshops (ICDMW)*, pages 241–246. IEEE.

Grcić, M., Bevandić, P., and Šegvić, S. (2020). Dense open-set recognition with synthetic outliers generated by real nvp. *arXiv preprint arXiv:2011.11094*.

Grcić, M., Bevandić, P., and Šegvić, S. (2021). Dense anomaly detection by robust learning on synthetic negative data. *arXiv preprint arXiv:2112.12833*.

Grcić, M., Bevandić, P., and Šegvić, S. (2022). Densehybrid: Hybrid anomaly detection for dense open-set recognition. *arXiv preprint arXiv:2207.02606*.

Guo, Y., Camporese, G., Yang, W., Sperduti, A., and Ballan, L. (2021). Conditional variational capsule network for open set recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 103–111.

Hanin, B. (2018). Which neural net architectures give rise to exploding and vanishing gradients? *Advances in neural information processing systems*, 31.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hendrycks, D., Mazeika, M., and Dietterich, T. (2018). Deep anomaly detection with outlier exposure. *arXiv preprint arXiv:1812.04606*.

Hong, J., Li, W., Han, J., Zheng, J., Fang, P., Harandi, M., and Petersson, L. (2022). Goss: Towards generalized open-set semantic segmentation. *arXiv preprint arXiv:2203.12116*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hu, Z., Zou, Q., and Li, Q. (2015). Watershed superpixel. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 349–353. IEEE.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

Huang, H., He, R., Sun, Z., Tan, T., et al. (2018). Introvae: Introspective variational autoencoders for photographic image synthesis. *Advances in neural information processing systems*, 31.

Jadon, S. (2020). A survey of loss functions for semantic segmentation. In *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–7. IEEE.

Jetley, S., Lord, N. A., Lee, N., and Torr, P. (2018). Learn to pay attention. In *International Conference on Learning Representations*.

Ji, J., Lu, X., Luo, M., Yin, M., Miao, Q., and Liu, X. (2020). Parallel fully convolutional network for semantic segmentation. *IEEE Access*, 9:673–682.

Jolliffe, I. T. (1990). Principal component analysis: a beginner's guide—i. introduction and application. *Weather*, 45(10):375–382.

Kang, J., Wang, Z., Zhu, R., Sun, X., Fernandez-Beltran, R., and Plaza, A. (2021). Picoco: Pixelwise contrast and consistency learning for semisupervised building footprint segmentation. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:10548–10559.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1):7–15.

Kong, S. and Ramanan, D. (2021). Opengan: Open-set recognition via open data generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 813–822.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50.

Kumar, M., Saxena, R., et al. (2013). Algorithm and technique on various edge detection: A survey. *Signal & Image Processing*, 4(3):65.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

Levinshtein, A., Stere, A., Kutulakos, K. N., Fleet, D. J., Dickinson, S. J., and Siddiqi, K. (2009). Turbopixels: Fast superpixels using geometric flows. *IEEE transactions on pattern analysis and machine intelligence*, 31(12):2290–2297.

Li, Z. and Chen, J. (2015). Superpixel segmentation using linear spectral clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1356–1363.

Liu, M.-Y., Tuzel, O., Ramalingam, S., and Chellappa, R. (2011). Entropy rate superpixel segmentation. In *CVPR 2011*, pages 2097–2104. IEEE.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

Machairas, V., Faessel, M., Cárdenas-Peña, D., Chabardes, T., Walter, T., and Decenciere, E. (2015). Waterpixels. *IEEE Transactions on Image Processing*, 24(11):3707–3716.

MAHALANOBIS, P. (1936). On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55.

Makhzani, A. and Frey, B. (2013). K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*.

Martinez, J. A. C., Oliveira, H., dos Santos, J. A., and Feitosa, R. Q. (2021). Open set semantic segmentation for multitemporal crop recognition. *IEEE Geoscience and Remote Sensing Letters*, 19:1–5.

Mehta, S., Hajishirzi, H., and Rastegari, M. (2020). Dicenet: Dimension-wise convolutions for efficient networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Melas-Kyriazi, L. and Manrai, A. K. (2021). Pixmatch: Unsupervised domain adaptation via pixelwise consistency training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12435–12445.

Merriam-Webster (2022). Attention.

Minaee, S., Boykov, Y. Y., Porikli, F., Plaza, A. J., Kehtarnavaz, N., and Terzopoulos, D. (2021). Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*.

Morerio, P., Marcenaro, L., and Regazzoni, C. S. (2014). A generative superpixel method. In *17th International Conference on Information Fusion (FUSION)*, pages 1–7. IEEE.

Ng, A. et al. (2011). Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19.

Nunes, I., Oliveira, H., Pereira, M. B., Santos, J. A. d., and Poggi, M. (2022a). Deep open-set segmentation in visual learning. In *Proceedings...* Conference on Graphics, Patterns and Images, 35. (SIBGRAPI).

Nunes, I., Pereira, M. B., Oliveira, H., dos Santos, J. A., and Poggi, M. (2022b). Conditional reconstruction for open-set semantic segmentation. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 946–950.

Oliveira, H., Silva, C., Machado, G. L., Nogueira, K., and dos Santos, J. A. (2021). Fully convolutional open set segmentation. *Machine Learning*, pages 1–52.

Oza, P. and Patel, V. M. (2019). C2ae: Class conditioned auto-encoder for open-set recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2307–2316.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Pradeep, K., Kamalavasan, K., Natheesan, R., and Pasqual, A. (2018). Edgenet: Squeezenet like convolution neural network on embedded fpga. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 81–84. IEEE.

Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer.

Ratajczak, R., Crispim, C., Fervers, B., Faure, E., and Tougne, L. (2020). Semantic segmentation post-processing with colorized pairwise potentials and deep edges. In *2020 Tenth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pages 1–6. IEEE.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

Roy, A. G., Navab, N., and Wachinger, C. (2018). Concurrent spatial and channel 'squeeze & excitation'in fully convolutional networks. In *International conference on medical image computing and computer-assisted intervention*, pages 421–429. Springer.

Rubio, A., Yu, L., Simo-Serra, E., and Moreno-Noguer, F. (2016). Bass: boundary-aware superpixel segmentation. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2824–2829. IEEE.

Safavian, S. R. and Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674.

Saito, K., Hu, P., Darrell, T., and Saenko, K. (2021). Learning to detect every thing in an open world. *arXiv preprint arXiv:2112.01698*.

Scheirer, W. J., de Rezende Rocha, A., Sapkota, A., and Boult, T. E. (2012). Toward open set recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(7):1757–1772.

Scheirer, W. J., Jain, L. P., and Boult, T. E. (2014). Probability models for open set recognition. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2317–2324.

Shwartz-Ziv, R. and Tishby, N. (2017). Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. (2017). Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*.

Sun, J., Wang, X., Xiong, N., and Shao, J. (2018). Learning sparse representation with variational auto-encoder for anomaly detection. *IEEE Access*, 6:33353–33361.

Sun, X., Yang, Z., Zhang, C., Ling, K.-V., and Peng, G. (2020). Conditional gaussian distribution learning for open set recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13480–13489.

Tan, M. and Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR.

Tao, A., Sapra, K., and Catanzaro, B. (2020). Hierarchical multi-scale attention for semantic segmentation. *arXiv preprint arXiv:2005.10821*.

Toffler, A. (1970). Future shock, 1970. *Sydney. Pan*.

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., and the scikit-image contributors (2014). scikit-image: image processing in Python. *PeerJ*, 2:e453.

Vedaldi, A. and Soatto, S. (2008). Quick shift and kernel methods for mode seeking. In *European conference on computer vision*, pages 705–718. Springer.

Vendramini, M., Oliveira, H., Machado, A., and dos Santos, J. A. (2021). Opening Deep Neural Networks With Generative Models. In *ICIP*, pages 1314–1318. IEEE.

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., et al. (2020). Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3349–3364.

Wang, M., Liu, X., Soomro, N. Q., Han, G., and Liu, W. (2019). Content-sensitive superpixel segmentation via self-organization-map neural network. *Journal of Visual Communication and Image Representation*, 63:102572.

Woo, S., Park, J., Lee, J.-Y., and Kweon, I. S. (2018). Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19.

Xie, S. and Tu, Z. (2015). Holistically-nested edge detection. In *Proceedings of the IEEE international conference on computer vision*, pages 1395–1403.

Yan, S., Zhou, J., Xie, J., Zhang, S., and He, X. (2021). An em framework for online incremental learning of semantic segmentation. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 3052–3060.

Yoshihashi, R., Shao, W., Kawakami, R., You, S., Iida, M., and Naemura, T. (2019). Classification-reconstruction learning for open-set recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4016–4025.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association.

Zhang, H., Jiang, K., Zhang, Y., Li, Q., Xia, C., and Chen, X. (2014). Discriminative feature learning for video semantic segmentation. In *2014 International Conference on Virtual Reality and Visualization*, pages 321–326. IEEE.

Zhang, H., Li, F., Liu, S., Zhang, L., Su, H., Zhu, J., Ni, L. M., and Shum, H.-Y. (2022). Dino: Detr with improved denoising anchor boxes for end-to-end object detection. *arXiv preprint arXiv:2203.03605*.

Zhang, J., Wang, P., Gong, F., Zhu, H., and Chen, N. (2020). Content-based superpixel segmentation and matching using its region feature descriptors. *IEICE Transactions on Information and Systems*, 103(8):1888–1900.

Zhou, C., Liu, F., Gong, C., Liu, T., Han, B., and Cheung, W. (2021). Krada: Known-region-aware domain alignment for open world semantic segmentation. *arXiv preprint arXiv:2106.06237*.

Zhu, Q., Yeh, M.-C., Cheng, K.-T., and Avidan, S. (2006). Fast human detection using a cascade of histograms of oriented gradients. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 1491–1498. IEEE.

A
Fusing Superpixels for Semantic Consistency - Code

Listing A.1 provides the complete code used for FuSC, implemented in Python 3.8. The comments detail the functioning of the method and the algorithmic complexity using big O notation.

The official implementation of all proposed approaches is available at `https://github.com/iannunes`.

```python
import scipy as sp
from scipy.spatial import distance

# method used to merge two different superpixel segmentations.
# s1 and s2: a 2D array mapping the superpixel segmentations. Each pixel in the represented in the segmentation array is set with the number of the respective segment
# img: the segmented image
# min_size: the minimum size threshold for the merged segmentation
def join_segmentations(s1, s2, img, min_size): #O(n)
    assert s1.shape == s2.shape
    counter = -1
    ret = np.zeros(s1.shape, dtype=int)
    final_labels = {}

    # merges two different segmentations. setting sequential labels to each intersection between s1 and s2
    for i in range(0, s1.shape[0]):    # O(n) - assuming constant time for dict operations
        for j in range(0, s1.shape[1]):
            label1 = s1[i,j]
            label2 = s2[i,j]
            if label1 not in final_labels:
                final_labels[label1]={}
            if label2 not in final_labels[label1]:
                final_labels[label1][label2] = counter
                counter-=1
            ret[i,j]=final_labels[label1][label2]

    ret = -1 * ret
    counter = -1 * counter
    existing_areas={}

    # ensure connectivity for each segment. O(n)
    for i in range(0, ret.shape[0]):
        for j in range(0, ret.shape[1]):
            label = ret[i,j]
            if label>0:
                if label in existing_areas:
                    ret[ret==label] = counter
                    # all operations are O(n) in the worst case when all labels must be replaced. This case is actually unfeasible.
                    label = counter
                    counter + 1
                existing_areas[label]=True
                ret = track_continuos(ret, i, j, label)
                # all operations are O(9n) in the worst case when all labels must be replaced. This case is actually unfeasible.

    ret = -1 * ret
    neighbors = {}
    # get the neighborhood for each segment
    for i in range(0, ret.shape[0]):         # O(n)
        for j in range(0, ret.shape[1]):
            label1 = ret[i,j]
            if label1 not in neighbors:
                neighbors[label1]={}

            for k in range(i-1,i+2):         # cte
                for h in range(j-1,j+2):
                    if (k==h and k==0) or (k<0 or h<0 or k>=ret.shape[0] or h>=ret.shape[1]):
                        continue

                    label2 = ret[k,h]
                    if label1 != label2:
                        if label2 not in neighbors:
                            neighbors[label2]={}
                        neighbors[label1][label2]=True
                        neighbors[label2][label1]=True

    return merge_superpixels(ret, neighbors, img, min_size)

# main procedure that merges neighboring areas if the minimum pixel count is not respected.
# sps: the 2D mapping superpixel segmentation
# neighbors: a list of each segment and its neighbors
```

```python
# img: the segmented image
# min_size: the minimum size threshold for the merged segmentation
def merge_superpixels(sps, neighbors, img, min_size):
# the complexity for the procedure is 3O(n)+ O(n * cte * minimum size^2) + 2O(n) = O(n * minimum size^2). As some assumed constants depend on the minimum size, we can say that the procedure is pseudo-polynomial.
    sps_sizes={}

    img = np.array(img, dtype = float)

    sps_uniques = np.unique(sps, return_counts = True) # O(n)

    sps_processed = {}
    flatten_superpixels = {}

    # pixel count
    for i in range(0,len(sps_uniques[0])):   # O(n)
        sps_sizes[sps_uniques[0][i]] = sps_uniques[1][i]

    # populate a dictionary with image pixels for each segment
    for i in range(0, sps.shape[0]):         # O(n)
        for j in range(0, sps.shape[1]):
            label = sps[i,j]
            if label not in flatten_superpixels:
                flatten_superpixels[label] = []
            flatten_superpixels[label].append(img[i,j])

    for key in flatten_superpixels:
        flatten_superpixels[key] = np.array(flatten_superpixels[key])

    sps_mapping = OrderedDict()

    # for each segment with less pixels of minimum pixel count
    # compare to all neighbors and merge with closest one
    for key in flatten_superpixels:
        # O(n) as superpixel segmentation is an over segmentation of the image, the expected number of segments is n/cte implying in O(n/cte) = O(n) executions of the for loop
        if key in sps_mapping:
            continue
        while sps_sizes[key] < min_size:

            min_dist = 9999999999999999
            final_smaller_key = -1
            final_bigger_key = -1

            # closest neighbor search
            for n_key in neighbors[key]:
            # worst case scenario is n/2 iterations - O(n)
            # assuming that superpixel segmentations produces segments approximately with the same pixel count. And assuming the maximum number of possible neighbors for each segment, the number of iterations are 2 x minimum size + 6. We can consider as O(cte*min_size) = O (cte)

                if n_key in sps_mapping:
                    continue
                smaller_sps_label = n_key
                bigger_sps_label = key
                if flatten_superpixels[key].shape[0] < flatten_superpixels[n_key].shape[0]:
                    smaller_sps_label = key
                    bigger_sps_label = n_key

                if final_smaller_key < 0:
                    final_smaller_key = smaller_sps_label
                    final_bigger_key = bigger_sps_label

                x = flatten_superpixels[smaller_sps_label]
                data = flatten_superpixels[bigger_sps_label]

                # computes the distance between the 2 segments
                dist = mahalanobis(x = np.mean(x,axis = 0), data = data)
                # the complexity of mahalanobis is the greatest between O((d**4)*((log d)**2)) and O(n*(d**2)), for n the number of elements in the biggest segment and d the number of features.
                # in our particular case, we have few features and the probable number of elements in the segment is cte*minimum size. The final complexity for our case is the greatest between O((cte**4)*((log cte)**2)) and O((cte*minimum size)*(cte**2)) = O(minimum size)

                if min_dist > dist:
                    min_dist = dist
                    final_smaller_key = smaller_sps_label
                    final_bigger_key = bigger_sps_label

            # create the merging mapping. In the end, the
```

```
mapping is executed to produces the final segmentation. O(cte)
            sps_mapping[final_smaller_key] = final_bigger_key
            # compute the size of the merged segment. O(cte)
            sps_sizes[final_bigger_key] = sps_sizes[final_smaller_key] + sps_sizes[final_bigger_key]

            for n_key in neighbors[final_smaller_key]:
                # as discussed before, the probable case is O(2*min_size) = O(cte)
                if final_smaller_key in neighbors[n_key]:
                    del neighbors[n_key][final_smaller_key]
                neighbors[final_bigger_key][n_key]=True

            if final_smaller_key in neighbors[final_bigger_key]:
                del neighbors[final_bigger_key][final_smaller_key]
            if final_bigger_key in neighbors[final_bigger_key]:
                del neighbors[final_bigger_key][final_bigger_key]

            key = final_bigger_key

    sps = merge_mapped(sps, sps_mapping) #O(number of pixels)
    sps = relabel(sps)                   #O(number of pixels)

    return sps

# auxiliary function to execute the relabel according to the intersection between the segmentations
# sps: the 2D mapping superpixel segmentation
# sps_mapping: a list maping merged superpixels
def merge_mapped(sps, sps_mapping):  # O(number of pixels)
    for i in range(0, sps.shape[0]):
        for j in range(0, sps.shape[1]):
            label = sps[i,j]
            while label in sps_mapping:
                sps[i,j] = sps_mapping[label]
                label=sps[i,j]
    return sps


# ensure that numbered lables are between 1 and n
# sps: the 2D mapping superpixel segmentation
def relabel(sps):  # O(number of pixels)
    counter = -1
    for i in range(0, sps.shape[0]):
        for j in range(0, sps.shape[1]):
            label = sps[i,j]
            if label<0:
                continue
            sps[sps==label] = counter
            counter -= 1
    return sps*-1

# recursive procedure that selects a continuous area and relabel it
def track_continuos(input_array, i, j, label, rec=False):
    if input_array[i, j] != label:
        return input_array

    input_array[i, j] = input_array[i, j]*-1
    for k in range(i-1,i+2):
        for h in range(j-1,j+2):
            if (k==h and k==0) or (k<0 or h<0 or k>=input_array.shape[0] or h>=input_array.shape[1]):
                continue
            input_array = track_continuos(input_array, k, h, label, rec=True)
    return input_array

# compute the mahalanobis distance between the 2 distributions.
# x: image pixels of a superpixel
# data: image pixels of a superpixel
# cov: pre calculated covariance matrix
def mahalanobis(x=None, data=None, cov=None):
    # O((n**4)*((log n)**2)) or O(N*(n**2))
    """Compute the Mahalanobis Distance between each row of x and the data
    x    : vector or matrix of data with, say, p columns.
    data : ndarray of the distribution from which Mahalanobis distance of each observation of x is to be computed.
    cov  : covariance matrix (p x p) of the distribution. If None, will be computed from data.
    """
    # N = number of data points in data
    # n = number of features in data

    x_minus_mu = x - np.mean(data,axis=0)
    if not cov:
        cov = np.cov(data.T)         # O(N*(n**2))
    inv_covmat = sp.linalg.inv(cov) # O((n**4)*((log n)**2))
    left_term = np.dot(x_minus_mu, inv_covmat) # O(n)
    m = np.dot(left_term, x_minus_mu.T)
    if type (m) is np.float64:
        return m
    return m.diagonal()
```

Listing A.1: FuSC implementation.

B
Complete Experimental Results

This appendix presents tables B.1 to B.40 with the complete set of results obtained during the performed tests for this work. Results presented in this appendix were used to compile all tables showed in in Chapter 7.

Each table shows one of the forty distinct test scenarios combining backbone, method, and dataset. For each tested scenario, twelve results are listed. The first line is the base results produced by the method and backbone without post-processing. The next lines present the results of the eleven distinct post-processing configurations selected in Chapter 6.

The notation *+Att* as the suffix of the name of the method or the backbone indicates the use of the CBAM attention mechanism as presented in Chapter 4 and Section 5.1.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Vaihingen	DN-121	OpenGMM	.851	.866	.641	.697	.713	$.7536 \pm .0996$	-
Vaihingen	DN-121	OpenGMM	.867	.875	.648	.714	.725	$.7658 \pm .1005$	fz_quick02
Vaihingen	DN-121	OpenGMM	.869	.875	.646	.718	.726	$.7668 \pm .1010$	fz_quick04
Vaihingen	DN-121	OpenGMM	.868	.875	.649	.717	.726	$.7670 \pm .1000$	fz_slic02
Vaihingen	DN-121	OpenGMM	.865	.873	.646	.710	.725	$.7638 \pm .1006$	fz_slic04
Vaihingen	DN-121	OpenGMM	.866	.872	.649	.706	.720	$.7626 \pm .1007$	fz01
Vaihingen	DN-121	OpenGMM	.864	.871	.650	.712	.723	$.7640 \pm .0985$	fz02
Vaihingen	DN-121	OpenGMM	.840	.849	.631	.720	.732	$.7544 \pm .0911$	fz04
Vaihingen	DN-121	OpenGMM	.867	.873	.649	.709	.719	$.7634 \pm .1009$	fz05
Vaihingen	DN-121	OpenGMM	.865	.871	.650	.713	.719	$.7636 \pm .0991$	fz06
Vaihingen	DN-121	OpenGMM	.867	.875	.639	.715	.706	$.7604 \pm .1052$	quick02
Vaihingen	DN-121	OpenGMM	.863	.872	.644	.710	.719	$.7616 \pm .1010$	slic06

Table B.1: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with DN-121 as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Vaihingen	DN-121	OpenPCS	.842	.866	.614	.733	.678	.7466 ± .1070	-
Vaihingen	DN-121	OpenPCS	.868	.879	.624	.757	.685	.7626 ± .1117	fz_quick02
Vaihingen	DN-121	OpenPCS	.871	.880	.624	.761	.686	.7644 ± .1125	fz_quick04
Vaihingen	DN-121	OpenPCS	.869	.879	.627	.759	.682	.7632 ± .1115	fz_slic02
Vaihingen	DN-121	OpenPCS	.862	.877	.624	.753	.683	.7598 ± .1102	fz_slic04
Vaihingen	DN-121	OpenPCS	.867	.876	.627	.752	.679	.7602 ± .1109	fz01
Vaihingen	DN-121	OpenPCS	.867	.875	.629	.756	.682	.7618 ± .1095	fz02
Vaihingen	DN-121	OpenPCS	.851	.857	.621	.758	.696	.7566 ± .1013	fz04
Vaihingen	DN-121	OpenPCS	.869	.878	.627	.756	.674	.7608 ± .1128	fz05
Vaihingen	DN-121	OpenPCS	.868	.876	.631	.758	.677	.7620 ± .1103	fz06
Vaihingen	DN-121	OpenPCS	.871	.882	.616	.760	.658	.7574 ± .1207	quick02
Vaihingen	DN-121	OpenPCS	.859	.876	.620	.751	.679	.7570 ± .1112	slic06

Table B.2: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with DN-121 as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg. AUROC**	**SPS config.**
			0	**1**	**2**	**3**	**4**		
Vaihingen	DN-121	OpenPCS++	.618	.703	.625	.729	.663	.6676 ± .0483	-
Vaihingen	DN-121	OpenPCS++	.635	.722	.634	.753	.670	.6828 ± .0531	fz_quick02
Vaihingen	DN-121	OpenPCS++	.637	.721	.634	.755	.670	.6834 ± .0532	fz_quick04
Vaihingen	DN-121	OpenPCS++	.633	.725	.631	.747	.673	.6818 ± .0528	fz_slic02
Vaihingen	DN-121	OpenPCS++	.629	.721	.629	.737	.674	.6780 ± .0504	fz_slic04
Vaihingen	DN-121	OpenPCS++	.634	.720	.633	.741	.671	.6798 ± .0493	fz01
Vaihingen	DN-121	OpenPCS++	.636	.719	.634	.743	.669	.6802 ± .0491	fz02
Vaihingen	DN-121	OpenPCS++	.631	.698	.621	.752	.665	.6734 ± .0534	fz04
Vaihingen	DN-121	OpenPCS++	.635	.723	.633	.749	.669	.6818 ± .0523	fz05
Vaihingen	DN-121	OpenPCS++	.635	.723	.632	.749	.665	.6808 ± .0528	fz06
Vaihingen	DN-121	OpenPCS++	.635	.727	.623	.750	.657	.6784 ± .0568	quick02
Vaihingen	DN-121	OpenPCS++	.624	.717	.628	.743	.670	.6764 ± .0529	slic06

Table B.3: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with DN-121 as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Vaihingen	DN-121+Att	OpenGMM	.900	.845	.621	.502	.808	.7352 ± .1673	-
Vaihingen	DN-121+Att	OpenGMM	.914	.858	.637	.500	.826	.7470 ± .1729	fz_quick02
Vaihingen	DN-121+Att	OpenGMM	.915	.858	.637	.501	.826	.7474 ± .1728	fz_quick04
Vaihingen	DN-121+Att	OpenGMM	.914	.858	.636	.496	.825	.7458 ± .1744	fz_slic02
Vaihingen	DN-121+Att	OpenGMM	.912	.856	.632	.499	.823	.7444 ± .1729	fz_slic04
Vaihingen	DN-121+Att	OpenGMM	.913	.855	.638	.493	.821	.7440 ± .1740	fz01
Vaihingen	DN-121+Att	OpenGMM	.911	.852	.639	.494	.823	.7438 ± .1727	fz02
Vaihingen	DN-121+Att	OpenGMM	.890	.834	.624	.498	.825	.7342 ± .1661	fz04
Vaihingen	DN-121+Att	OpenGMM	.913	.857	.639	.492	.820	.7442 ± .1744	fz05
Vaihingen	DN-121+Att	OpenGMM	.911	.853	.638	.494	.821	.7434 ± .1728	fz06
Vaihingen	DN-121+Att	OpenGMM	.915	.861	.629	.495	.809	.7418 ± .1749	quick02
Vaihingen	DN-121+Att	OpenGMM	.910	.854	.630	.498	.819	.7422 ± .1723	slic06

Table B.4: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with DN-121+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg.	SPS
			0	1	2	3	4	AUROC	config.
Vaihingen	DN-121+Att	OpenPCS	.894	.820	.622	.541	.796	.7346 ± .1472	-
Vaihingen	DN-121+Att	OpenPCS	.915	.838	.640	.538	.823	.7508 ± .1560	fz_quick02
Vaihingen	DN-121+Att	OpenPCS	.916	.838	.641	.538	.825	.7516 ± .1563	fz_quick04
Vaihingen	DN-121+Att	OpenPCS	.915	.837	.639	.535	.823	.7498 ± .1571	fz_slic02
Vaihingen	DN-121+Att	OpenPCS	.911	.835	.635	.535	.817	.7466 ± .1557	fz_slic04
Vaihingen	DN-121+Att	OpenPCS	.913	.834	.641	.532	.816	.7472 ± .1560	fz01
Vaihingen	DN-121+Att	OpenPCS	.911	.832	.643	.533	.820	.7478 ± .1550	fz02
Vaihingen	DN-121+Att	OpenPCS	.892	.815	.628	.533	.832	.7400 ± .1521	fz04
Vaihingen	DN-121+Att	OpenPCS	.915	.837	.643	.535	.817	.7494 ± .1557	fz05
Vaihingen	DN-121+Att	OpenPCS	.913	.834	.643	.534	.819	.7486 ± .1553	fz06
Vaihingen	DN-121+Att	OpenPCS	.918	.844	.636	.535	.811	.7488 ± .1581	quick02
Vaihingen	DN-121+Att	OpenPCS	.910	.834	.633	.535	.814	.7452 ± .1554	slic06

Table B.5: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with DN-121+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Vaihingen	DN-121+Att	OpenPCS++	.694	.647	.638	.611	.725	.6630 ± .0458	-
Vaihingen	DN-121+Att	OpenPCS++	.726	.654	.647	.624	.733	.6768 ± .0494	fz_quick02
Vaihingen	DN-121+Att	OpenPCS++	.725	.656	.649	.628	.734	.6784 ± .0479	fz_quick04
Vaihingen	DN-121+Att	OpenPCS++	.726	.655	.647	.615	.735	.6756 ± .0524	fz_slic02
Vaihingen	DN-121+Att	OpenPCS++	.718	.654	.646	.616	.731	.6730 ± .0493	fz_slic04
Vaihingen	DN-121+Att	OpenPCS++	.721	.651	.649	.620	.734	.6750 ± .0497	fz01
Vaihingen	DN-121+Att	OpenPCS++	.716	.648	.651	.623	.736	.6748 ± .0485	fz02
Vaihingen	DN-121+Att	OpenPCS++	.691	.634	.628	.620	.743	.6632 ± .0527	fz04
Vaihingen	DN-121+Att	OpenPCS++	.727	.654	.645	.623	.733	.6764 ± .0503	fz05
Vaihingen	DN-121+Att	OpenPCS++	.722	.651	.650	.622	.732	.6754 ± .0486	fz06
Vaihingen	DN-121+Att	OpenPCS++	.732	.658	.640	.624	.714	.6736 ± .0471	quick02
Vaihingen	DN-121+Att	OpenPCS++	.715	.655	.642	.616	.731	.6718 ± .0491	slic06

Table B.6: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with DN-121+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Vaihingen	U-net	CoReSeg	.880	.817	.680	.658	.676	.7422 ± .0999	-
Vaihingen	U-net	CoReSeg	.915	.847	.686	.683	.705	.7672 ± .1070	fz_quick02
Vaihingen	U-net	CoReSeg	.916	.850	.684	.688	.706	.7688 ± .1072	fz_quick04
Vaihingen	U-net	CoReSeg	.916	.848	.688	.679	.703	.7668 ± .1082	fz_slic02
Vaihingen	U-net	CoReSeg	.912	.843	.688	.674	.704	.7642 ± .1068	fz_slic04
Vaihingen	U-net	CoReSeg	.914	.847	.687	.680	.699	.7654 ± .1079	fz01
Vaihingen	U-net	CoReSeg	.911	.849	.683	.685	.701	.7658 ± .1068	fz02
Vaihingen	U-net	CoReSeg	.885	.842	.646	.670	.712	.7510 ± .1065	fz04
Vaihingen	U-net	CoReSeg	.916	.851	.686	.682	.696	.7662 ± .1096	fz05
Vaihingen	U-net	CoReSeg	.912	.852	.682	.686	.698	.7660 ± .1082	fz06
Vaihingen	U-net	CoReSeg	.917	.854	.681	.681	.656	.7578 ± .1191	quick02
Vaihingen	U-net	CoReSeg	.909	.842	.687	.671	.702	.7622 ± .1067	slic06

Table B.7: The table shows the base open-set prediction quantitative results obtained by combining using CoReSeg with U-net as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg. AUROC**	**SPS config.**
			0	**1**	**2**	**3**	**4**		
Vaihingen	U-net	OpenGMM	.841	.527	.641	.712	.547	.6536 ± .1285	-
Vaihingen	U-net	OpenGMM	.868	.524	.653	.713	.541	.6598 ± .1403	fz_quick02
Vaihingen	U-net	OpenGMM	.877	.522	.656	.714	.541	.6620 ± .1442	fz_quick04
Vaihingen	U-net	OpenGMM	.873	.526	.656	.715	.545	.6630 ± .1410	fz_slic02
Vaihingen	U-net	OpenGMM	.862	.528	.651	.713	.546	.6600 ± .1361	fz_slic04
Vaihingen	U-net	OpenGMM	.876	.526	.657	.709	.546	.6628 ± .1414	fz01
Vaihingen	U-net	OpenGMM	.880	.520	.659	.710	.547	.6632 ± .1442	fz02
Vaihingen	U-net	OpenGMM	.873	.501	.641	.698	.563	.6552 ± .1430	fz04
Vaihingen	U-net	OpenGMM	.879	.524	.656	.710	.542	.6622 ± .1439	fz05
Vaihingen	U-net	OpenGMM	.881	.522	.659	.711	.543	.6632 ± .1450	fz06
Vaihingen	U-net	OpenGMM	.871	.526	.643	.704	.518	.6524 ± .1454	quick02
Vaihingen	U-net	OpenGMM	.855	.526	.647	.712	.539	.6558 ± .1355	slic06

Table B.8: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with U-net as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	**1**	**2**	**3**	**4**		
Vaihingen	U-net	OpenPCS	.815	.493	.672	.737	.455	.6344 ± .1555	-
Vaihingen	U-net	OpenPCS	.846	.484	.685	.747	.440	.6404 ± .1734	fz_quick02
Vaihingen	U-net	OpenPCS	.855	.482	.688	.749	.441	.6430 ± .1767	fz_quick04
Vaihingen	U-net	OpenPCS	.851	.486	.689	.748	.442	.6432 ± .1743	fz_slic02
Vaihingen	U-net	OpenPCS	.839	.489	.683	.746	.448	.6410 ± .1676	fz_slic04
Vaihingen	U-net	OpenPCS	.855	.483	.691	.747	.447	.6446 ± .1747	fz01
Vaihingen	U-net	OpenPCS	.860	.477	.693	.748	.450	.6456 ± .1770	fz02
Vaihingen	U-net	OpenPCS	.857	.459	.679	.733	.458	.6372 ± .1754	fz04
Vaihingen	U-net	OpenPCS	.858	.480	.689	.748	.443	.6436 ± .1774	fz05
Vaihingen	U-net	OpenPCS	.862	.475	.692	.745	.446	.6440 ± .1787	fz06
Vaihingen	U-net	OpenPCS	.850	.484	.673	.745	.417	.6338 ± .1803	quick02
Vaihingen	U-net	OpenPCS	.832	.488	.676	.745	.441	.6364 ± .1672	slic06

Table B.9: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with U-net as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg. AUROC**	**SPS config.**
			0	**1**	**2**	**3**	**4**		
Vaihingen	U-net	OpenPCS++	.656	.476	.533	.600	.609	.5748 ± .0705	-
Vaihingen	U-net	OpenPCS++	.700	.466	.527	.596	.618	.5814 ± .0893	fz_quick02
Vaihingen	U-net	OpenPCS++	.708	.464	.523	.598	.623	.5832 ± .0938	fz_quick04
Vaihingen	U-net	OpenPCS++	.704	.468	.526	.604	.622	.5848 ± .0909	fz_slic02
Vaihingen	U-net	OpenPCS++	.692	.471	.528	.600	.621	.5824 ± .0854	fz_slic04
Vaihingen	U-net	OpenPCS++	.712	.468	.527	.590	.622	.5838 ± .0930	fz01
Vaihingen	U-net	OpenPCS++	.716	.465	.525	.591	.623	.5840 ± .0957	fz02
Vaihingen	U-net	OpenPCS++	.733	.454	.523	.569	.634	.5826 ± .1067	fz04
Vaihingen	U-net	OpenPCS++	.719	.465	.522	.591	.622	.5838 ± .0971	fz05
Vaihingen	U-net	OpenPCS++	.723	.463	.520	.585	.623	.5828 ± .0995	fz06
Vaihingen	U-net	OpenPCS++	.712	.460	.522	.610	.608	.5824 ± .0960	quick02
Vaihingen	U-net	OpenPCS++	.687	.470	.529	.596	.614	.5792 ± .0830	slic06

Table B.10: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with U-net as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Vaihingen	U-net+Att	CoReSeg	.886	.770	.688	.738	.627	.7418 ± .0971	-
Vaihingen	U-net+Att	CoReSeg	.911	.802	.724	.785	.652	.7748 ± .0962	fz_quick02
Vaihingen	U-net+Att	CoReSeg	.910	.809	.724	.791	.650	.7768 ± .0973	fz_quick04
Vaihingen	U-net+Att	CoReSeg	.910	.805	.723	.784	.651	.7746 ± .0965	fz_slic02
Vaihingen	U-net+Att	CoReSeg	.909	.796	.721	.774	.653	.7706 ± .0950	fz_slic04
Vaihingen	U-net+Att	CoReSeg	.910	.808	.725	.779	.646	.7736 ± .0981	fz01
Vaihingen	U-net+Att	CoReSeg	.906	.812	.726	.785	.647	.7752 ± .0967	fz02
Vaihingen	U-net+Att	CoReSeg	.867	.813	.699	.769	.643	.7582 ± .0891	fz04
Vaihingen	U-net+Att	CoReSeg	.911	.811	.727	.787	.641	.7754 ± .1002	fz05
Vaihingen	U-net+Att	CoReSeg	.906	.815	.726	.789	.641	.7754 ± .0991	fz06
Vaihingen	U-net+Att	CoReSeg	.910	.807	.718	.791	.597	.7646 ± .1161	quick02
Vaihingen	U-net+Att	CoReSeg	.906	.793	.718	.771	.647	.7670 ± .0959	slic06

Table B.11: The table shows the base open-set prediction quantitative results obtained by combining using CoReSeg with U-net+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg.	SPS
			0	1	2	3	4	AUROC	config.
Vaihingen	U-net+Att	CoReSeg+Att	.842	.900	.686	.718	.723	.7738 ± .0922	-
Vaihingen	U-net+Att	CoReSeg+Att	.867	.933	.729	.746	.789	.8128 ± .0858	fz_quick02
Vaihingen	U-net+Att	CoReSeg+Att	.866	.936	.732	.746	.793	.8146 ± .0857	fz_quick04
Vaihingen	U-net+Att	CoReSeg+Att	.868	.934	.728	.747	.792	.8138 ± .0861	fz_slic02
Vaihingen	U-net+Att	CoReSeg+Att	.866	.929	.723	.744	.784	.8092 ± .0865	fz_slic04
Vaihingen	U-net+Att	CoReSeg+Att	.865	.934	.727	.744	.780	.8100 ± .0874	fz01
Vaihingen	U-net+Att	CoReSeg+Att	.860	.935	.731	.743	.784	.8106 ± .0859	fz02
Vaihingen	U-net+Att	CoReSeg+Att	.830	.927	.725	.703	.790	.7950 ± .0895	fz04
Vaihingen	U-net+Att	CoReSeg+Att	.867	.936	.731	.748	.783	.8130 ± .0864	fz05
Vaihingen	U-net+Att	CoReSeg+Att	.861	.936	.734	.745	.786	.8124 ± .0852	fz06
Vaihingen	U-net+Att	CoReSeg+Att	.868	.936	.730	.752	.765	.8102 ± .0881	quick02
Vaihingen	U-net+Att	CoReSeg+Att	.865	.927	.720	.746	.780	.8076 ± .0863	slic06

Table B.12: The table shows the base open-set prediction quantitative results obtained by combining using CoReSeg+Att with U-net+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Vaihingen	U-net+Att	OpenGMM	.842	.738	.622	.561	.685	.6896 ± .1080	-
Vaihingen	U-net+Att	OpenGMM	.876	.752	.637	.569	.702	.7072 ± .1168	fz_quick02
Vaihingen	U-net+Att	OpenGMM	.885	.755	.639	.576	.709	.7128 ± .1179	fz_quick04
Vaihingen	U-net+Att	OpenGMM	.882	.753	.637	.571	.704	.7094 ± .1184	fz_slic02
Vaihingen	U-net+Att	OpenGMM	.869	.748	.635	.565	.700	.7034 ± .1154	fz_slic04
Vaihingen	U-net+Att	OpenGMM	.885	.755	.641	.558	.697	.7072 ± .1231	fz01
Vaihingen	U-net+Att	OpenGMM	.887	.756	.644	.567	.700	.7108 ± .1208	fz02
Vaihingen	U-net+Att	OpenGMM	.879	.741	.627	.593	.725	.7130 ± .1121	fz04
Vaihingen	U-net+Att	OpenGMM	.889	.758	.640	.557	.699	.7086 ± .1253	fz05
Vaihingen	U-net+Att	OpenGMM	.890	.759	.642	.568	.701	.7120 ± .1221	fz06
Vaihingen	U-net+Att	OpenGMM	.881	.760	.624	.554	.691	.7020 ± .1260	quick02
Vaihingen	U-net+Att	OpenGMM	.860	.744	.628	.567	.693	.6984 ± .1123	slic06

Table B.13: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with U-net+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Vaihingen	U-net+Att	OpenPCS	.812	.652	.621	.595	.563	.6486 ± .0970	-
Vaihingen	U-net+Att	OpenPCS	.850	.668	.636	.608	.559	.6642 ± .1113	fz_quick02
Vaihingen	U-net+Att	OpenPCS	.858	.671	.641	.615	.561	.6692 ± .1130	fz_quick04
Vaihingen	U-net+Att	OpenPCS	.854	.669	.640	.608	.555	.6652 ± .1137	fz_slic02
Vaihingen	U-net+Att	OpenPCS	.843	.665	.635	.602	.561	.6612 ± .1087	fz_slic04
Vaihingen	U-net+Att	OpenPCS	.858	.670	.644	.601	.563	.6672 ± .1142	fz01
Vaihingen	U-net+Att	OpenPCS	.861	.674	.648	.610	.567	.6720 ± .1131	fz02
Vaihingen	U-net+Att	OpenPCS	.858	.670	.630	.632	.586	.6752 ± .1064	fz04
Vaihingen	U-net+Att	OpenPCS	.861	.673	.644	.601	.561	.6680 ± .1160	fz05
Vaihingen	U-net+Att	OpenPCS	.864	.676	.647	.609	.561	.6714 ± .1160	fz06
Vaihingen	U-net+Att	OpenPCS	.856	.676	.622	.596	.537	.6574 ± .1218	quick02
Vaihingen	U-net+Att	OpenPCS	.835	.661	.629	.605	.552	.6564 ± .1075	slic06

Table B.14: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with U-net+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Vaihingen	U-net+Att	OpenPCS++	.590	.571	.507	.485	.513	.5332 ± .0449	-
Vaihingen	U-net+Att	OpenPCS++	.586	.573	.496	.482	.522	.5318 ± .0461	fz_quick02
Vaihingen	U-net+Att	OpenPCS++	.585	.573	.493	.480	.525	.5312 ± .0468	fz_quick04
Vaihingen	U-net+Att	OpenPCS++	.588	.574	.495	.481	.521	.5318 ± .0474	fz_slic02
Vaihingen	U-net+Att	OpenPCS++	.588	.573	.498	.481	.521	.5322 ± .0466	fz_slic04
Vaihingen	U-net+Att	OpenPCS++	.585	.575	.496	.478	.519	.5306 ± .0475	fz01
Vaihingen	U-net+Att	OpenPCS++	.581	.573	.494	.478	.518	.5288 ± .0463	fz02
Vaihingen	U-net+Att	OpenPCS++	.573	.558	.491	.476	.515	.5226 ± .0419	fz04
Vaihingen	U-net+Att	OpenPCS++	.590	.574	.491	.476	.523	.5308 ± .0500	fz05
Vaihingen	U-net+Att	OpenPCS++	.586	.573	.490	.474	.520	.5286 ± .0495	fz06
Vaihingen	U-net+Att	OpenPCS++	.582	.570	.489	.483	.526	.5300 ± .0453	quick02
Vaihingen	U-net+Att	OpenPCS++	.587	.571	.499	.484	.520	.5322 ± .0450	slic06

Table B.15: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with U-net+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Vaihingen	WRN-50	OpenGMM	.834	.864	.498	.517	.681	.6788 ± .1712	-
Vaihingen	WRN-50	OpenGMM	.868	.880	.514	.523	.715	.7000 ± .1780	fz_quick02
Vaihingen	WRN-50	OpenGMM	.872	.880	.519	.524	.716	.7022 ± .1774	fz_quick04
Vaihingen	WRN-50	OpenGMM	.868	.879	.513	.520	.710	.6980 ± .1787	fz_slic02
Vaihingen	WRN-50	OpenGMM	.861	.876	.510	.520	.711	.6956 ± .1771	fz_slic04
Vaihingen	WRN-50	OpenGMM	.866	.877	.519	.511	.708	.6962 ± .1784	fz01
Vaihingen	WRN-50	OpenGMM	.867	.877	.522	.514	.717	.6994 ± .1773	fz02
Vaihingen	WRN-50	OpenGMM	.852	.859	.519	.503	.742	.6950 ± .1744	fz04
Vaihingen	WRN-50	OpenGMM	.869	.879	.518	.513	.706	.6970 ± .1794	fz05
Vaihingen	WRN-50	OpenGMM	.868	.878	.522	.517	.713	.6996 ± .1770	fz06
Vaihingen	WRN-50	OpenGMM	.872	.884	.507	.518	.684	.6930 ± .1829	quick02
Vaihingen	WRN-50	OpenGMM	.858	.875	.507	.520	.705	.6930 ± .1768	slic06

Table B.16: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with WRN-50 as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	**1**	**2**	**3**	**4**		
Vaihingen	WRN-50	OpenPCS	.808	.846	.490	.531	.678	.6706 ± .1595	-
Vaihingen	WRN-50	OpenPCS	.836	.864	.506	.548	.699	.6906 ± .1626	fz_quick02
Vaihingen	WRN-50	OpenPCS	.839	.865	.511	.551	.700	.6932 ± .1614	fz_quick04
Vaihingen	WRN-50	OpenPCS	.836	.864	.505	.546	.698	.6898 ± .1633	fz_slic02
Vaihingen	WRN-50	OpenPCS	.831	.860	.503	.543	.697	.6868 ± .1623	fz_slic04
Vaihingen	WRN-50	OpenPCS	.832	.861	.513	.538	.698	.6884 ± .1612	fz01
Vaihingen	WRN-50	OpenPCS	.833	.863	.517	.544	.705	.6924 ± .1595	fz02
Vaihingen	WRN-50	OpenPCS	.819	.849	.517	.531	.731	.6894 ± .1572	fz04
Vaihingen	WRN-50	OpenPCS	.836	.864	.511	.540	.692	.6886 ± .1629	fz05
Vaihingen	WRN-50	OpenPCS	.834	.863	.517	.547	.699	.6920 ± .1590	fz06
Vaihingen	WRN-50	OpenPCS	.843	.870	.498	.549	.661	.6842 ± .1682	quick02
Vaihingen	WRN-50	OpenPCS	.829	.859	.499	.542	.691	.6840 ± .1629	slic06

Table B.17: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with WRN-50 as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg. AUROC**	**SPS config.**
			0	**1**	**2**	**3**	**4**		
Vaihingen	WRN-50	OpenPCS++	.419	.543	.527	.625	.653	.5534 ± .0921	-
Vaihingen	WRN-50	OpenPCS++	.406	.546	.523	.650	.657	.5564 ± .1034	fz_quick02
Vaihingen	WRN-50	OpenPCS++	.401	.541	.523	.655	.657	.5554 ± .1065	fz_quick04
Vaihingen	WRN-50	OpenPCS++	.405	.544	.525	.640	.659	.5546 ± .1019	fz_slic02
Vaihingen	WRN-50	OpenPCS++	.409	.541	.526	.635	.665	.5552 ± .1011	fz_slic04
Vaihingen	WRN-50	OpenPCS++	.405	.542	.527	.636	.660	.5540 ± .1013	fz01
Vaihingen	WRN-50	OpenPCS++	.398	.539	.526	.631	.652	.5492 ± .1009	fz02
Vaihingen	WRN-50	OpenPCS++	.392	.532	.531	.593	.650	.5396 ± .0961	fz04
Vaihingen	WRN-50	OpenPCS++	.402	.543	.525	.639	.656	.5530 ± .1021	fz05
Vaihingen	WRN-50	OpenPCS++	.399	.541	.524	.640	.648	.5504 ± .1016	fz06
Vaihingen	WRN-50	OpenPCS++	.411	.551	.516	.653	.639	.5540 ± .0987	quick02
Vaihingen	WRN-50	OpenPCS++	.410	.539	.524	.638	.662	.5546 ± .1007	slic06

Table B.18: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with WRN-50 as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Vaihingen	WRN-50+Att	OpenGMM	.845	.794	.491	.511	.789	.6860 ± .1704	-
Vaihingen	WRN-50+Att	OpenGMM	.868	.811	.526	.522	.822	.7098 ± .1710	fz_quick02
Vaihingen	WRN-50+Att	OpenGMM	.870	.812	.532	.526	.826	.7132 ± .1695	fz_quick04
Vaihingen	WRN-50+Att	OpenGMM	.868	.810	.525	.519	.822	.7088 ± .1719	fz_slic02
Vaihingen	WRN-50+Att	OpenGMM	.863	.808	.515	.515	.817	.7036 ± .1734	fz_slic04
Vaihingen	WRN-50+Att	OpenGMM	.865	.810	.533	.510	.813	.7062 ± .1702	fz01
Vaihingen	WRN-50+Att	OpenGMM	.864	.808	.538	.519	.817	.7092 ± .1665	fz02
Vaihingen	WRN-50+Att	OpenGMM	.846	.794	.527	.525	.837	.7058 ± .1653	fz04
Vaihingen	WRN-50+Att	OpenGMM	.867	.812	.535	.512	.818	.7088 ± .1707	fz05
Vaihingen	WRN-50+Att	OpenGMM	.865	.810	.539	.519	.818	.7102 ± .1669	fz06
Vaihingen	WRN-50+Att	OpenGMM	.871	.819	.522	.515	.819	.7092 ± .1754	quick02
Vaihingen	WRN-50+Att	OpenGMM	.861	.807	.511	.517	.816	.7024 ± .1732	slic06

Table B.19: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with WRN-50+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg.	SPS
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Vaihingen	WRN-50+Att	OpenPCS	.796	.784	.505	.537	.761	.6766 ± .1430	-
Vaihingen	WRN-50+Att	OpenPCS	.824	.805	.534	.562	.801	.7052 ± .1441	fz_quick02
Vaihingen	WRN-50+Att	OpenPCS	.825	.805	.541	.570	.807	.7096 ± .1413	fz_quick04
Vaihingen	WRN-50+Att	OpenPCS	.825	.804	.534	.559	.804	.7052 ± .1454	fz_slic02
Vaihingen	WRN-50+Att	OpenPCS	.819	.801	.524	.551	.796	.6982 ± .1473	fz_slic04
Vaihingen	WRN-50+Att	OpenPCS	.822	.803	.542	.551	.794	.7024 ± .1427	fz01
Vaihingen	WRN-50+Att	OpenPCS	.821	.800	.549	.569	.798	.7074 ± .1360	fz02
Vaihingen	WRN-50+Att	OpenPCS	.806	.788	.541	.594	.824	.7106 ± .1326	fz04
Vaihingen	WRN-50+Att	OpenPCS	.825	.807	.544	.557	.801	.7068 ± .1430	fz05
Vaihingen	WRN-50+Att	OpenPCS	.823	.802	.549	.569	.802	.7090 ± .1374	fz06
Vaihingen	WRN-50+Att	OpenPCS	.827	.814	.529	.561	.799	.7060 ± .1477	quick02
Vaihingen	WRN-50+Att	OpenPCS	.817	.801	.520	.551	.797	.6972 ± .1482	slic06

Table B.20: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with WRN-50+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg.	SPS
			0	1	2	3	4	AUROC	config.
Vaihingen	WRN-50+Att	OpenPCS++	.557	.562	.612	.644	.457	.5664 ± .0710	-
Vaihingen	WRN-50+Att	OpenPCS++	.556	.558	.611	.660	.467	.5704 ± .0720	fz_quick02
Vaihingen	WRN-50+Att	OpenPCS++	.556	.553	.610	.660	.467	.5692 ± .0721	fz_quick04
Vaihingen	WRN-50+Att	OpenPCS++	.556	.556	.609	.649	.468	.5676 ± .0681	fz_slic02
Vaihingen	WRN-50+Att	OpenPCS++	.558	.562	.610	.649	.468	.5694 ± .0679	fz_slic04
Vaihingen	WRN-50+Att	OpenPCS++	.559	.558	.614	.649	.456	.5672 ± .0731	fz01
Vaihingen	WRN-50+Att	OpenPCS++	.552	.551	.616	.660	.455	.5668 ± .0776	fz02
Vaihingen	WRN-50+Att	OpenPCS++	.537	.530	.605	.650	.470	.5584 ± .0701	fz04
Vaihingen	WRN-50+Att	OpenPCS++	.559	.557	.613	.655	.466	.5700 ± .0710	fz05
Vaihingen	WRN-50+Att	OpenPCS++	.549	.552	.610	.664	.467	.5684 ± .0738	fz06
Vaihingen	WRN-50+Att	OpenPCS++	.549	.557	.597	.656	.500	.5718 ± .0583	quick02
Vaihingen	WRN-50+Att	OpenPCS++	.551	.563	.608	.651	.476	.5698 ± .0657	slic06

Table B.21: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with WRN-50+Att as the backbone for the Vaihingen dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Potsdam	DN-121	OpenGMM	.717	.760	.417	.518	.928	.6680 ± .2025	-
Potsdam	DN-121	OpenGMM	.740	.777	.413	.518	.932	.6760 ± .2085	fz_quick02
Potsdam	DN-121	OpenGMM	.745	.774	.410	.517	.926	.6744 ± .2079	fz_quick04
Potsdam	DN-121	OpenGMM	.740	.773	.412	.517	.926	.6736 ± .2067	fz_slic02
Potsdam	DN-121	OpenGMM	.734	.773	.413	.518	.936	.6748 ± .2089	fz_slic04
Potsdam	DN-121	OpenGMM	.747	.775	.413	.518	.931	.6768 ± .2086	fz01
Potsdam	DN-121	OpenGMM	.745	.781	.396	.512	.925	.6718 ± .2139	fz02
Potsdam	DN-121	OpenGMM	.629	.781	.357	.490	.872	.6258 ± .2094	fz04
Potsdam	DN-121	OpenGMM	.750	.774	.412	.514	.923	.6746 ± .2074	fz05
Potsdam	DN-121	OpenGMM	.745	.781	.398	.510	.919	.6706 ± .2119	fz06
Potsdam	DN-121	OpenGMM	.739	.781	.409	.515	.910	.6708 ± .2042	quick02
Potsdam	DN-121	OpenGMM	.729	.768	.414	.518	.934	.6726 ± .2070	slic06

Table B.22: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with DN-121 as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Potsdam	DN-121	OpenPCS	.751	.786	.388	.335	.873	.6266 ± .2468	-
Potsdam	DN-121	OpenPCS	.714	.795	.386	.333	.873	.6202 ± .2453	fz_quick02
Potsdam	DN-121	OpenPCS	.718	.798	.382	.335	.874	.6214 ± .2468	fz_quick04
Potsdam	DN-121	OpenPCS	.593	.796	.387	.334	.875	.5970 ± .2399	fz_slic02
Potsdam	DN-121	OpenPCS	.563	.793	.388	.333	.873	.5900 ± .2392	fz_slic04
Potsdam	DN-121	OpenPCS	.578	.795	.386	.339	.875	.5946 ± .2387	fz01
Potsdam	DN-121	OpenPCS	.540	.796	.379	.354	.878	.5894 ± .2388	fz02
Potsdam	DN-121	OpenPCS	.604	.775	.359	.449	.877	.6128 ± .2165	fz04
Potsdam	DN-121	OpenPCS	.657	.797	.386	.339	.875	.6108 ± .2403	fz05
Potsdam	DN-121	OpenPCS	.677	.796	.378	.355	.877	.6166 ± .2393	fz06
Potsdam	DN-121	OpenPCS	.770	.796	.383	.331	.870	.6300 ± .2526	quick02
Potsdam	DN-121	OpenPCS	.682	.792	.388	.332	.873	.6134 ± .2419	slic06

Table B.23: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with DN-121 as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg.	SPS
			0	1	2	3	4	AUROC	config.
Potsdam	DN-121	OpenPCS++	.666	.692	.537	.645	.903	.6886 ± .1336	-
Potsdam	DN-121	OpenPCS++	.705	.725	.494	.663	.922	.7018 ± .1531	fz_quick02
Potsdam	DN-121	OpenPCS++	.727	.735	.490	.646	.914	.7024 ± .1539	fz_quick04
Potsdam	DN-121	OpenPCS++	.711	.723	.524	.647	.921	.7052 ± .1442	fz_slic02
Potsdam	DN-121	OpenPCS++	.693	.719	.529	.658	.922	.7042 ± .1420	fz_slic04
Potsdam	DN-121	OpenPCS++	.743	.717	.517	.645	.920	.7084 ± .1472	fz01
Potsdam	DN-121	OpenPCS++	.744	.733	.537	.633	.905	.7104 ± .1374	fz02
Potsdam	DN-121	OpenPCS++	.696	.661	.505	.601	.866	.6658 ± .1333	fz04
Potsdam	DN-121	OpenPCS++	.739	.751	.495	.647	.915	.7094 ± .1539	fz05
Potsdam	DN-121	OpenPCS++	.750	.742	.525	.656	.906	.7158 ± .1397	fz06
Potsdam	DN-121	OpenPCS++	.707	.734	.501	.663	.904	.7018 ± .1447	quick02
Potsdam	DN-121	OpenPCS++	.684	.719	.523	.654	.923	.7006 ± .1447	slic06

Table B.24: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with DN-121 as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg.	SPS
			0	1	2	3	4	AUROC	config.
Potsdam	DN-121+Att	OpenGMM	.783	.773	.448	.455	.948	.6814 ± .2211	-
Potsdam	DN-121+Att	OpenGMM	.797	.792	.446	.456	.955	.6892 ± .2271	fz_quick02
Potsdam	DN-121+Att	OpenGMM	.802	.798	.446	.456	.951	.6906 ± .2273	fz_quick04
Potsdam	DN-121+Att	OpenGMM	.799	.791	.447	.457	.959	.6906 ± .2279	fz_slic02
Potsdam	DN-121+Att	OpenGMM	.793	.786	.450	.457	.959	.6890 ± .2259	fz_slic04
Potsdam	DN-121+Att	OpenGMM	.804	.796	.447	.450	.954	.6902 ± .2294	fz01
Potsdam	DN-121+Att	OpenGMM	.798	.800	.438	.448	.951	.6870 ± .2313	fz02
Potsdam	DN-121+Att	OpenGMM	.686	.786	.412	.453	.892	.6458 ± .2084	fz04
Potsdam	DN-121+Att	OpenGMM	.805	.800	.445	.449	.948	.6894 ± .2291	fz05
Potsdam	DN-121+Att	OpenGMM	.799	.804	.435	.446	.944	.6856 ± .2312	fz06
Potsdam	DN-121+Att	OpenGMM	.797	.802	.444	.451	.938	.6864 ± .2253	quick02
Potsdam	DN-121+Att	OpenGMM	.789	.784	.449	.456	.957	.6870 ± .2251	slic06

Table B.25: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with DN-121+Att as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Potsdam	DN-121+Att	OpenPCS	.771	.763	.525	.481	.947	.6974 ± .1927	-
Potsdam	DN-121+Att	OpenPCS	.788	.781	.519	.482	.955	.7050 ± .1997	fz_quick02
Potsdam	DN-121+Att	OpenPCS	.794	.785	.517	.483	.951	.7060 ± .1997	fz_quick04
Potsdam	DN-121+Att	OpenPCS	.788	.780	.518	.483	.959	.7056 ± .2008	fz_slic02
Potsdam	DN-121+Att	OpenPCS	.783	.775	.522	.481	.959	.7040 ± .1995	fz_slic04
Potsdam	DN-121+Att	OpenPCS	.783	.783	.519	.478	.954	.7034 ± .2002	fz01
Potsdam	DN-121+Att	OpenPCS	.790	.789	.513	.476	.950	.7036 ± .2022	fz02
Potsdam	DN-121+Att	OpenPCS	.683	.770	.463	.469	.894	.6558 ± .1888	fz04
Potsdam	DN-121+Att	OpenPCS	.798	.788	.511	.477	.948	.7044 ± .2026	fz05
Potsdam	DN-121+Att	OpenPCS	.792	.792	.504	.474	.945	.7014 ± .2040	fz06
Potsdam	DN-121+Att	OpenPCS	.788	.790	.514	.478	.939	.7018 ± .1980	quick02
Potsdam	DN-121+Att	OpenPCS	.779	.774	.522	.481	.957	.7026 ± .1984	slic06

Table B.26: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with DN-121+Att as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Potsdam	DN-121+Att	OpenPCS++	.520	.712	.492	.463	.907	.6188 ± .1883	-
Potsdam	DN-121+Att	OpenPCS++	.534	.738	.475	.464	.940	.6302 ± .2053	fz_quick02
Potsdam	DN-121+Att	OpenPCS++	.531	.752	.457	.463	.938	.6282 ± .2106	fz_quick04
Potsdam	DN-121+Att	OpenPCS++	.524	.735	.484	.465	.944	.6304 ± .2058	fz_slic02
Potsdam	DN-121+Att	OpenPCS++	.528	.729	.492	.463	.940	.6304 ± .2020	fz_slic04
Potsdam	DN-121+Att	OpenPCS++	.556	.758	.445	.459	.938	.6312 ± .2122	fz01
Potsdam	DN-121+Att	OpenPCS++	.550	.759	.461	.455	.934	.6318 ± .2089	fz02
Potsdam	DN-121+Att	OpenPCS++	.541	.751	.415	.499	.900	.6212 ± .1990	fz04
Potsdam	DN-121+Att	OpenPCS++	.559	.772	.422	.457	.936	.6292 ± .2191	fz05
Potsdam	DN-121+Att	OpenPCS++	.541	.760	.430	.459	.934	.6248 ± .2158	fz06
Potsdam	DN-121+Att	OpenPCS++	.535	.753	.470	.463	.927	.6296 ± .2036	quick02
Potsdam	DN-121+Att	OpenPCS++	.524	.725	.484	.460	.937	.6260 ± .2029	slic06

Table B.27: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with DN-121+Att as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Potsdam	U-net	OpenGMM	.807	.725	.349	.387	.906	.6348 ± .2522	-
Potsdam	U-net	OpenGMM	.834	.744	.343	.386	.923	.6460 ± .2651	fz_quick02
Potsdam	U-net	OpenGMM	.843	.753	.339	.387	.919	.6482 ± .2674	fz_quick04
Potsdam	U-net	OpenGMM	.836	.749	.343	.387	.929	.6488 ± .2672	fz_slic02
Potsdam	U-net	OpenGMM	.826	.738	.347	.386	.925	.6444 ± .2625	fz_slic04
Potsdam	U-net	OpenGMM	.848	.750	.337	.386	.920	.6482 ± .2691	fz01
Potsdam	U-net	OpenGMM	.848	.750	.321	.393	.919	.6462 ± .2719	fz02
Potsdam	U-net	OpenGMM	.744	.725	.317	.414	.875	.6150 ± .2375	fz04
Potsdam	U-net	OpenGMM	.850	.754	.335	.386	.913	.6476 ± .2687	fz05
Potsdam	U-net	OpenGMM	.850	.751	.319	.393	.912	.6450 ± .2713	fz06
Potsdam	U-net	OpenGMM	.836	.754	.342	.384	.898	.6428 ± .2609	quick02
Potsdam	U-net	OpenGMM	.820	.733	.348	.386	.922	.6418 ± .2600	slic06

Table B.28: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with U-net as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg. AUROC**	**SPS config.**
			0	**1**	**2**	**3**	**4**		
Potsdam	U-net	OpenPCS	.771	.707	.326	.384	.848	.6072 ± .2365	-
Potsdam	U-net	OpenPCS	.800	.730	.318	.383	.878	.6218 ± .2542	fz_quick02
Potsdam	U-net	OpenPCS	.809	.740	.312	.384	.874	.6238 ± .2575	fz_quick04
Potsdam	U-net	OpenPCS	.802	.735	.318	.384	.885	.6248 ± .2566	fz_slic02
Potsdam	U-net	OpenPCS	.792	.724	.323	.383	.881	.6206 ± .2514	fz_slic04
Potsdam	U-net	OpenPCS	.813	.737	.309	.383	.876	.6236 ± .2595	fz01
Potsdam	U-net	OpenPCS	.816	.739	.293	.392	.874	.6228 ± .2627	fz02
Potsdam	U-net	OpenPCS	.700	.726	.290	.422	.840	.5956 ± .2298	fz04
Potsdam	U-net	OpenPCS	.815	.744	.309	.383	.869	.6240 ± .2589	fz05
Potsdam	U-net	OpenPCS	.817	.745	.292	.392	.869	.6230 ± .2627	fz06
Potsdam	U-net	OpenPCS	.802	.743	.316	.382	.851	.6188 ± .2503	quick02
Potsdam	U-net	OpenPCS	.786	.719	.324	.383	.876	.6176 ± .2483	slic06

Table B.29: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with U-net as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Potsdam	U-net	OpenPCS++	.561	.714	.557	.572	.681	.6170 ± .0746	-
Potsdam	U-net	OpenPCS++	.571	.750	.549	.587	.719	.6352 ± .0923	fz_quick02
Potsdam	U-net	OpenPCS++	.588	.755	.534	.570	.720	.6334 ± .0978	fz_quick04
Potsdam	U-net	OpenPCS++	.580	.757	.559	.585	.724	.6410 ± .0921	fz_slic02
Potsdam	U-net	OpenPCS++	.572	.743	.563	.583	.716	.6354 ± .0867	fz_slic04
Potsdam	U-net	OpenPCS++	.595	.770	.497	.591	.719	.6344 ± .1094	fz01
Potsdam	U-net	OpenPCS++	.581	.774	.498	.567	.702	.6244 ± .1113	fz02
Potsdam	U-net	OpenPCS++	.552	.752	.448	.568	.671	.5982 ± .1168	fz04
Potsdam	U-net	OpenPCS++	.585	.784	.517	.574	.733	.6386 ± .1139	fz05
Potsdam	U-net	OpenPCS++	.606	.773	.495	.596	.713	.6366 ± .1085	fz06
Potsdam	U-net	OpenPCS++	.570	.757	.549	.587	.726	.6378 ± .0962	quick02
Potsdam	U-net	OpenPCS++	.567	.740	.551	.576	.713	.6294 ± .0896	slic06

Table B.30: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with U-net as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Potsdam	U-net+Att	CoReSeg+Att	.749	.876	.646	.544	.772	.7174 ± .1268	-
Potsdam	U-net+Att	CoReSeg+Att	.769	.889	.665	.553	.820	.7392 ± .1323	fz_quick02
Potsdam	U-net+Att	CoReSeg+Att	.771	.891	.668	.554	.820	.7408 ± .1322	fz_quick04
Potsdam	U-net+Att	CoReSeg+Att	.770	.889	.664	.552	.825	.7400 ± .1336	fz_slic02
Potsdam	U-net+Att	CoReSeg+Att	.768	.886	.661	.551	.818	.7368 ± .1324	fz_slic04
Potsdam	U-net+Att	CoReSeg+Att	.768	.891	.665	.552	.820	.7392 ± .1332	fz01
Potsdam	U-net+Att	CoReSeg+Att	.753	.892	.656	.545	.817	.7326 ± .1360	fz02
Potsdam	U-net+Att	CoReSeg+Att	.636	.872	.584	.499	.791	.6764 ± .1525	fz04
Potsdam	U-net+Att	CoReSeg+Att	.769	.892	.666	.550	.826	.7406 ± .1350	fz05
Potsdam	U-net+Att	CoReSeg+Att	.753	.893	.657	.545	.822	.7340 ± .1370	fz06
Potsdam	U-net+Att	CoReSeg+Att	.769	.891	.665	.549	.824	.7396 ± .1349	quick02
Potsdam	U-net+Att	CoReSeg+Att	.765	.886	.659	.548	.822	.7360 ± .1342	slic06

Table B.31: The table shows the base open-set prediction quantitative results obtained by combining using CoReSeg+Att with U-net+Att as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Potsdam	U-net+Att	OpenGMM	.773	.744	.432	.397	.904	.6500 ± .2236	-
Potsdam	U-net+Att	OpenGMM	.808	.754	.436	.395	.917	.6620 ± .2330	fz_quick02
Potsdam	U-net+Att	OpenGMM	.822	.759	.436	.395	.914	.6652 ± .2350	fz_quick04
Potsdam	U-net+Att	OpenGMM	.813	.756	.439	.396	.923	.6654 ± .2346	fz_slic02
Potsdam	U-net+Att	OpenGMM	.797	.752	.438	.396	.918	.6602 ± .2306	fz_slic04
Potsdam	U-net+Att	OpenGMM	.835	.755	.438	.394	.915	.6674 ± .2369	fz01
Potsdam	U-net+Att	OpenGMM	.841	.752	.435	.401	.916	.6690 ± .2367	fz02
Potsdam	U-net+Att	OpenGMM	.762	.724	.446	.434	.879	.6490 ± .1992	fz04
Potsdam	U-net+Att	OpenGMM	.838	.757	.438	.393	.909	.6670 ± .2363	fz05
Potsdam	U-net+Att	OpenGMM	.843	.755	.435	.401	.910	.6688 ± .2358	fz06
Potsdam	U-net+Att	OpenGMM	.810	.761	.434	.393	.895	.6586 ± .2293	quick02
Potsdam	U-net+Att	OpenGMM	.788	.750	.437	.397	.915	.6574 ± .2282	slic06

Table B.32: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with U-net+Att as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Potsdam	U-net+Att	OpenPCS	.704	.704	.460	.384	.874	.6252 ± .1999	-
Potsdam	U-net+Att	OpenPCS	.744	.714	.454	.383	.888	.6366 ± .2112	fz_quick02
Potsdam	U-net+Att	OpenPCS	.761	.720	.450	.383	.886	.6400 ± .2143	fz_quick04
Potsdam	U-net+Att	OpenPCS	.750	.718	.452	.384	.893	.6394 ± .2139	fz_slic02
Potsdam	U-net+Att	OpenPCS	.732	.711	.457	.384	.889	.6346 ± .2088	fz_slic04
Potsdam	U-net+Att	OpenPCS	.783	.714	.449	.382	.886	.6428 ± .2176	fz01
Potsdam	U-net+Att	OpenPCS	.801	.710	.437	.386	.887	.6442 ± .2222	fz02
Potsdam	U-net+Att	OpenPCS	.743	.694	.399	.405	.855	.6192 ± .2067	fz04
Potsdam	U-net+Att	OpenPCS	.787	.714	.447	.381	.879	.6416 ± .2171	fz05
Potsdam	U-net+Att	OpenPCS	.803	.717	.433	.385	.881	.6438 ± .2227	fz06
Potsdam	U-net+Att	OpenPCS	.748	.721	.452	.381	.860	.6324 ± .2054	quick02
Potsdam	U-net+Att	OpenPCS	.725	.709	.458	.384	.885	.6322 ± .2064	slic06

Table B.33: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with U-net+Att as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Potsdam	U-net+Att	OpenPCS++	.586	.692	.365	.435	.688	.5532 ± .1482	-
Potsdam	U-net+Att	OpenPCS++	.621	.720	.344	.425	.721	.5662 ± .1732	fz_quick02
Potsdam	U-net+Att	OpenPCS++	.640	.711	.345	.417	.717	.5660 ± .1735	fz_quick04
Potsdam	U-net+Att	OpenPCS++	.632	.706	.371	.426	.728	.5726 ± .1640	fz_slic02
Potsdam	U-net+Att	OpenPCS++	.613	.708	.360	.426	.719	.5652 ± .1642	fz_slic04
Potsdam	U-net+Att	OpenPCS++	.646	.720	.366	.423	.719	.5748 ± .1685	fz01
Potsdam	U-net+Att	OpenPCS++	.642	.734	.344	.437	.720	.5754 ± .1755	fz02
Potsdam	U-net+Att	OpenPCS++	.608	.735	.372	.446	.693	.5708 ± .1568	fz04
Potsdam	U-net+Att	OpenPCS++	.634	.728	.330	.431	.716	.5678 ± .1784	fz05
Potsdam	U-net+Att	OpenPCS++	.644	.727	.361	.420	.706	.5716 ± .1694	fz06
Potsdam	U-net+Att	OpenPCS++	.611	.724	.354	.424	.722	.5670 ± .1706	quick02
Potsdam	U-net+Att	OpenPCS++	.611	.713	.354	.421	.722	.5642 ± .1688	slic06

Table B.34: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with U-net+Att as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Potsdam	WRN-50	OpenGMM	.665	.763	.304	.468	.932	.6264 ± .2463	-
Potsdam	WRN-50	OpenGMM	.670	.799	.298	.468	.951	.6372 ± .2595	fz_quick02
Potsdam	WRN-50	OpenGMM	.672	.811	.292	.465	.949	.6378 ± .2634	fz_quick04
Potsdam	WRN-50	OpenGMM	.672	.800	.297	.470	.954	.6386 ± .2607	fz_slic02
Potsdam	WRN-50	OpenGMM	.670	.788	.301	.471	.951	.6362 ± .2564	fz_slic04
Potsdam	WRN-50	OpenGMM	.672	.804	.291	.465	.949	.6362 ± .2626	fz01
Potsdam	WRN-50	OpenGMM	.664	.812	.277	.466	.947	.6332 ± .2675	fz02
Potsdam	WRN-50	OpenGMM	.574	.799	.300	.461	.890	.6048 ± .2417	fz04
Potsdam	WRN-50	OpenGMM	.671	.812	.289	.462	.954	.6376 ± .2665	fz05
Potsdam	WRN-50	OpenGMM	.664	.815	.276	.464	.942	.6322 ± .2672	fz06
Potsdam	WRN-50	OpenGMM	.669	.810	.296	.461	.939	.6350 ± .2595	quick02
Potsdam	WRN-50	OpenGMM	.667	.782	.302	.471	.949	.6342 ± .2545	slic06

Table B.35: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with WRN-50 as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg.	SPS
			0	1	2	3	4	AUROC	config.
Potsdam	WRN-50	OpenPCS	.697	.738	.301	.445	.934	.6230 ± .2504	-
Potsdam	WRN-50	OpenPCS	.719	.777	.291	.454	.953	.6388 ± .2643	fz_quick02
Potsdam	WRN-50	OpenPCS	.724	.789	.284	.454	.951	.6404 ± .2680	fz_quick04
Potsdam	WRN-50	OpenPCS	.719	.777	.291	.455	.957	.6398 ± .2653	fz_slic02
Potsdam	WRN-50	OpenPCS	.715	.766	.295	.454	.954	.6368 ± .2616	fz_slic04
Potsdam	WRN-50	OpenPCS	.724	.783	.282	.454	.952	.6390 ± .2681	fz01
Potsdam	WRN-50	OpenPCS	.718	.790	.270	.462	.949	.6378 ± .2705	fz02
Potsdam	WRN-50	OpenPCS	.609	.787	.291	.469	.893	.6098 ± .2414	fz04
Potsdam	WRN-50	OpenPCS	.725	.790	.281	.453	.947	.6392 ± .2682	fz05
Potsdam	WRN-50	OpenPCS	.716	.795	.268	.462	.944	.6370 ± .2703	fz06
Potsdam	WRN-50	OpenPCS	.719	.789	.288	.450	.940	.6372 ± .2638	quick02
Potsdam	WRN-50	OpenPCS	.710	.759	.296	.452	.952	.6338 ± .2598	slic06

Table B.36: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with WRN-50 as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg.	SPS
			0	1	2	3	4	AUROC	config.
Potsdam	WRN-50	OpenPCS++	.538	.628	.458	.465	.851	.5880 ± .1622	-
Potsdam	WRN-50	OpenPCS++	.558	.676	.450	.464	.915	.6126 ± .1917	fz_quick02
Potsdam	WRN-50	OpenPCS++	.559	.700	.457	.464	.907	.6174 ± .1893	fz_quick04
Potsdam	WRN-50	OpenPCS++	.543	.672	.481	.468	.914	.6156 ± .1853	fz_slic02
Potsdam	WRN-50	OpenPCS++	.547	.664	.463	.463	.912	.6098 ± .1880	fz_slic04
Potsdam	WRN-50	OpenPCS++	.580	.694	.473	.465	.908	.6240 ± .1841	fz01
Potsdam	WRN-50	OpenPCS++	.559	.713	.472	.475	.892	.6222 ± .1798	fz02
Potsdam	WRN-50	OpenPCS++	.545	.709	.459	.511	.869	.6186 ± .1683	fz04
Potsdam	WRN-50	OpenPCS++	.573	.702	.445	.468	.912	.6200 ± .1923	fz05
Potsdam	WRN-50	OpenPCS++	.563	.706	.469	.479	.902	.6238 ± .1822	fz06
Potsdam	WRN-50	OpenPCS++	.560	.693	.447	.463	.909	.6144 ± .1916	quick02
Potsdam	WRN-50	OpenPCS++	.546	.659	.463	.463	.909	.6080 ± .1865	slic06

Table B.37: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with WRN-50 as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Potsdam	WRN-50+Att	OpenGMM	.673	.748	.356	.382	.907	.6132 ± .2386	-
Potsdam	WRN-50+Att	OpenGMM	.688	.777	.353	.378	.923	.6238 ± .2504	fz_quick02
Potsdam	WRN-50+Att	OpenGMM	.691	.784	.347	.374	.918	.6228 ± .2529	fz_quick04
Potsdam	WRN-50+Att	OpenGMM	.689	.777	.353	.378	.927	.6248 ± .2517	fz_slic02
Potsdam	WRN-50+Att	OpenGMM	.684	.769	.357	.382	.926	.6236 ± .2478	fz_slic04
Potsdam	WRN-50+Att	OpenGMM	.692	.778	.347	.373	.921	.6222 ± .2531	fz01
Potsdam	WRN-50+Att	OpenGMM	.685	.780	.339	.373	.917	.6188 ± .2540	fz02
Potsdam	WRN-50+Att	OpenGMM	.579	.758	.335	.392	.864	.5856 ± .2278	fz04
Potsdam	WRN-50+Att	OpenGMM	.693	.785	.346	.370	.915	.6218 ± .2535	fz05
Potsdam	WRN-50+Att	OpenGMM	.686	.784	.339	.371	.912	.6184 ± .2537	fz06
Potsdam	WRN-50+Att	OpenGMM	.688	.790	.350	.371	.905	.6208 ± .2498	quick02
Potsdam	WRN-50+Att	OpenGMM	.680	.766	.358	.383	.924	.6222 ± .2460	slic06

Table B.38: The table shows the base open-set prediction quantitative results obtained by combining using OpenGMM with WRN-50+Att as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	**Backbone**	**Method**	**UUCs**					**Avg.**	**SPS**
			0	**1**	**2**	**3**	**4**	**AUROC**	**config.**
Potsdam	WRN-50+Att	OpenPCS	.660	.739	.386	.381	.896	.6124 ± .2256	-
Potsdam	WRN-50+Att	OpenPCS	.667	.775	.363	.380	.917	.6204 ± .2440	fz_quick02
Potsdam	WRN-50+Att	OpenPCS	.668	.784	.356	.379	.913	.6200 ± .2464	fz_quick04
Potsdam	WRN-50+Att	OpenPCS	.669	.773	.364	.381	.922	.6218 ± .2448	fz_slic02
Potsdam	WRN-50+Att	OpenPCS	.666	.766	.373	.383	.919	.6214 ± .2398	fz_slic04
Potsdam	WRN-50+Att	OpenPCS	.670	.778	.356	.378	.915	.6194 ± .2463	fz01
Potsdam	WRN-50+Att	OpenPCS	.661	.782	.344	.380	.911	.6156 ± .2481	fz02
Potsdam	WRN-50+Att	OpenPCS	.557	.768	.334	.405	.858	.5844 ± .2259	fz04
Potsdam	WRN-50+Att	OpenPCS	.669	.784	.353	.376	.910	.6184 ± .2471	fz05
Potsdam	WRN-50+Att	OpenPCS	.661	.784	.342	.378	.906	.6142 ± .2480	fz06
Potsdam	WRN-50+Att	OpenPCS	.667	.789	.356	.375	.898	.6170 ± .2438	quick02
Potsdam	WRN-50+Att	OpenPCS	.663	.762	.376	.384	.917	.6204 ± .2374	slic06

Table B.39: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS with WRN-50+Att as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

Dataset	Backbone	Method	UUCs					Avg. AUROC	SPS config.
			0	1	2	3	4		
Potsdam	WRN-50+Att	OpenPCS++	.367	.565	.492	.501	.821	.5492 ± .1680	-
Potsdam	WRN-50+Att	OpenPCS++	.298	.610	.437	.496	.879	.5440 ± .2184	fz_quick02
Potsdam	WRN-50+Att	OpenPCS++	.300	.634	.430	.475	.871	.5420 ± .2193	fz_quick04
Potsdam	WRN-50+Att	OpenPCS++	.276	.601	.448	.488	.878	.5382 ± .2229	fz_slic02
Potsdam	WRN-50+Att	OpenPCS++	.302	.596	.461	.495	.875	.5458 ± .2122	fz_slic04
Potsdam	WRN-50+Att	OpenPCS++	.284	.628	.444	.496	.867	.5438 ± .2187	fz01
Potsdam	WRN-50+Att	OpenPCS++	.314	.628	.414	.478	.857	.5382 ± .2115	fz02
Potsdam	WRN-50+Att	OpenPCS++	.429	.596	.468	.514	.809	.5632 ± .1508	fz04
Potsdam	WRN-50+Att	OpenPCS++	.304	.648	.403	.483	.870	.5416 ± .2226	fz05
Potsdam	WRN-50+Att	OpenPCS++	.339	.625	.418	.487	.866	.5470 ± .2070	fz06
Potsdam	WRN-50+Att	OpenPCS++	.301	.625	.436	.491	.873	.5452 ± .2170	quick02
Potsdam	WRN-50+Att	OpenPCS++	.312	.595	.467	.498	.876	.5496 ± .2089	slic06

Table B.40: The table shows the base open-set prediction quantitative results obtained by combining using OpenPCS++ with WRN-50+Att as the backbone for the Potsdam dataset. The table shows the performance of the base open-set prediction compared to all tested post-processing configurations. The UUCs number stands for 0 - impervious surfaces; 1 - building; 2 - low vegetation; 3 - high vegetation; and 4 - car.

C
Publication Status

Section 3.1 (“A Systematic Mapping of Open-set Segmentation in Visual Learning”) has already been presented at the SIBGRAPI. As a result of the previous publication, Computer & Graphics, published by Elsevier, invited the authors to write an extended version of the article.

– Reference: Nunes, I.; Oliveira, H.; Pereira, M. B.; Santos, J. A.; Poggi, M. **“Deep Open-Set Segmentation in Visual Learning”**. In: Conference on Graphics, Patterns and Images, 35. (SIBGRAPI), 2022, Natal, RN. Proceedings... 2022. On-line. IBI: 8JMKD3MGPEW34M/47MJCTH. Available from: `http://urlib.net/ibi/8JMKD3MGPEW34M/47MJCTH`.

Section 4.2 (“Conditional Reconstruction for Open-set Semantic Segmentation”) has already been presented at the ICIP.

– Reference: I. Nunes, M. B. Pereira, H. Oliveira, J. A. dos Santos and M. Poggi, **“Conditional Reconstruction for Open-Set Semantic Segmentation”** 2022 IEEE International Conference on Image Processing (ICIP), 2022, pp. 946-950, doi: 10.1109/ICIP46576.2022.9897407.

The method described in section 4.1 (“Open Gaussian Mixture of Models’) and the post-processing described in section 4.3 (“Improving Semantic Consistency with Superpixels”) have been submitted to a journal and are currently under review.