Università degli Studi di Salerno

.DIEM

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata

Corso di Laurea Magistrale in Ingegneria Informatica

Machine Learning
a.a. 2024/2025

Project Work
Gruppo n. 09 – AH

Cognome e Nome	Matricola	e-mail
Cirillo Francesco Pio	0622702466	f.cirillo36@studenti.unisa.it
Fasolino Alessandra	0622702465	a.fasolino35@studenti.unisa.it

Introduction

The project objective is to develop an image segmentation system to elaborate images acquired by vehicles in rural settings. The objective is to be reached through the utilization of machine learning techniques.

The system will have to classify every pixel of the image into 8 predefined classes (sky, rough trail, smooth trail, traversable grass, high vegetation, non-traversable low vegetation, puddle, and obstacle), while respecting computational constraints regarding the amount of VRAM used during training and inference. To train the model it has been provided a dataset which cannot be altered, although augmentation is allowed.

1. Overview of the technologies used

1.1 Networks

The training and research activity started with the decision to perform fine-tuning on an already existing network pretrained on a dataset as similar to ours as possible.

The decision to start from a pretrained network stems from two main reasons:

- the dataset available for training is relatively small, making it difficult for the network to learn both low level and high level features;
- using a pretrained network grants the possibility to choose an architecture which has already proven itself. If the focus had been put towards developing a new network from scratch it would have been necessary to experiment with a very high number of different solutions just to reach the level of quality of a pre-existing network, which would have been difficult considering the time constraint.

In particular we have focused on the study of 3 different networks: DeepLabV3, DeepLabV3+ and BiSeNetV2. For DeepLabV3, since it has shown to be the most promising network, the study also focused on the usage of different backbones in combination with it, specifically: MobileNetV3 Large, ResNet50 and ResNet101.

By testing all these different networks it has been possible to choose the one that performed better after fine-tuning it on the dataset of interest. After the selection of the best network more work has been put towards the optimization of the results achieved by using different techniques which will be explored later on.

1.1.1 DeepLabV3

DeepLabV3 is an architecture for image segmentation. It belongs to the family of DeepLab models developed by Google Research and published in 2017. This model has been considered of interest because of the auxiliary instruments it uses, in particular Atrous Convolution and ASPP (Atrous Convolution Spatial Pyramid Pooling) which use kernels to improve performance.

In DeepLabV3, Atrous Convolution is a fundamental technique that allows to control the sampling density of convolutional filters by expanding the receptive field without having to increase the number of parameters and without reducing the resolution of the activation map, which is what happens with pooling and with convolutions with stride greater than 1.

The main advantages of Atrous Convolution are that it allows the network to:

1. Capture context on a larger scale, which is useful to better understand the global structure of a scene;
2. Maintain fine spatial details, which is fundamental for image segmentation, especially to better define the contours of objects.

DeepLabV3 uses ASPP (Atrous Spatial Pyramid Pooling), a technique which applies many dilated convolutions in parallel with different rates of dilation. This way it manages to combine information on different scales in an efficient manner, thus improving the capability of the network to manage objects of different sizes and complex sceneries.

DeepLabV3 allows the choice of different backbones. The backbone is a convolutional network which acts as feature extractor, that is the part of the model which analyzes the input image to produce a useful representation for the task (in this case image segmentation).

The backbones associated with DeepLabV3 are, usually, MobileNetV3 Large, ResNet50 and ResNet101; for this reason and for the built-in compatibility with PyTorch these are the backbone that have been used.

The choice of the backbone is really important because the backbone influences the speed of the network, the amount of memory necessary for training and inference and, most importantly, the quality of the extracted features.

Moreover, even the same backbone can produce different results based on the set of pretrained weights used. In the specific case of the work produced the backbones used were all initialized with weights trained on COCO (a famous dataset for large-scale object detection, segmentation, and captioning dataset) and validated on the 20 categories belonging to Pascal VOC (bike, dog, bus, airplane and so on).

The choice of using weights pretrained on the same dataset allows for a more fair comparison between the three backbones. COCO is appropriate as dataset for the pretraining, of course there are others with images that look more like the ones contained in the dataset of the project, however COCO has been chosen because weights pretrained on it are already packaged in PyTorch, however, further experimentation (specifically that on DeepLabV3+ with ResNet101 pretrained on Cityscapes) has been performed to explore different possibilities.

1.1.1.1 ResNet

Residual Networks are deep neural networks introduced by Microsoft Research in 2015, they revolution introduced by them is the addition of direct connections between the beginning and the end of a block of layers (residual connection or skip connection) which

leads to an improvement in the quality of the information back-propagated to the deepest layers of the network.

The introduction of residual blocks and skip connections also minimizes the problem posed by vanishing/exploding gradient which is connected to the high number of layers of this network, thus facilitating the proper updating of the weights during the training process. This allows to get both the benefits of a deep network and the stability in training of a shallow one.

ResNets are particularly suitable for the process of recognition of very complex features thus facilitating subtle classification processes, like distinguishing between traversable grass and non-traversable low vegetation, which are two important classes in the problem faced. The way ResNets make this possible is by transferring directly some of the raw features to the final layers, thus combining both simple and complex features in parallel.

Based on the number of layers in the backbone we distinguish between

- **Resnet50** (with 50 layers);
- **Resnet101** (with 101 layers).

The two backbone are really valid, with Resnet50 being faster but less precise and Resnet101 being slower and heavier on the memory but more precise on fine details.

1.1.1.2 MobilenetV3 Large

MobilenetV3 Large is by far the lightest of the three backbones, it's mainly focused on optimization for mobile use, as the name suggests.

Of course this aggressive optimization comes at a cost, it compromises computational complexity with performance, thus achieving worse mIoU score and worse capability of discerning between similar classes than the other backbones used. The main reason this backbone was chosen is to experiment with higher batch sizes, impossible with already demanding networks.

1.1.2 DeepLabV3+

DeepLabV3+ is an evolution of the image segmentation network DeepLabV3.

After many experiments conducted with DeepLabV3 a research for models pretrained on datasets more similar to ours led us to discover a repository with a an implementation of DeepLabV3+, with ResNet101 as backbone, pretrained on Cityscapes. The striking resemblance between Cityscapes and our dataset is the driving force that led to further experimentation with DeepLabV3+.

1.1.3 BiSeNetV2

The choice to experiment with BiSeNetV2 stems from our research for a semantic segmentation network pretrained on a dataset that would reflect our needs.

After a long search we have found pretrained weights for BiSeNetV2 on the following datasets:

- RUGD;
- Rellis3D.

Both datasets have images with a similar setting to those that our network will have to classify; we have therefore decided to fine-tune BiSeNetV2 two times, one for each set of pretrained weights.

Network Source: https://github.com/CoinCheung/BiSeNet

1.2 Development Enviroment

1.2.1 Hardware

The following hardware resources have been used for training:

- server with a Nvidia A100-SXM4-40GB offered by the university;
- laptop with a Nvidia 1650ti GPU;
- laptop with a Nvidia 4060 mobile GPU;
- server with a Tesla T4 GPU offered for free on Google Colab.

1.2.2 Software

Regarding the software the training script has been written with Python, specifically using PyTorch, since it has been the focus of the course.

Other important libraries used include:

- albumentations for data augmentation;
- iterstrat to guarantee equal splitting of the samples between training and validation set based on the classes in the images.

1.3 Dataset

1.3.1 Description

The dataset comprises of 932 samples, each consisting of:

- an RGB image which is a photo taken by a vehicle on a rural-road;
- a map of semantic segmentation where each color represents a specific class.

The classes present in the dataset are: sky, rough trail, smooth trail, traversable grass, high vegetation, non-traversable low vegetation, puddle, and obstacle.

The labels are not actually images, as said they are maps of semantic segmentation, each color is actually a number corresponding to a class. The correspondence is shown in the following map (Figure 1).

```
CLASS_CODE_TO_CLASS_NAME_MAP = {
    0: "background",
    1: "smooth trail",
    2: "traversable grass",
    3: "rough trail",
    4: "puddle",
    5: "obstacle",
    6: "non-traversable low vegetation",
    7: "high vegetation",
    8: "sky",
}
```

The colors visualized when opening the label with a specific program may vary but with many they are as shown in the following map (Figure 1).

```
COLOR_TO_CLASS_NAME_MAP = {
    (255, 255, 255): "background",
    (178, 176, 153): "smooth trail",
    (128, 255, 0): "traversable grass",
    (156, 76, 30): "rough trail",
    (255, 0, 128): "puddle",
    (255, 0, 0): "obstacle",
    (0, 160, 0): "non-traversable low vegetation",
    (40, 80, 0): "high vegetation",
    (1, 88, 255): "sky",
}
```

The data processing necessary to feed the data to the model includes the conversion of the label in a numpy array, when there is a need to visualize it the array is converted back into an image, however this process changes the color map (Figure 2). To facilitate the interpretation of the data a new map has been produced which illustrates the correspondences between the original colors and the new colors which are visualized at the end of the training pipeline and therefore in the images in this document.

Figure 1 - Code-class-color map

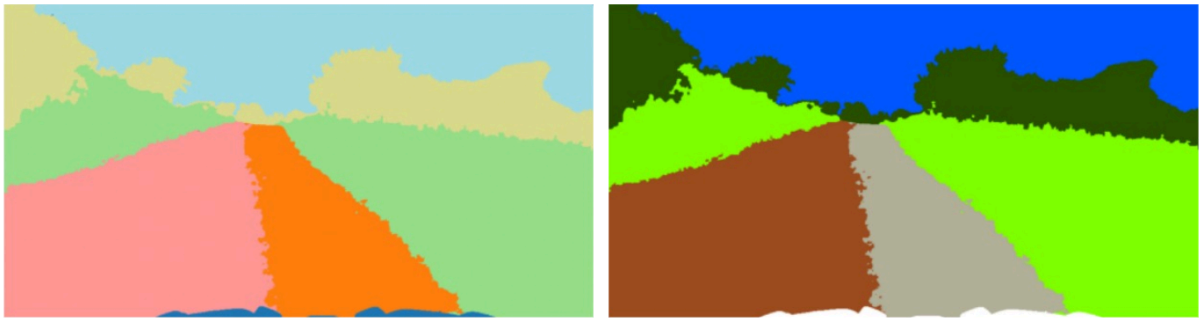


Figure 2 - Old to new colors map

```
OLD_TO_NEW_COLORS = {
      "bianco" = "blu",
      "blu" = "azzurro",
      "marrone" = "rosa",
      "grigio" = "arancione",
      "verde chiaro" = "verde",
      "verde scuro" = "giallo",
      "verde" = "grigio",
      "magenta" = "prugna",
      "rosso" = "grigio scuro",
}
```

1.3.2 Pre-processing

The pre-processing of the dataset involves two main steps:

- identifying wrong labels to exclude the corresponding samples from the training process, this proved to be vital to improve performance;
- apply data augmentation, in general to all samples but also specifically targeting images containing rare classes.

1.3.3 Split (train/val)

Initially the splitting procedure was performed just by randomly dividing the available data into two sets.

However to achieve a more reliable estimate of the performance of the model it was decided to switch to a more refined method.

The updated code divides the dataset in training and validation set in a balanced manner with respect to the classes.

The method build_presence_matrix creates a binary matrix of size (n_sample, n_classes) in which every line represents a directory (a sample from the dataset) and each column represents a class. The value in a cell is 1 if the class is present in the corresponding labels.png file, 0 otherwise.

MultilabelStratifiedShuffleSplit executes a stratified split of the dataset, which means that it divides the data in train and val while maintaining the distribution of the classes as similar as possible between the two sets; this assures that rare classes are present in both sets.

A random split can produce an imbalanced presence of rare classes between the two set, in the worst case scenario certain classes can be completely absent in the validation set, the stratified splitting allows to avoid these issues altogether.

2. Solution Design

In this chapter the choices made and the activities performed are explained in detail. Particular attention is directed towards the choices that have produced the most significant outcomes, the main issues encountered and the way they were mitigated.

2.1 Preliminary analysis and choice of the starting network

At the beginning of the project it was performed a comparative analysis of different neural network architectures for image segmentation. The analysis aimed at identifying the best possible starting point for the system to develop.

The networks tested are:

- DeeplabV3 with ResNet101 as backbone pretrained on Coco;
- DeeplabV3 with ResNet50 as backbone pretrained on Coco;
- DeeplabV3 with MobileNetV3 Large as backbone pretrained on Coco.

Additional tests have involved:

- DeepLabV3Plus with ResNet101 as backbone pretrained on Cityscapes;
- BiSeNetV2 pretrained on RUGD;
- BiSeNetV2 pretrained on Rellis3D.

The Mean Intersection over Union (mIoU) achieved in the initial phase of the research for each of these architectures are listed in the following table.

I risultati ottenuti in termini di Mean Intersection over Union (Mean IoU) sono riportati nella seguente tabella:

Rete	Backbone	Esperimento	Mean IoU
DeepLabV3	ResNet50	DLRN50000	0.5070
DeepLabV3	ResNet101	DLRN101000	0.5117
DeepLabV3	MobileNetV3Large	DLMN000	0.5000
DeepLabV3Plus	ResNet101 (pretrained Cityscapes)	DLPRN1010000	0.5009
BiSeNetV2	Rugd	BSNV20001	0.49
BiSeNetV2	Rellis3D	BSNV20002	0.41

It is important to consider that these results are a starting point, from this point forward strategies and techniques have been used to improve performance, as well documented in the following.

The analysis of this first batch of data shows that the architecture DeepLabV3 with ResNet101 as backbone is the one that achieved the best results during validation. In

particular, the experiment DLRN101000 has reached the minimum Validation Loss during epoch 7 with a Mean IoU of 0.5117. The maximum Mean IoU was, instead, 0.5985 during epoch 26. The choice has been to proceed considering the results achieved during the epoch with the lowest validation loss, the aim of this choice is to offer a more truthful view of the performance of the model during test phase and avoid overfitting.

Considered the results, the performance that were compared with a more focused approach are those of DeepLabV3 with ResNet50 and with ResNet101. The focus was on making sure that both of them respected the constraint of 5GB of VRAM. It was feared that a network with 101 layer could have been too heavy.

However, the experiment proved that both architecture are well inside the limits imposed, with DeepLabV3 + ResNet50 only using 1.54GB of memory and DeepLabV3 + ResNet101 using 2.62GB. In both cases memory consumption is acceptable.

All the observations made regarding memory consumption are, of course, relative not only to the models used but also to the hyperparameters chosen as most beneficial for the learning process. All these details can be traced thanks to the experiment code shown in the table.

Among the hyperparameters the most important for memory consumption are batch size, which is set to 4, and crop size, which is set to 256x256.

All of this taken into consideration, the final choice was to continue experimenting on DeepLabV3 with ResNet101 since it is the best compromise of accuracy and stability during training. This configuration has been the baseline for the successive development of the project.

2.2 Underrepresented classes

During the analysis of the performance of the model, a significant problem has been observed with some classes of the dataset which appear to be unrepresented. In particular, the results achieved during validation show a clear disparity between the IoUs for the different classes, with some of them achieving way higher performance compared to the others.

In the table are present the IoU values obtained in the experiment DLRN101000 during the seventh epoch, where the Validation Loss is at its lowest (0.5766):

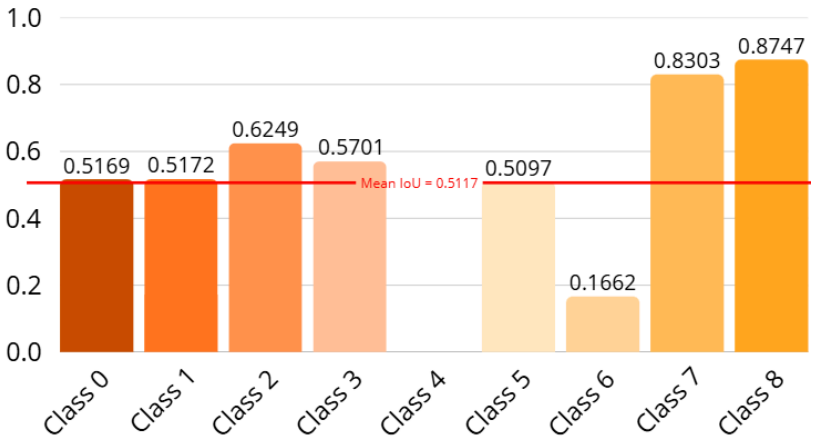


Figure 3 - Sample containing class 5 (obstacle)

Figure 4 - Sample containing class 4 (puddle)

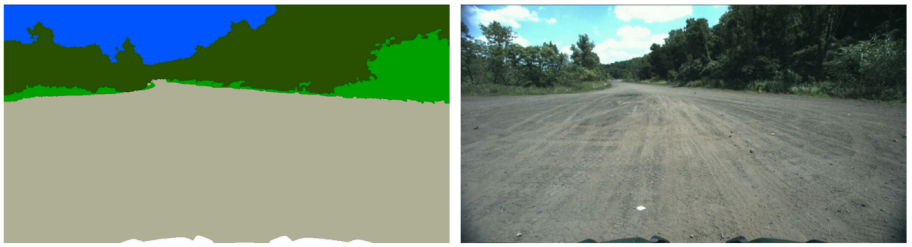


Figure 5 - Sample containing class 6 (non-traversable low vegetation)

It is clear that class 4 (puddle) (Figure 4) never gets correctly classified from the model in the initial phase, thus achieving IoU equal to zero. Similarly, also classes 5 (obstacle) (Figure 3) and 6 (non-traversable low vegetation) (Figure 5) show limited performance compared to the average.

These three categories are underrepresented in the dataset: they are rarely present in an image and even if they are they usually occupy a relatively small area. This imbalance has a direct impact on the performance of the model.

The consequence is a lower Mean IoU, which doesn't reflect the capability of the model to properly segment the main classes, which comprise the majority of the images. This calls for the identification of strategies to manage this imbalance.

2.2.1 Weighted Loss

To face this issue the strategy chosen is weighting the loss function with the objective to penalize more the errors on rare classes and incentivize the model to learn how to properly segment them.

The Loss Function is still Cross-Entropy but initialized with a vector of weights corresponding to the classes chosen heuristically in a way that is inversely proportional to the frequency of occurrence of the classes in the dataset.

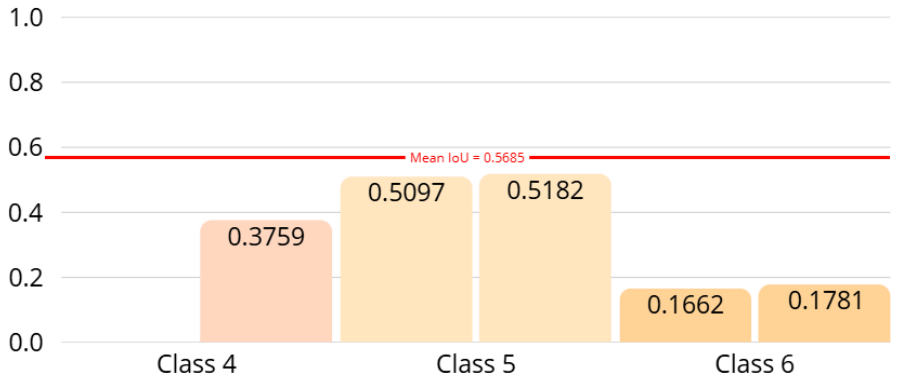
The weights introduced for the nine classes in experiment DLRN101001, after a tuning phase, are:

```
[1.0, 1.0, 1.0, 1.0, 5.0, 4.0, 3.5, 0.8, 0.8]
```

This way the classes puddle (class 4), obstacle (class 5) and non-traversable low vegetation (class 6), that previously negatively impacted the mIoU, are made more important in the objective function.

The efficacy of this solution is confirmed by the new results achieved during validation. During the eighth epoch, where the Validation Loss is at its lowest with a value of 0.7024, it is possible to observe an improvement in the mIoU, which reaches 0.5685.

In particular the improvement in the 3 classes deemed problematic is clear and can be seen in the following graph:



These results show that the introduction of class weights has favoured a more balanced learning process, improving the capacity of the model to correctly recognize all categories, even the less frequent ones, without compromising performance on more common classes.

2.2.2 Augmentation and oversampling

To improve performance on less represented classes experiment DLRN101002 introduces a new strategy based on two combined actions:

1. Selective oversampling
2. Targeted data augmentation

2.2.2.1 Selective oversampling

During the loading of the dataset every image has been analyzed to verify the presence of at least one of the classes considered rare. The images containing these classes have been artificially duplicated during the training phase with the objective to increase the relative frequency of these classes without modifying the original distribution of the dataset. This approach helps in compensating the imbalance in the data in a controlled and localized manner, avoiding distortions in the evaluation metrics.

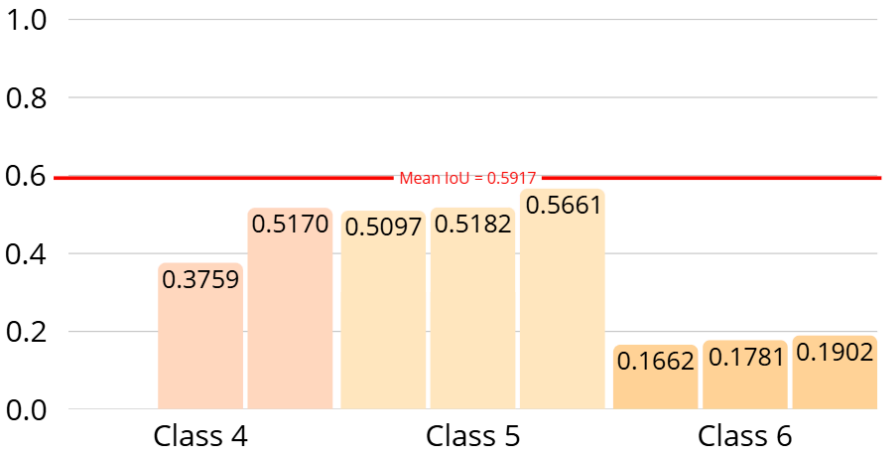
2.2.2.2 Targeted data augmentation

Images containing rare classes have been augmented using a special set of transformations, designed to expand visual variations between the samples without altering the semantic of the scene. The transformations include:

- Random horizontal flip: useful to double the spatial variety of the classes;
- Brightness and contrast variations: introduced to simulate different lighting conditions, which is coherent with images acquired in natural settings;
- Resize and normalization: these were kept coherent with the general preprocessing to grant compatibility with the backbone.

For the images that do not contain rare classes only a base augmentation was applied (resize, normalization and conversion to tensors), so to not introduce unnecessary noise in the dataset regarding already well represented classes.

This combination of oversampling and targeted augmentation allowed a significant improvement in the amount of exposure of the model to critical classes, thus improving its capacity to properly classify them. The results registered confirm the efficacy of the approach: during the fourth epoch the model has reached a Mean IoU of 0.5917, with a significant increment in the performance relative to classes previously considered problematic.



In particular, the class puddle (class 4) has reached an mIoU of 0.5617, starting from an initial value of 0.0000 in the first experiments. The classes obstacle (class 5) and non-traversable low vegetation (class 6) benefited from the strategy also, with an IoU of respectively 0.5661 and 0.1902.

These improvements have been achieved without compromising the general equilibrium of the learning process, thus maintaining good results on more recent classes also.

2.3 Evaluation of the Impact of Pretraining Dataset Selection on Semantic Segmentation Performance

In order to improve the performance achieved many different paths have been analyzed.

The most important results have come from the analysis of DeepLabV3 with the backbones previously introduced, all pretrained on COCO.

However, it has been explored the possibility that the performance of the model was being held back by its inadequate capability to properly distinguish low level features.

Specifically, it was thought that the weights used for the backbone may not have been ideal because of the specifics of the dataset they were trained on, that is COCO.

The idea was to look for datasets which contained images more similar to those in the main dataset of the project, this research landed on the following ones:

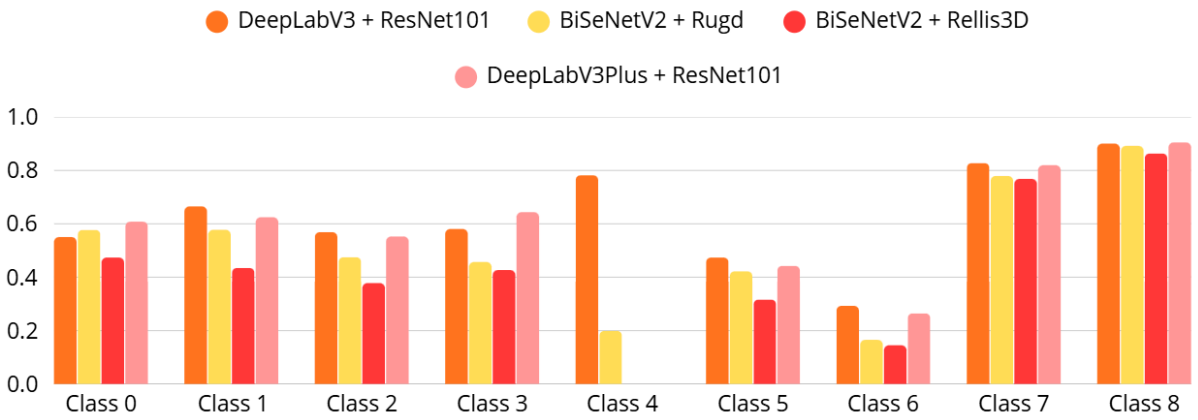
- RUGD;
- Rellis3D;
- Cityscapes.

It would have been interesting to compare the performance achieved by the same model (DeepLabV3 + resnet101/resnet50/MobileNetV3 Large) pretrained on different datasets. Unfortunately this proved to be prohibitive in the context of the time and computation constraints of this project and is therefore postponed to future research.

In the context described it has been decided that, while not ideal, it could have been of value an analysis of the performance achieved with these datasets, even if measured on models with different architectures.

Specifically, the following models have been examined and tested:

- DeepLabV3Plus with ResNet101 pretrained on Cityscapes, which achieved its best result in the experiment DLPRN1010007 with mIoU equal to 0.5518;
- BiSeNetV2 pretrained on RUGD, which achieved its best result in the experiment BSNV20001 with mIoU equal to 0.49;
- BiSeNetV2 pretrained on Rellis3D, which achieved its best result in the experiment BSNV20002 with mIoU equal to 0.41.



However, as shown in the experiment's log, none of these solutions proved to be valid.

These experiments have shown that, while theoretically important, the difference in performance using a more “on theme” dataset is non-existent, at least in the context of the specific experiments run for this project.

The conclusion reached about these unexpected results is that, while the nature of the dataset used for the learning of the feature extraction process is important, as long as it is general enough that it allows for the distinction of all the low level features needed for this problem it doesn’t make a significant difference in the distinction of high level features (like the classes of the segmentation problem faced). The dataset used for fine-tuning is therefore the most important one.

2.4 Rough contours

During a qualitative analysis of the segmentation results, a recurrent problem was observed: the contours in the predicted mask appear overly smooth compared to the ground truth (Figure 6).

One of the hypotheses formulated is that this problem was caused by the loss of spatial information that happens when the images are resized to 256x256 pixels in order to comply with the memory constraints imposed. This downscaling may have compromised the conservation of fine details that are fundamental for a proper segmentation of the contours.

To verify this hypotheses a new experiment was conducted (DLRN101004), this time resizing the images to a higher resolution, 512x512 pixels, while keeping all other hyperparameters the same. The objective was to give the model inputs richer in visual details, which should improve the capability of the model to identify more precise and coherent contours.

However, the experiment didn’t produce positive results. The higher resolution combined with the heavy backbone ended up exceeding the memory constraints of the project. For this reason it was decided to go back to a lower resolution although the potential of a higher resolution is recognized, at least on a qualitative level.

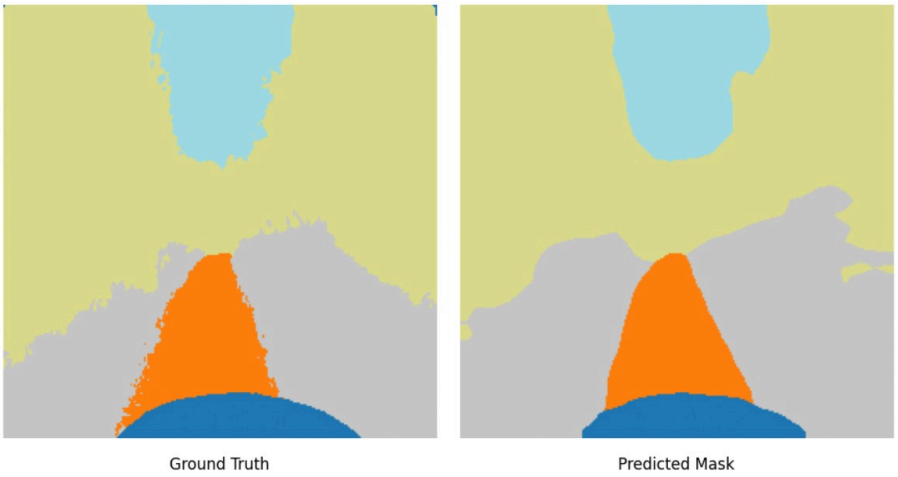


Figure 6 - Predicted Mask with smooth contours

2.5 Recognition of small objects

Another critical point emerged during the development of the model concerns the difficulty in recognizing very small objects (like obstacles and puddles) and, at the same time, larger objects in the scene. This limitation is often tied to the capability of the model to capture contexts at different scales.

To face this problem, different experiments were performed with the module ASPP (Atrous Spatial Pyramid Pooling) of the network. This allows the model to simultaneously capture local and global information, improving its sensitivity both towards fine details (like small objects) and larger structures.

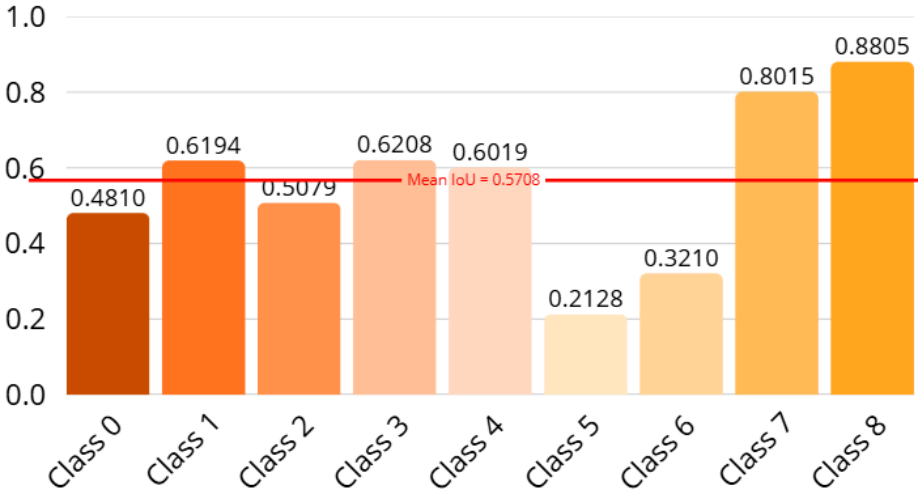
Different experiments were conducted, varying:

- number and size of the convolutional kernels;
- dilation rates.

The most efficient approach proved to be the one adopted in the experiment DLRN101011 which is based on an ASPP module with three dilated convolutions and a successive elaboration through two convolutions with kernel 5x5, designed to increase even more the field of view of the model while maintaining granularity.

The dilation rate chosen after careful tuning are [3, 7, 3], a configuration which allowed good multi-scale coverage.

The integration of the custom ASPP led to an improvement of the classification both of small and big classes, with a mean IoU of 0.5708 during the second epoch. Which also shows very fast convergence. Here are the results measured:



The addition of the ASPP has improved the flexibility of the model without compromising the general equilibrium in its capabilities.

2.6 Problems with the dataset

Another obstacle faced in the development of the model has been the quality of the dataset, more so than its limited size. Manually analyzing the samples in the training set led to the individuation of samples with labels that were either imprecise or wrong altogether(Figure 7-8), which of course stunts the learning of the model.

These misleading samples have been removed from the dataset in order to avoid introducing useless noise in an already undersized dataset. Of course, since the dataset is small, the wisest choice would have been relabeling, but this proved incompatible with the time constraints of the project and is therefore postponed to future work on the subject matter.

As stated not every problematic sample had a completely wrong label, in some cases the label was just imprecise (Figure 9-10), especially around the borders of the background. These mistakes are of course less severe but they still introduce semi-systematic noise which is problematic. To avoid reducing too much the size of the dataset these samples haven't been deleted but it was instead decided to implement regularization strategies to contain their effects.

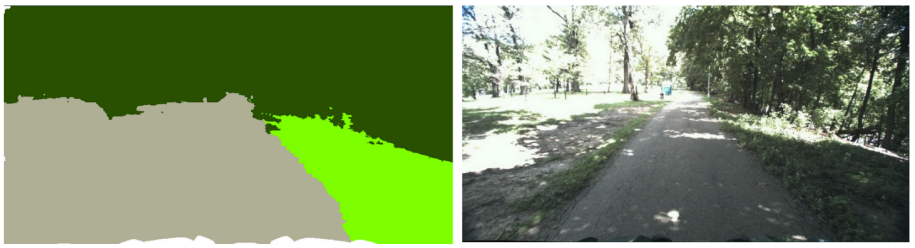
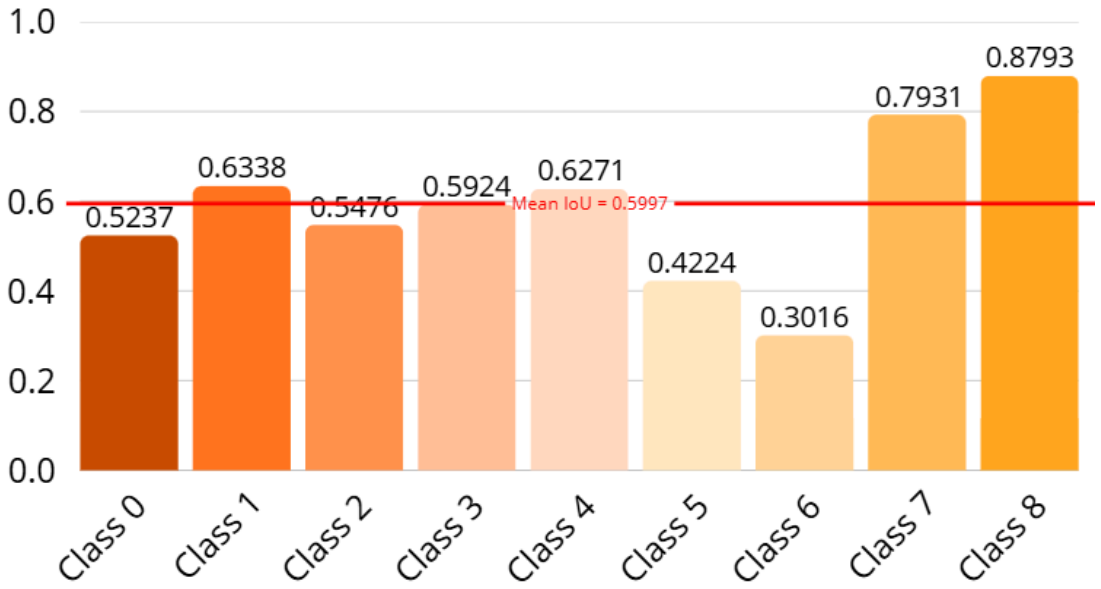


Figure 7 - Sample number 0000

Figure 8 - Sample number 0007

To manage this kind of noise the classifier head of the model was customized to add a dropout layer. A dropout layer randomly deactivates parts of the network during training, this helps with overfitting and makes the model more robust to imperfect data.

Adding the dropout layer required some tuning to decide the dropout probability, tests were made with values between 0.2 and 0.02. The results showed that higher values (like 0.2) are too aggressive in a data-poor context and end up compromising the model's ability to generalize. On the contrary, a lower dropout probability, like 0.05 which was used in the experiment DLRN101012 works better. Specifically 0.05 has been the value with the best results, leading to a mean IoU of 0.5997 which is the highest value among all the experiments conducted.



These settings produced a model that is stable, able to generalize and able to segment even when there is noise, for these reasons this model is the starting point for future phases of the project.

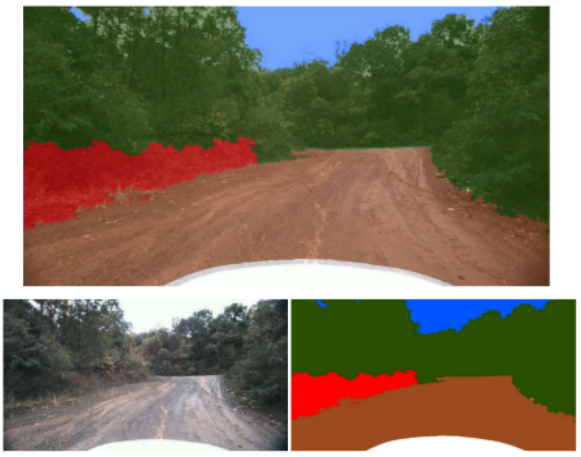


Figure 9 - Sample number 0008

Figure 10 - Sample number 0256

2.7 Confusion between the classes Rough trail and Smooth trail

During the training of the model the analysis of the predictions produced showed a high degree of confusion between the classes rough trail (class 3) and smooth trail (class 1) (Figure 11). This behaviour can be explained by the fact that the two kinds of trail share many visual features, like color and spatial position in the image (they are both usually in the lower half of the frame). These similarities make it more difficult for the model to understand the differences between the two, which leads to ambiguous predictions.

2.7.1 Class-focused loss boost

To face this issue the solution proposed is the introduction of a targeted loss strategy, based on a combination of Cross-Entropy Loss with weighted classes and an additional penalty for cases of direct confusion between the two classes of interest.

The module `FocusedCrossEntropyLoss`, introduced in the experiment DLRN101018, introduces a penalization term that is proportional to the sum of the probabilities wrongly assigned to the two target classes when one of the two gets confused with the other.

This approach has proven to be counterproductive because the model learnt to systematically avoid predicting both classes involved in the penalty (Figure 12-13), preferring alternative classes so as to avoid the additional penalty. This caused the mean IoU to fall to 0.3508.

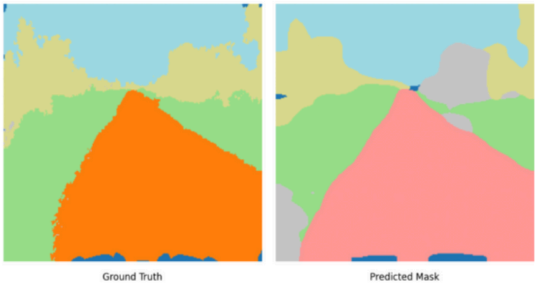


Figure 11 - Prediction with confusion between classes 1 and 3

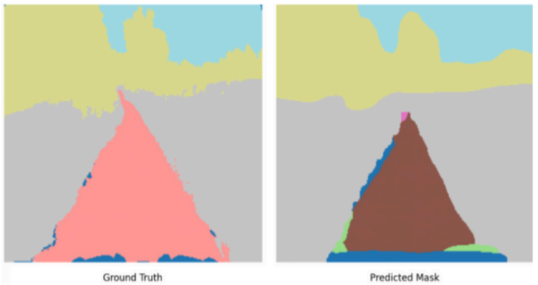


Figure 12 - Wrong prediction of class 1

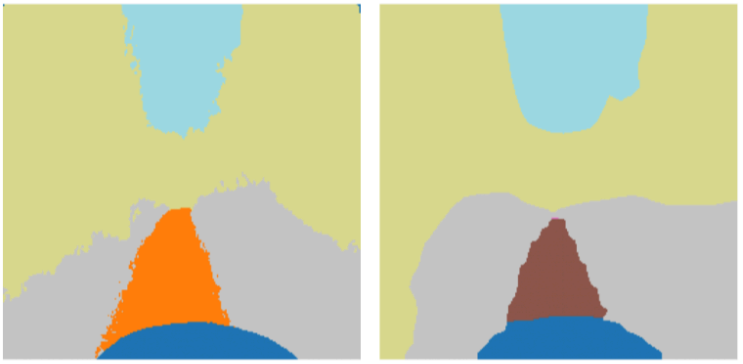


Figure 13 - Wrong prediction of class 3

2.7.2 Texture augmentation

To face the persistent confusion between the classes "rough trail" (class 3) and "smooth trail" (class 1), in the experiment DLRN101020 (Figure 14) it was introduced a scheme of conditional data augmentation. The idea is automatically identifying, during data loading, the images which contain both classes, and in these cases it's applied a special texture-focused augmentation which is designed to increase the variety and visual complexity of the elements of the rough trail, making it more recognizable. The transformations applied include blurring, gaussian noise, sharpening, brightness variations and contrast. All of this with the objective of increasing the robustness of the model with respect to slight surface variations which are the only thing that distinguish class 3 and 4.

Differently from the penalty introduced through targeted loss, this strategy doesn't block the model from choosing the classes that are more difficult to distinguish but it stimulates to learn more discerning representations, by taking advantage of a higher local diversity in the input data.

The augmentation is only applied selectively in relevant cases, leaving the pipeline for standard images and that for images containing rare classes the same.

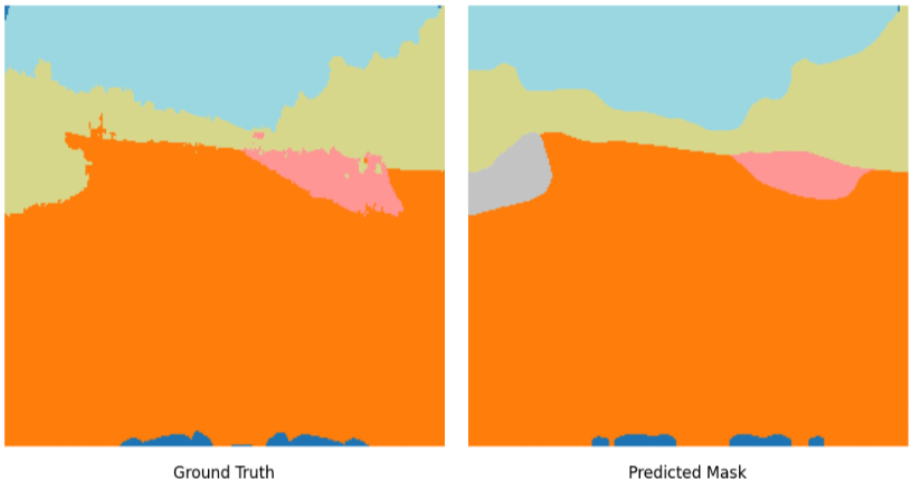
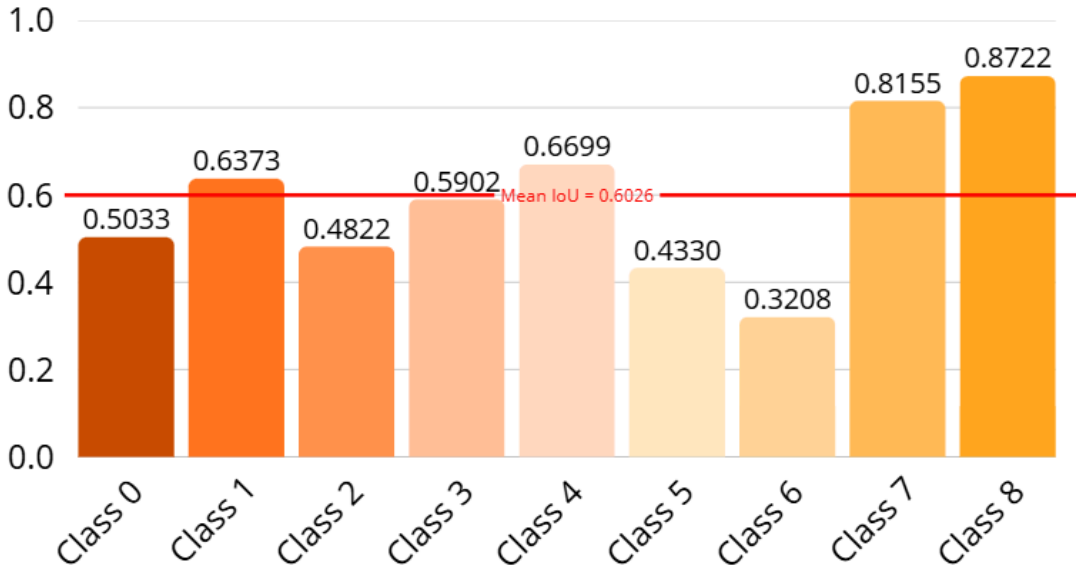


Figure 14 - Correct prediction of classes 1 and 3

The results registered show a tangible improvement, during epoch 3 the Validation Loss got as low as 0.6834 while the Mean IoU was 0.6026.



These results suggest that targeted augmentation produced a generally positive result, it improved local discriminabilità of the classes with semantic ambiguities, which in turn favored better generalization without collateral effects on other classes.

2.8 Experiments comparison

The process of development of the model has been strongly led by an iterative experimental approach, as summarized in the decision tree of the experiments (Figure 15). Each branch of the tree represents a change that has been introduced, evaluated with the mean IoU achieved during the epoch in which the Validation Loss was at its lowest value.

Lighter boxes indicate choices that have been confirmed and integrated in the final pipeline, while dark ones represent experiments abandoned because of lack of improvements or incompatibility with the constraints.

As stated in previous sections, apart from the one chosen for submission alternative models have been tested, like BiSeNetV2 (pretrained on RUGD and pretrained on Rellis3D) and DeepLabV3Plus + Resnet101 (pretrained on Cityscapes). However:

- BiSeNetV2, while really fast, showed subpar performance (IoU ≈ 0.49), probably because of its structure being too simple for a dataset with samples as details-rich as the one used.
- DeepLabV3Plus showed results comparable to those of DeepLabV3 (with the same backbone) but it occupied more memory. As said COCO is probably already enough to allow the network to recognize low level features.

Ultimately, the comparison between architectures confirmed that using DeepLabV3 + ResNet101 is the best solution since it granted the best balance between accuracy, robustness and adaptability to the specific domain of the project.

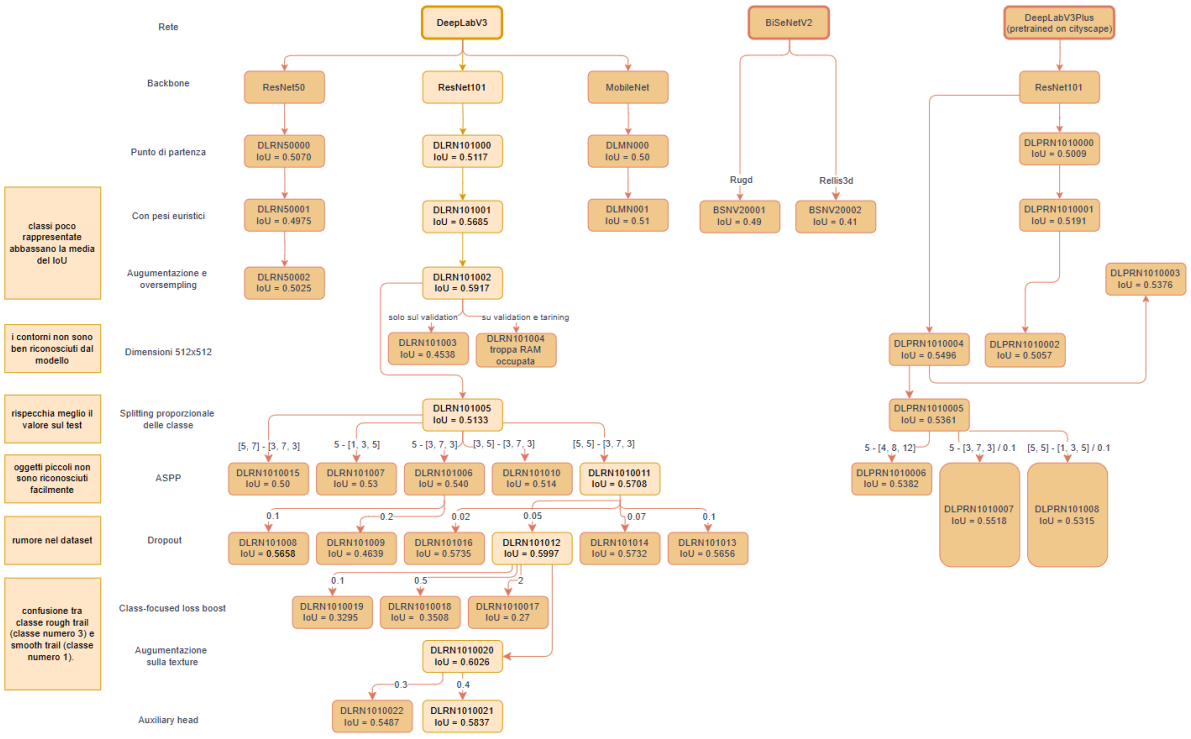
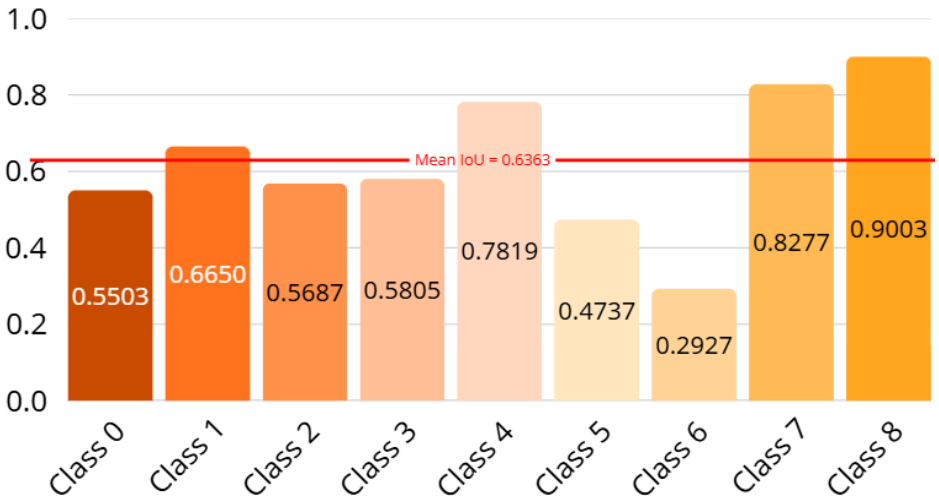


Figure 15 - Albero decisionale degli esperimenti

Starting from the base model DeepLabV3 + ResNet101 (IoU = 0.5117), different techniques have been progressively added to face specific problems: imbalance classes, blurry contours, small objects, noise in the dataset and confusion between classes semantically close. Moreover, since the architecture also includes an Auxiliary head, it has been activated and associated with a custom classifier to grant higher stability and robustness. The performance of the model are those reached in the experiment DLRN101021.



For the final project submission it was decided to not choose the weights that the model achieved during the epoch with the lowest validation loss but rather those that the model achieved during the epoch with the highest IoU but still with a value of validation loss close to the minimum. This trade-off has been chosen because while indicative of the general

quality of the model, Cross Entropy Loss doesn't necessarily reflect the main metric of importance for the project, that is the IoU.

This choice has allowed to submit a model with just a slightly higher loss but with a significantly higher mIoU, especially boosted by higher scores on rare classes.

It's important to clarify that the mean IoU values reported in the experiments could be higher than the ones achieved during the test phase. This is not because of overfitting but rather because there is a slight difference in the formula used to calculate the IoU:

- during validation:
  the IoU is calculated on all images combined, this leads to a single IoU value per class that gets then averaged to achieve the mean IoU. This way, bigger images or more frequent classes weigh more on the final result, which slightly boosts the score since more frequent classes are also the ones on which the model performs better. The Mean IoU score estimated with this metric is equal to 0.63.

- during testing:
  the IoU is computed singularly for each image, then the results are averaged. In this case, every image has the same weight, even if a class is only present because of a few pixels. The Mean IoU score estimated with this more severe metric is equal to 0.55.

It is therefore expected that the results achieved during the final testing phase will be slightly more conservative than the results reported. However, this does not compromise the validity of the choices made during the experimental phase of the project, since all evaluations were made coherently with the same formula to compute IoU and always choosing the model with the lowest validation loss.

2.8.1 Memory efficiency

An important aspect of the project was memory optimization targeted at complying with the memory constraints imposed. The final model appears to be light and compatible with low-resources contexts like edge computing.

- During training the highest amount of memory occupied on the GPU Tesla T4 provided by Colab was 2.7GB, well within the 5GB limit.

- During inference the highest amount of memory occupied on the GPU Tesla T4 provided by Colab was 0.52GB, well within the 4GB limit, which means the model could be potentially deployed on embedded hardware.

This efficiency has been achieved without compromising performance thanks to precise architectural choices, like the low batch size and the low input resolution.

3. Implementation and description of the functionalities

The system has been completely implemented with Python in Google Colab. The whole operative flow is organized into well distinguished phases: loading and cleaning of the dataset, building of the neural network, training and visualization of the result.

3.1 Preparation of the dataset

The work environment is initially connected to Google Drive, from which it loads the dataset organized in subfolders, each containing an RGB image and the corresponding label map. An automatic system excludes the sample known for having wrong labels (like samples with roads labeled as puddles).

Next, a multilabel matrix gets generated to represent the presence/absence of every class in every image. This matrix is then used to perform a multilabel stratified split in training and validation set, which grants that even rare classes will be equally distributed in the two sets.

3.1.1 Data augmentation and management of rare classes

Before training, a pipeline gets activated which performs conditional data augmentation:

- if an image contains at least a rare class (puddle, obstacle, non-traversable low vegetation), it gets duplicated 2 times and then undergoes augmentation targeted only to these images (horizontal flip, variation of brightness and contrast);
- Normal images are instead elaborated through a simpler preprocessing (resize and normalization).
- A third pipeline is activated only for specific cases which are considered ambiguous (contemporary presence of rough trail and smooth trail in the same image, it applies transformations based on blurriness and noise to improve the robustness of the model in confusing situations.

3.2 Model customization

The model is based on DeepLabV3 with Resnet101 as backbone pretrained on COCO. The backbone has been left untouched while the classification head was customized with:

- dilation [3, 7, 3] for the ASPP module, to better identify the context at different spatial scales;
- two successive convolutions with kernel 5x5 to increase the receptive field;
- a dropout layer (p = 0.05) to increase the robustness of the model in presence of noise in the input data.

The model also includes an **auxiliary head**, activated explicitly to optimize learning in the intermediate layers of the network. This secondary branch helps stabilizing the gradient during training and helps with the model convergence, in particular in the first epochs.

The output of the auxiliary head is combined with that of the main head in the loss function, using a weight for the auxiliary head. This approach is used in the final configuration, because while it slightly increases the complexity of the model it also proves to be beneficial for its robustness, especially for less represented classes.

3.3 Training and validation

The training is managed by a Trainer class which implements:

- a complete cycle of training/validation for each epoch;
- monitoring of the main metrics (loss, per class IoU, mean IoU);
- automatic save of the best model based on validation loss and based on mean IoU;
- **early stopping** to avoid overfitting;
- tracking of the highest GPU use.

During training, the model gets evaluated on each epoch based on its performance on validation images. The metrics get saved in a log JSON and used to create graphs.

3.4 Testing

The system implements the predict() function as asked by the specifications, the function can:

- elaborate a batch of RGB images;
- automatically apply the preprocessing;
- instantiate the model;
- produce a mask as output with the same resolution as the input;

4. Conclusions and future perspectives

During the project different architectures have been tested but surely DeepLabV3 with ResNet50, DeepLabV3 with ResNet101 and DeepLabV3 with MobileNetV3 Large have been the most important.

The final proposed solution uses DeepLabV3 + ResNet101 because it has proven to be the most effective, offering the best balance between accuracy and memory consumption.

The customization experimented have had a significant impact:

- the auxiliary head has improved stability during training;
- the dropout with p = 0.02 has reduced overfitting with noisy input;
- the freezing of the backbone has been avoided to not limit the adaptability to the domain.

From the logs it is possible to observe that the use of class weights for the loss and the oversampling have led to great improvements regarding rare classes. The metrics converge in the span of 8 to 10 epochs.

Regarding future developments it is proposed to:

- revise manually or automatically the labels to correct evident errors and improve the quality of the dataset.
- pre-train the model DeepLabV3 + ResNet101 on datasets such as Cityscapes, RUGD and Rellis3D, to allow for more fair comparisons with the performance achieved with COCO, leaving all other hyperparameters the same.
- introduce pseudo-labelling techniques or automatic data cleaning to reduce the impact of noise on the labels;
- experiment with reinforcement learning, for example to dynamically adapt the training parameters or guide the learning process in presence of semantic ambiguities.

All these possibilities have not been explored in this project because of limitations linked to time and computational resources, but also to adhere to the guidelines of the project; however, they are interesting and realistic ideas that could concern future works.