# Computer Vision :
# Project 3

**Team members :**
Alessandro Belotti: ER1565
Ezra Kenig : ER1579

**Chosen subject :**
Semantic segmentation

**Dataset :**
https://www.kaggle.com/code/bulentsiyah/deep-learning-based-semantic-segmentation-keras

**References:**

- GitHub - qubvel/segmentation_models.pytorch: Segmentation models with pretrained backbones. PyTorch. (Segmentation models)
- IEEE Xplore Full-Text PDF: (Multi-Scale Attention Network)
- GitHub link

## Description of the problem

The goal of a **semantic segmentation** algorithm with an Aerial Semantic Segmentation **Drone Dataset** is to accurately classify and label each pixel in an aerial image captured by a drone, assigning it to a specific semantic category. Semantic segmentation involves dividing an image into meaningful segments or regions, where each segment corresponds to a specific object or class.
The Aerial Semantic Segmentation Drone Dataset provides a collection of annotated aerial images, where **each image has ground truth labels** (the mask) indicating the correct semantic class for each pixel. The algorithm's training involves learning from these annotated examples, enabling it to generalize and accurately segment new, unseen aerial images.

## The dataset

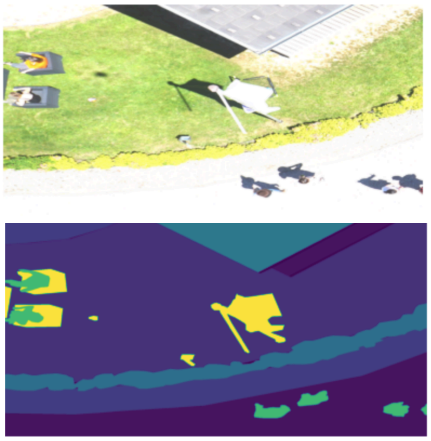
The Semantic Drone Dataset focuses on semantic understanding of urban scenes for increasing the safety of autonomous drone flight and landing procedures. The imagery depicts more than 20 houses from nadir (bird's eye) view acquired at an altitude of 5 to 30 meters above ground. A high resolution camera was used to acquire images at a size of **6000x4000px** (24Mpx). The **training set** contains **400 publicly available images**, and the **test set** is made up of **200 private images**.
We prepared pixel-accurate annotation for the same training and test set. The complexity of the dataset is limited to 20 classes, as listed in the following table.
Semantic **classes** of the Drone Dataset

- tree
- grass
- other vegetation
- dirt
- gravel
- rocks
- water
- paved area
- pool
- person
- dog
- car
- bicycle
- roof
- wall
- fence
- fence-pol
- window
- door
- obstacle

After the pre-processing each image has the size of [3, 1024, 2016].

**Data Augmentation**

To better train our model and reinforce its generalization ability, we applied a data augmentation pipeline using the **albumentations** framework, which provides multiple methods to modify our images. In our case, we chose to apply **Gaussian noise, flip transforms, contrast and brightness modification and grid shuffle**. All those transforms are applied, **both** to the images and the corresponding masks, according to a probability p used as a parameter in all the framework's function, so that all images are not modified the same way. Here are two examples of transformed images with their mask:



## Architectures and models

**Semantic_models_pytorch** is a python library which allows you to use different models architecture for multi class segmentation. In the specific we use:

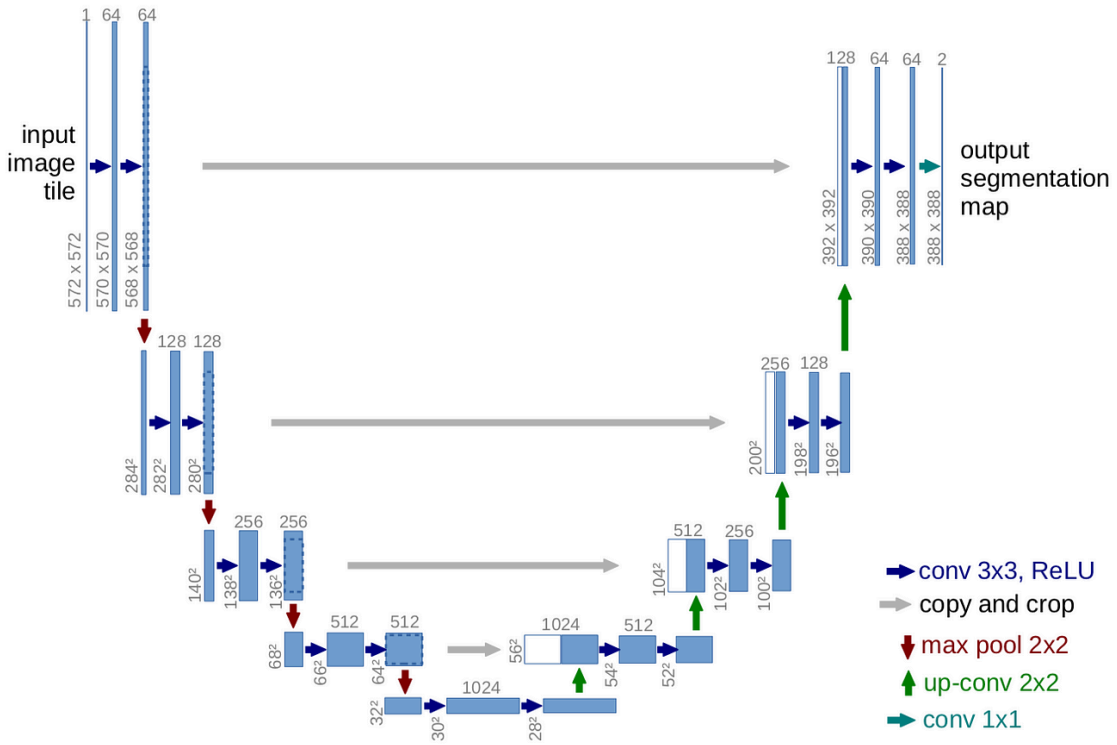
- Unet and MAnet as model architectures
- timm-mobilenetv3_small_075 encoders
- the encoder weights are initialized from the ImageNet dataset

## Model Analysis

For the encoder, we choose to use the smallest version of **MobileNet** because it's less time-consuming for the training phase.
The encoder is called timm-mobilenetv3_small_075, and it's trained on **Imagenet** dataset with 0.57 millions of parameters.
The ImageNet dataset is large-scale, containing millions of labeled images divided in thousands of categories. In the ILSVRC challenges, the dataset typically includes around 1.2 million training images, 50,000 validation images, and 100,000 test images.

## U-Net

The **contracting part** adopts a typical convolutional neural network (CNN) structure, with convolutional and pooling layers. These layers progressively reduce spatial dimensions while enhancing feature channels, enabling the network to learn hierarchical features.
The **expansive path** upscales learned features using transpose convolutions. This mechanism, called skip connections, merges high-level contextual information with detailed spatial data, mitigating information loss.

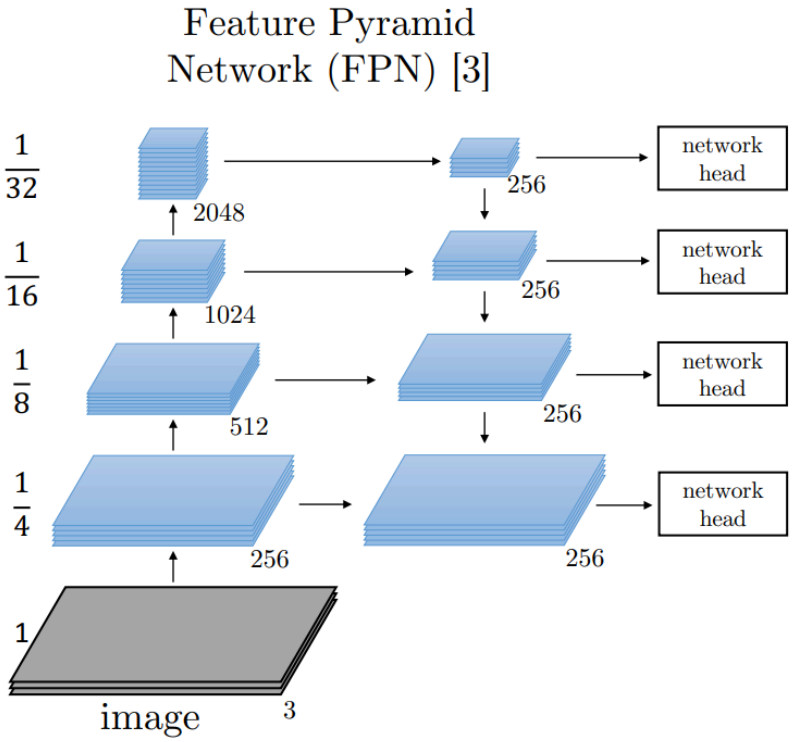


This network is already trained on a similar problem but with a different number of classes. So we **transfer learning** in order to modify the decoder part and train it on our custom dataset.
We are changing the last layer of the UNet encoder, the convolutional layer:

```
SegmentationHead(
  (0): Conv2d(16, 23, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): Identity()
  (2): Activation((activation): Softmax(dim=None))
)
```

**FPN net**

The primary goal of Feature Pyramid Network is to build a feature pyramid within a convolutional neural network (CNN) that captures rich semantic information at multiple resolutions. This is particularly beneficial for handling **objects of different sizes** in an image.
The idea is to create a **"pyramid" of features**, where each level of the pyramid corresponds to features at a different spatial resolution.
The architecture includes a **top-down** pathway, where higher-resolution feature maps are obtained by upsampling features from the coarser levels of the pyramid. This process helps maintain detailed spatial information.




The **bottom-up** pathway involves processing the input image through a convolutional backbone, such as a ResNet or a similar architecture, to extract high-level semantic features. These features are then used as the basis for constructing the top-down pathway.

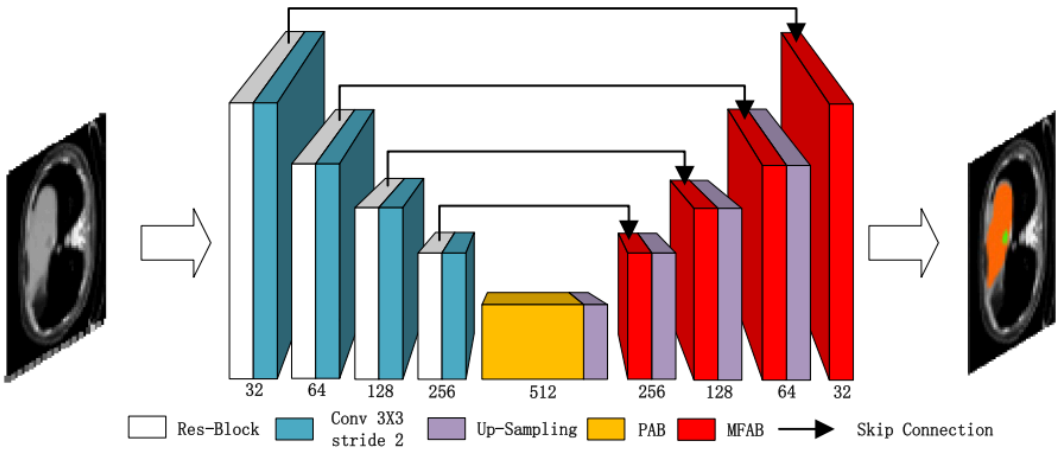
FPN introduces lateral connections that connect features from the bottom-up pathway to the corresponding levels of the top-down pathway. These lateral connections help **fuse**

**low-level and high-level information**, enabling the network to capture object details at different scales.
We act **transfer learning** on the same way of the previous network on the final layer used for segmentation:

```
SegmentationHead(
  (0): Conv2d(16, 23, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): Identity()
  (2): Activation((activation): Softmax(dim=None))
)
```

## MA-net

The self-attention mechanism is used in the MA-Net. Specifically, in the paper they use two blocks based on self-attention mechanism to capture spatial and channel dependencies of feature maps. One is **Position-wise Attention Block** (PAB), and the other is **Multi-scale Fusion Attention Block** (MFAB). The PAB is used to obtain the spatial dependencies between pixels in feature maps by a self-attention mechanism manner. The MFAB is used to capture the channel dependencies between any feature maps by applying an attention mechanism.




Also here we act transfer learning on the same way of the previous network on the final layer used for segmentation:

```
SegmentationHead(
  (0): Conv2d(16, 23, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): Identity()
  (2): Activation((activation): Softmax(dim=None))
)
```

## Training phase

We choose to implement three different models (one of them with Attention Layer), we define a fit function which iterates over the specified number of epochs.
For each epoch, it enters the training phase (`model.train()`) and iterates over batches of training data.
For each batch, it performs the following steps:

- **Forward pass**: Passes the input images through the model to obtain predictions.
- Computes the **loss** between the predictions and the ground truth masks using the specified criterion.
- **Backward pass**: Computes gradients with respect to the loss and performs a step of optimization to update the model's parameters.
- **Learning rate scheduling**: Adjusts the learning rate using the provided scheduler.

After completing each epoch's training, the function enters the validation phase (`model.eval()`).
It iterates over batches of validation data and evaluates the model on the validation set. Metrics such as Intersection over Union (mIoU) and pixel accuracy are computed for both training and validation sets.

After the training of the models, we choose to store the history and the model to make the code easily reusable in terms of time and resources.

**Hyperparameters**

The following are the parameters we chose:

- **learning rate** (step size during optimization): we choose 0.01, and we set a **learning rate scheduler** to improve the convergence.
- **batch size** (number of samples used in each iteration of training): we choose 3 because otherwise we have a GPU memory error in Colab
- **epochs** (how many times the entire training dataset is passed through the model during training): we set this to 8 epochs for limiting the training duration
- **weight decay** (regularization term added to the loss function to penalize large weights): we choose a common value of 0.001
- **Optimizers**: we choose first to train the models with AdamW (with weight decay) and then (after choosing the best model) try also RMSprop and SGD.

**Metrics**

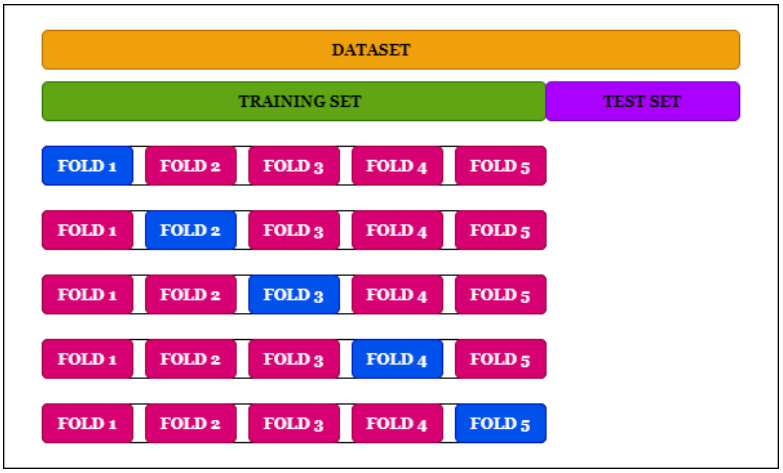
The metrics we're using are the following:

- **accuracy**: ratio of correctly predicted instances to the total number of instances in the dataset
- **loss**: mismatch between the model's predictions and the actual (ground truth) values. Due to the fact that we're doing a multiclass classification problem we're using **CrossEntropyLoss** first, and after choosing the best model we're testing these:
  - **DiceLoss** (the loss It is based on the Dice coefficient which measures the similarity between two sets)
  - **DiceCrossEntropyLoss** (is composite loss function that combines the properties of Cross-Entropy Loss and Dice Loss. It aims to leverage the strengths of both loss functions for tasks like semantic segmentation)
- **mean Intersection over Union**: measures the overlap between the predicted region and the ground truth region for a given object or class

**Cross Validation**

As a way to improve the performances of our model and its generalizing capacity, we implemented a k-fold cross-validation algorithm, which consists in dividing our training set (training data + validation data) in k subset, and using one of these subsets as a validation

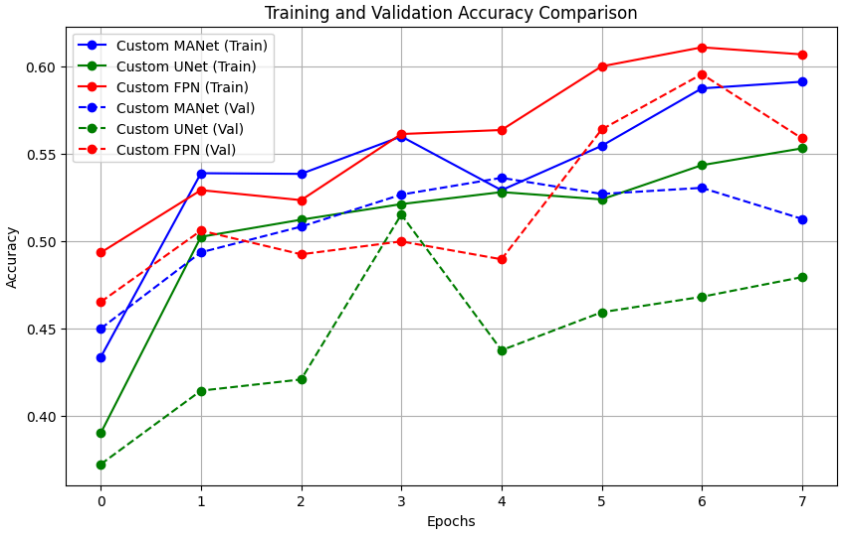
set. At each epoch, a different subset is used as a validation set (see below for a demonstrating image).




In the image, each line corresponds to an epoch of training, the blue fold is used as the validation set, and the rest of the folds are used as the training set.

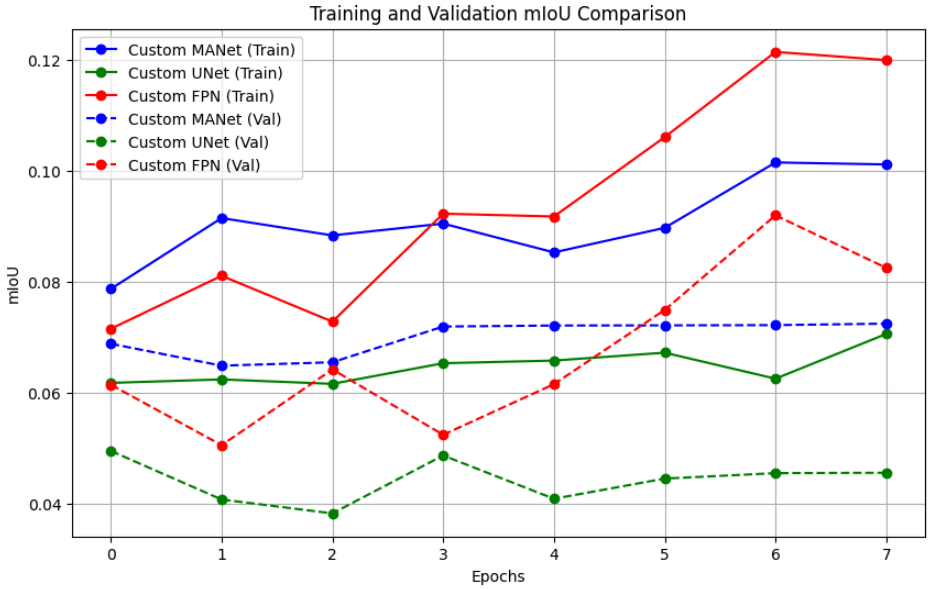
**Results**

We trained and tested Unet, MAnet and FPN with the same hyperparameters and optimizers. The following graph compare training and validation accuracy




By looking at the **accuracy** evaluation metric, we can say that the best model is the FPN net. We can also say that training and validation values are comparable, so there is no overfitting. The magnitude isn't so high due to the limited epochs, an improvement could be to train the models on more epoch.

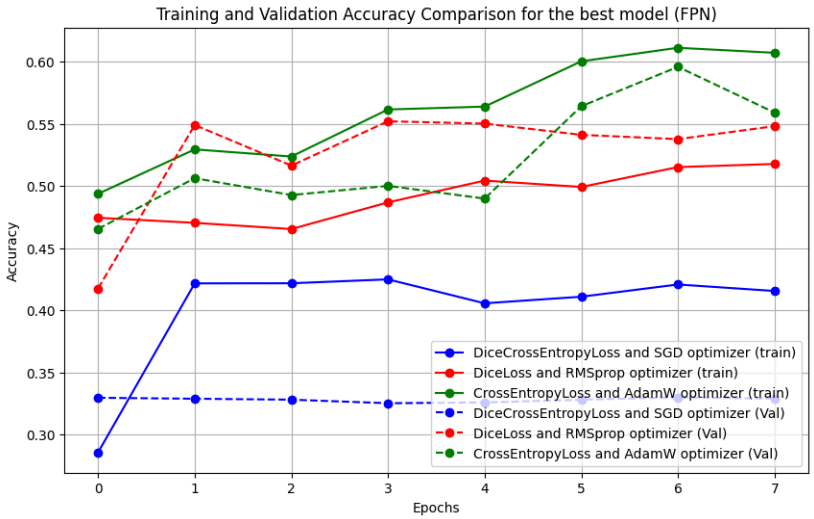
As an example, we present also the plot of the Intersection over Union (miou), also here the best model is the FPN net:



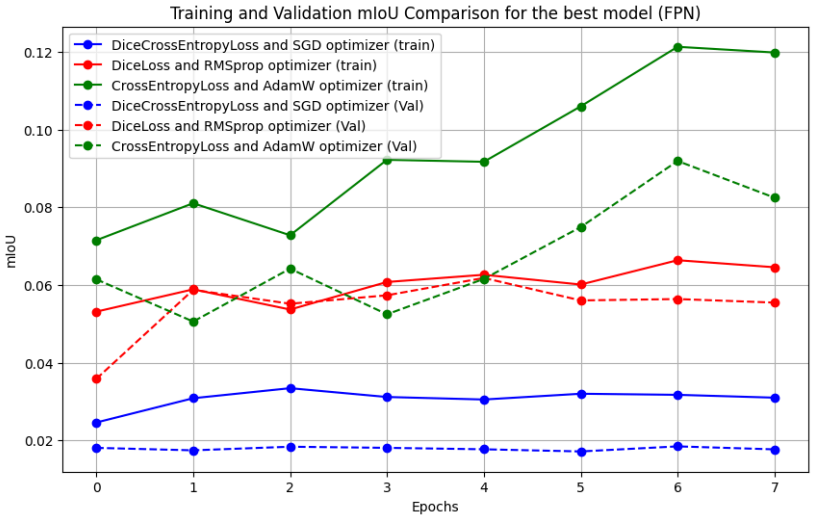

After define which is the best model, we changed the Optimizer and the loss function with this combination:

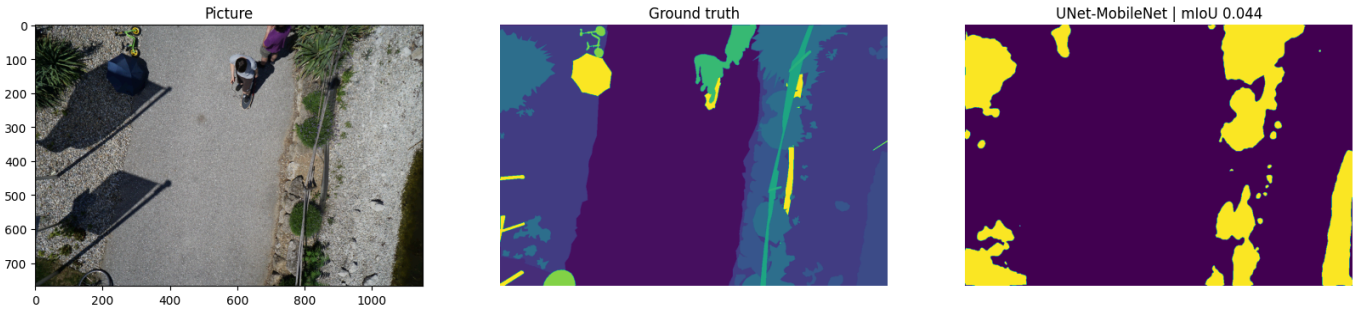
- **DiceLoss()** and **RMSprop()** optimizer (the loss It is based on the Dice coefficient which measures the similarity between two sets)
- **DiceCrossEntropyLoss()** and **SGD()** optimizer (is composite loss function that combines the properties of Cross-Entropy Loss and Dice Loss. It aims to leverage the strengths of both loss functions for tasks like semantic segmentation)

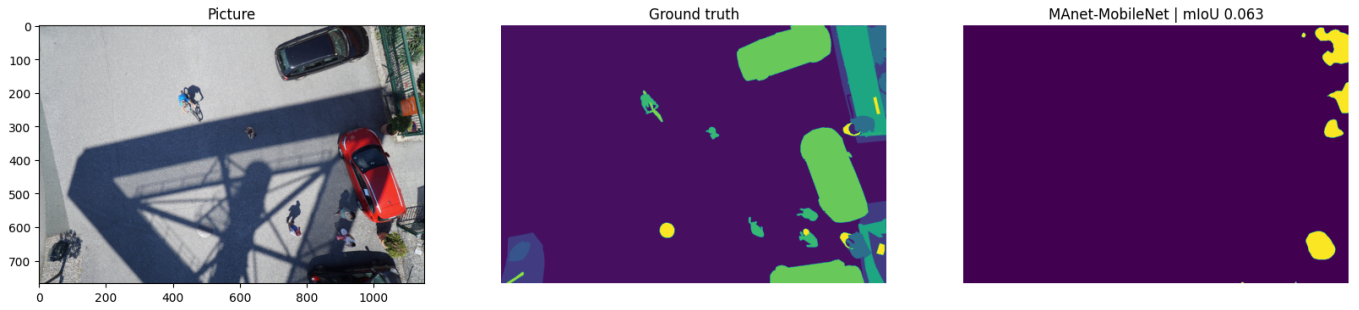
By looking at the evaluation metrics we can look at these accuracy trend for the following combination of optimizers and loss functions.




As we saw, the best accuracy is achieved with the model which is trained with the previous loss function and optimizer (CrossEntropyLoss and AdamW). The same considerations can be done with mIoU metric:




In these examples we can see that the accuracy is not so high (as we said before we didn't train the network with a high number of epochs due to resource limitation). However all the objects belonging to the tree class are classified correctly. It seems to be a case of **underfitting**.

**Completed items**

| | |
|---|---|
| **Semantic segmentation** | **1** |
| **Additional loss function for multiclass** | **1** |
| **pretrained model on different problem (transfer learning)** | **1** |
| **different model with different architectures** | **1** |
| **your own part of the dataset (>500)** | **1** |
| **architecture tuning** | **1** |
| **Data augmentation** | **1** |
| **Cross Validation** | **1** |
| **testing a few optimizers** | **1** |
| **testing a few loss functions** | **1** |