UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

Presentazione Progetto di

Image Processing for Computer Vision
Semantic Segmentation - Cityscapes

Prof

Giuseppe Scarpa
Matteo Ciotola

Candidati

Vincenzo D'Angelo M63/1595
Giorgio Di Costanzo M63/1579

Sommario

- Dataset
- Loss Functions and Evaluation Metrics
- Training U-Net
- Training ResNet
- Training with ASPP
- Extra: Transfer Learning

Sommario

- **Dataset**
- Loss Functions and Evaluation Metrics
- Training U-Net
- Training ResNet
- Training with ASPP
- Extra: Transfer Learning

Dataset

CITYSCAPES

Cityscapes Dataset – Semantic Understanding of Urban Street Scenes

Dataset

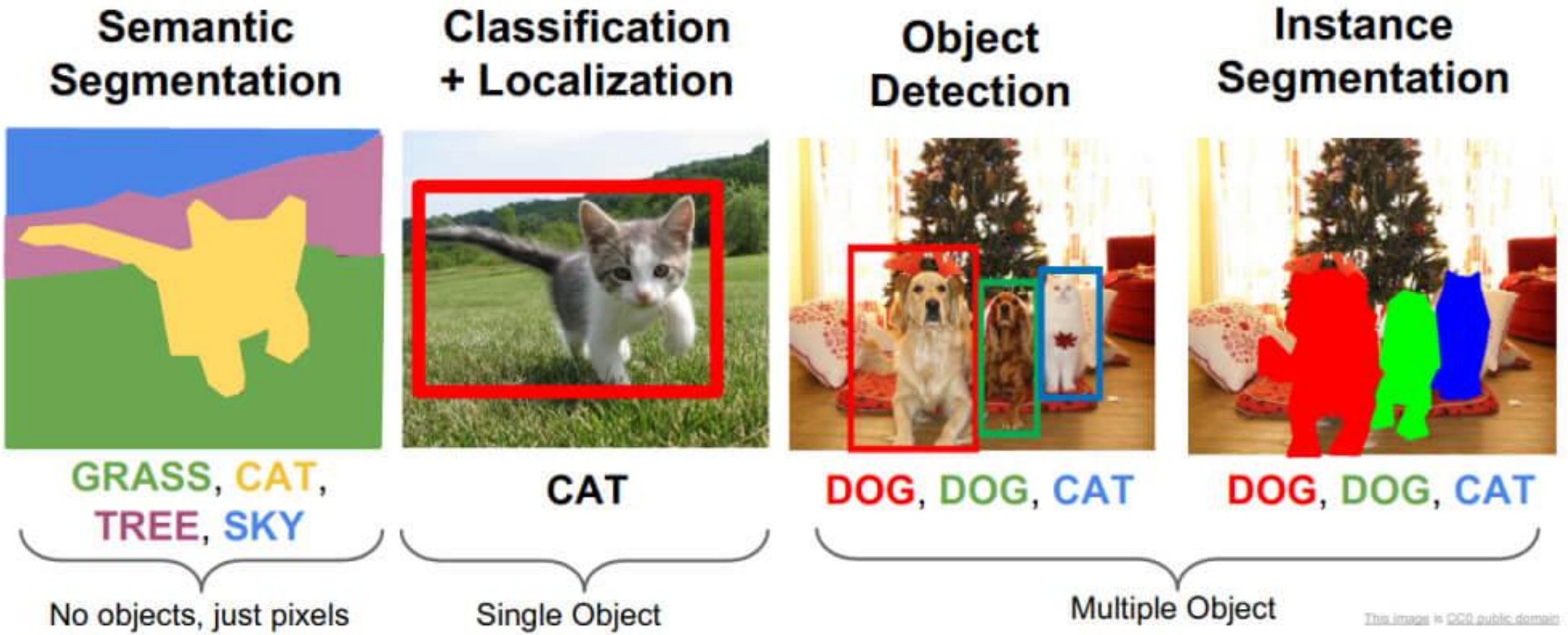
Large-scale dataset:

- Stereo video sequences recorded in street scenes from 50 different cities in Germany
- Represents typical urban scenes with varying levels of traffic, pedestrians, buildings, roads, and other urban elements.
- high quality pixel-level annotations of 5 000 frames
- 20 000 weakly annotated frames

Supported Tasks:

- **Semantic segmentation**
- Instance segmentation
- Panoptic segmentation

Dataset


Semantic Segmentation
Classification + Localization
Object Detection
Instance Segmentation
GRASS, CAT, TREE, SKY
CAT
DOG, DOG, CAT
DOG, DOG, CAT
No objects, just pixels
Single Object
Multiple Object

Dataset

Group	Classes
flat	road · sidewalk · parking+ · rail track+
human	person* · rider*
vehicle	car* · truck* · bus* · on rails* · motorcycle* · bicycle* · caravan*+ · trailer*+
construction	building · wall · fence · guard rail+ · bridge+ · tunnel+
object	pole · pole group+ · traffic sign · traffic light
nature	vegetation · terrain
sky	sky
void	ground+ · dynamic+ · static+

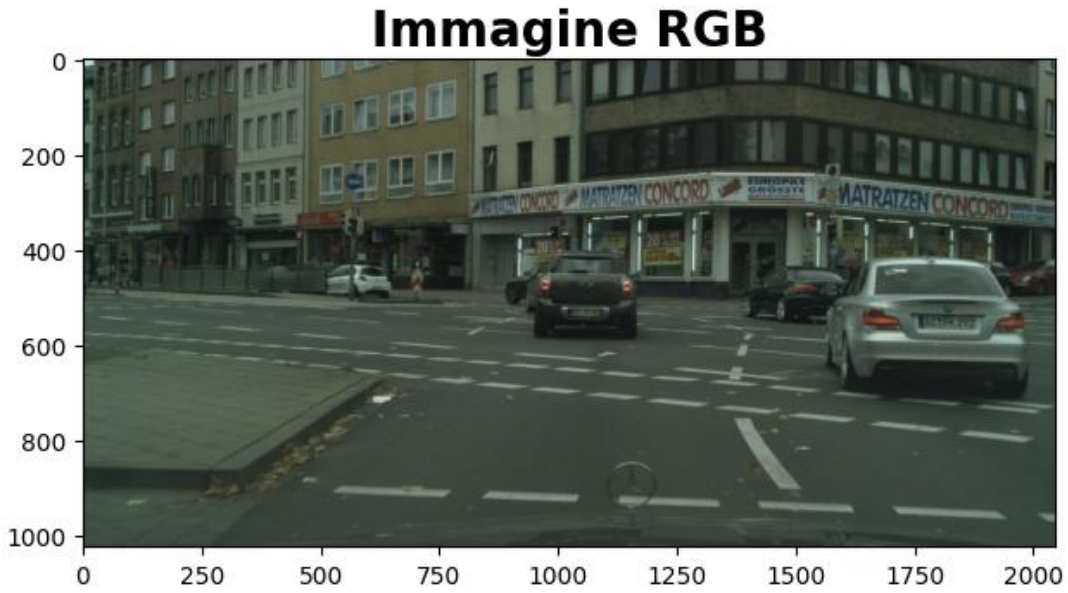
* Single instance annotations are available. However, if the boundary between such instances cannot be clearly seen, the whole crowd/group is labeled together and annotated as group, e.g. car group.
\+ This label is not included in any evaluation and treated as void (or in the case of *license plate* as the vehicle mounted on).

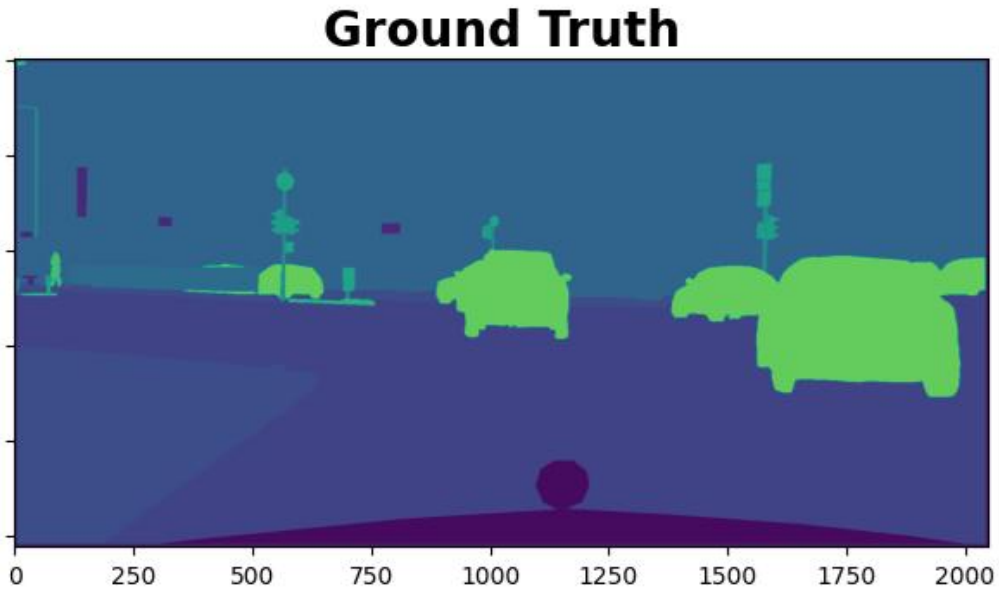
cityscapesScripts/cityscapesscripts/helpers/labels.py at master · mcordts/cityscapesScripts · GitHub

Dataset

```
labels = [
    #       name                     id    trainId   category            catId     hasInstances   ignoreInEval   color
    Label(  'unlabeled'            ,  0 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'ego vehicle'          ,  1 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'rectification border' ,  2 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'out of roi'           ,  3 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'static'               ,  4 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'dynamic'              ,  5 ,      255 , 'void'            , 0       , False        , True         , (111, 74,  0) ),
    Label(  'ground'               ,  6 ,      255 , 'void'            , 0       , False        , True         , ( 81,  0, 81) ),
    Label(  'road'                 ,  7 ,        0 , 'flat'            , 1       , False        , False        , (128, 64,128) ),
    Label(  'sidewalk'             ,  8 ,        1 , 'flat'            , 1       , False        , False        , (244, 35,232) ),
    Label(  'parking'              ,  9 ,      255 , 'flat'            , 1       , False        , True         , (250,170,160) ),
    Label(  'rail track'           , 10 ,      255 , 'flat'            , 1       , False        , True         , (230,150,140) ),
    Label(  'building'             , 11 ,        2 , 'construction'    , 2       , False        , False        , ( 70, 70, 70) ),
    Label(  'wall'                 , 12 ,        3 , 'construction'    , 2       , False        , False        , (102,102,156) ),
    Label(  'fence'                , 13 ,        4 , 'construction'    , 2       , False        , False        , (190,153,153) ),
    Label(  'guard rail'           , 14 ,      255 , 'construction'    , 2       , False        , True         , (180,165,180) ),
    Label(  'bridge'               , 15 ,      255 , 'construction'    , 2       , False        , True         , (150,100,100) ),
    Label(  'tunnel'               , 16 ,      255 , 'construction'    , 2       , False        , True         , (150,120, 90) ),
    Label(  'pole'                 , 17 ,        5 , 'object'          , 3       , False        , False        , (153,153,153) ),
    Label(  'polegroup'            , 18 ,      255 , 'object'          , 3       , False        , True         , (153,153,153) ),
    Label(  'traffic light'        , 19 ,        6 , 'object'          , 3       , False        , False        , (250,170, 30) ),
    Label(  'traffic sign'         , 20 ,        7 , 'object'          , 3       , False        , False        , (220,220,  0) ),
    Label(  'vegetation'           , 21 ,        8 , 'nature'          , 4       , False        , False        , (107,142, 35) ),
    Label(  'terrain'              , 22 ,        9 , 'nature'          , 4       , False        , False        , (152,251,152) ),
    Label(  'sky'                  , 23 ,       10 , 'sky'             , 5       , False        , False        , ( 70,130,180) ),
    Label(  'person'               , 24 ,       11 , 'human'           , 6       , True         , False        , (220, 20, 60) ),
    Label(  'rider'                , 25 ,       12 , 'human'           , 6       , True         , False        , (255,  0,  0) ),
    Label(  'car'                  , 26 ,       13 , 'vehicle'         , 7       , True         , False        , (  0,  0,142) ),
    Label(  'truck'                , 27 ,       14 , 'vehicle'         , 7       , True         , False        , (  0,  0, 70) ),
    Label(  'bus'                  , 28 ,       15 , 'vehicle'         , 7       , True         , False        , (  0, 60,100) ),
    Label(  'caravan'              , 29 ,      255 , 'vehicle'         , 7       , True         , True         , (  0,  0, 90) ),
    Label(  'trailer'              , 30 ,      255 , 'vehicle'         , 7       , True         , True         , (  0,  0,110) ),
    Label(  'train'                , 31 ,       16 , 'vehicle'         , 7       , True         , False        , (  0, 80,100) ),
    Label(  'motorcycle'           , 32 ,       17 , 'vehicle'         , 7       , True         , False        , (  0,  0,230) ),
    Label(  'bicycle'              , 33 ,       18 , 'vehicle'         , 7       , True         , False        , (119, 11, 32) ),
    Label(  'license plate'        , -1 ,       -1 , 'vehicle'         , 7       , False        , True         , (  0,  0,142) ),
]
```

Dataset





- Original size: 1024x2048 pixels
- Format: .png

Dataset

TRAIN SET
2472 images (~70%)

VALIDATION SET
503 images (~15%)

TEST SET
500 images (~15%)

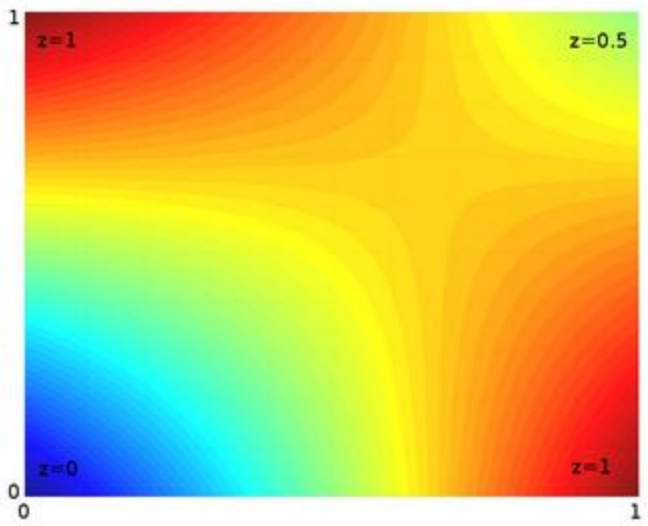
Dataset

```
T.Normalize(mean=(0.28881703, 0.32694227, 0.28622528), std=(0.18679116, 0.18973092, 0.18646142))
```

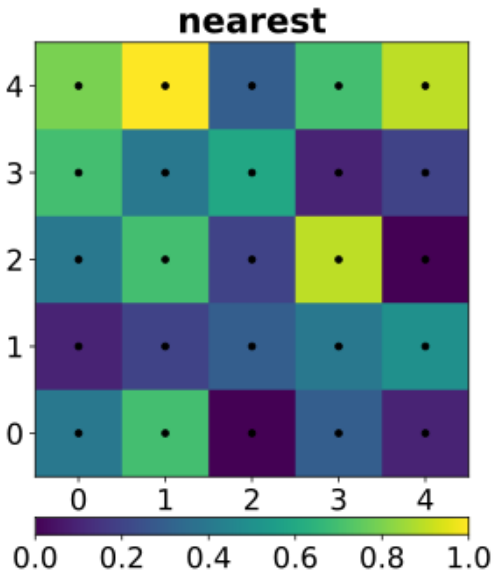
Dataset

```
image = T.functional.resize(image, (256, 512), interpolation=InterpolationMode.BILINEAR)
gt = T.functional.resize(gt, (256, 512), interpolation=InterpolationMode.NEAREST_EXACT)
```

Bilinear: Uses all nearby pixels to calculate the pixel's value, using linear interpolations.



Nearest-Neighbor: Copies the value from the nearest pixel.



Sommario

- Dataset
- **Loss Functions and Evaluation Metrics**
- Training U-Net
- Training ResNet
- Training with ASPP
- Extra: Transfer Learning

Loss Functions and Evaluation Metrics

CrossEntropyLoss

`torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean', label_smoothing=0.0)`

- first loss: ~ - log(1/N) ~ log(N)

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore\_index}\}$$

- **Pixel-Wise Precision**
- **Effective for Well-Balanced Classes**
- **Mathematically Simple and Stable**

- **Sensitive to Class Imbalance**
- **No Emphasis on Region-Level Overlap**
- **May Not Capture Global Structure**

CrossEntropyLoss — PyTorch 2.5 documentation

Loss Functions and Evaluation Metrics

Focal Loss

$$\mathrm{FL}(p_\mathrm{t}) = -\alpha_\mathrm{t}(1-p_\mathrm{t})^\gamma \log(p_\mathrm{t})$$

$$p_\mathrm{t} = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise,} \end{cases}$$

$$\mathrm{CE}(p_\mathrm{t}) = -\alpha_\mathrm{t} \log(p_\mathrm{t})$$

- p_t large (correct prediction) $\rightarrow (1-p_t)^\gamma$ small. Penalty term of the Focal Loss reduced
- p_t small (incorrect prediction) $\rightarrow (1-p_t)^\gamma$ large. Penalty term of the Focal Loss increased
- first loss $(1-p_t)^\gamma \approx 1 \rightarrow \mathrm{Focal}_{Loss} \approx \mathrm{CE}_{Loss}$
- if $\gamma > 0$, the Focal Loss becomes equivalent to the classic Cross-Entropy Loss.
- if $\gamma > 0$, the Focal Loss will penalize hard examples more significantly.

[1708.02002] Focal Loss for Dense Object Detection

Loss Functions and Evaluation Metrics

Generalized Dice Loss (GDL)

$$\mathrm{GDL} = 1 - 2\frac{\sum_{l=1}^{2} w_l \sum_n r_{ln} p_{ln}}{\sum_{l=1}^{2} w_l \sum_n r_{ln} + p_{ln}}$$

- R: reference foreground segmentation with voxel values r_n
- P: predicted probabilistic map for the foreground label over N image elements p_n,

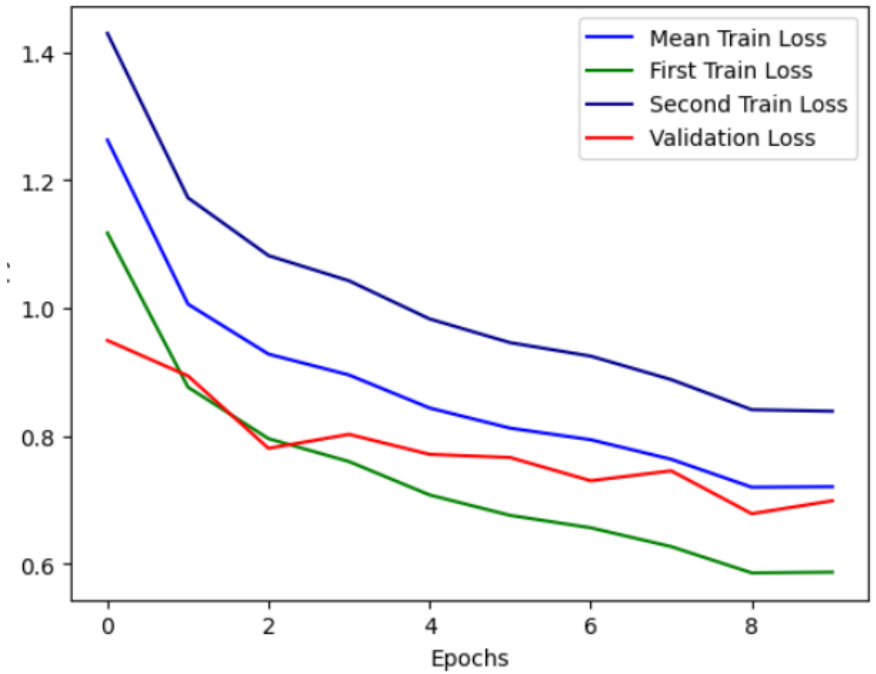
- **Handles Class Imbalance Effectively**
- **Direct Optimization for Overlap Metrics**
- **Focus on Region-Level Performance**
- **Gradient Instability for Small Regions**
- **Less Sensitive to Individual Pixel Errors**
- **Bias Toward Larger Classes**

[1707.03237] Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations

Loss Functions and Evaluation Metrics

Combined Loss

$$Loss = weight_1 Loss_1 + weight_2 Loss_2$$




```python
if loss1_old != np.inf and loss2_old != np.inf:

        delta1 = (loss1_old - loss1)/(loss1 + 1e-8)
        delta2 = (loss2_old - loss2)/(loss2 + 1e-8)

        if delta1 < 0 or delta2 < 0:
            delta1 = abs(delta1)
            delta2 = abs(delta2)
        else:
            delta1 = 1/delta1
            delta2 = 1/delta2
        delta1, delta2 = delta1/(delta1 + delta2), delta2/(delta1 + delta2)
        delta1 = round(delta1, 3)
        delta2 = round(delta2, 3)

        weight1 = (1 - alpha)*weight1 + alpha*delta1
        weight2 = (1 - alpha)*weight2 + alpha*delta2
        weight1 = round(weight1, 3)
        weight2 = round(weight2, 3)
        weigh1, weight2 = weight1/(weight1 + weight2), weight2/(weight1 + weight2)

else:
    weight1 = round((1/loss1),3)
    weight2 = round((1/loss2),3)
    weight1, weight2 = weight1/(weight1 + weight2), weight2/(weight1 + weight2)
```

This approach leverages the complementary characteristics of the two loss functions to address specific common issues in these tasks.

Loss Functions and Evaluation Metrics

Jaccard Index

`torchmetrics.JaccardIndex(num_classes, average='macro', ignore_index=None, absent_score=0.0, threshold=0.5, multilabel=False, **kwargs)`

$$JaccardIndex = \frac{|A \cap B|}{|A \cup B|}$$

- **average**
  Defines the reduction that is applied. Should be one of the following:
  - `'macro'` [default]: Calculate the metric for each class separately, and average the metrics across classes (with equal weights for each class).
  - `'micro'`: Calculate the metric globally, across all samples and classes.
  - `'weighted'`: Calculate the metric for each class separately, and average the metrics across classes, weighting each class by its support (`tp + fn`).
  - `'none'` or `None`: Calculate the metric for each class separately, and return the metric for every class. Note that if a given class doesn't occur in the preds or target, the value for the class will be `nan`.

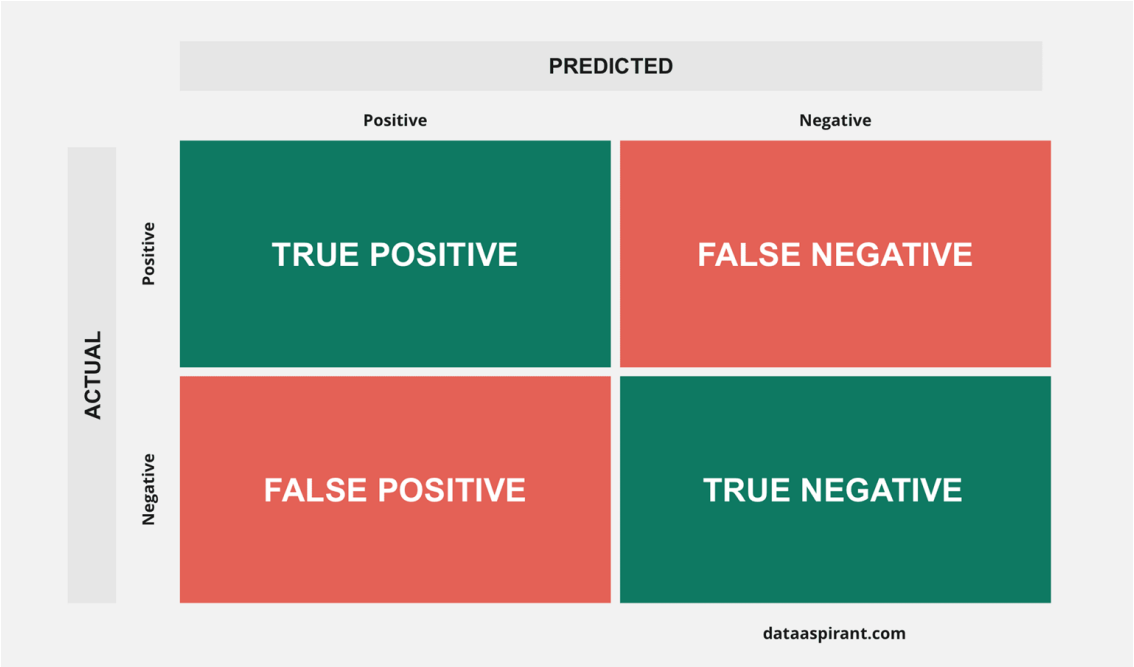
Jaccard Index — PyTorch-Metrics 0.10.2 documentation

Loss Functions and Evaluation Metrics

Confusion Matrix

`torchmetrics.ConfusionMatrix(num_classes, normalize=None, threshold=0.5, multilabel=False, compute_on_step=None, **kwargs)`

- **normalize**
  Normalization mode for confusion matrix. Choose from:
  - `None` or `'none'`: no normalization (default)
  - `'true'`: normalization over the targets (most commonly used)
  - `'pred'`: normalization over the predictions
  - `'all'`: normalization over the whole matrix



confusion_matrix — scikit-learn 1.6.0 documentation

Loss Functions and Evaluation Metrics

- Precision = $\frac{\text{True Positive (TP)}}{\text{True Positive (TP)+False Positive (FP)}}$
  - Measures the accuracy of positive predictions.
  - High precision indicates that the model rarely misclassifies negatives as positives.

- Recall (Sensitivity) = $\frac{\text{True Positive (TP)}}{\text{True Positive (TP)+False Negative (FN)}}$
  - Measures the ability to capture all relevant positive cases.
  - High recall means the model identifies most of the positives.

- F1 Score = $\frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision+Recall}}$
  - Combines precision and recall into a single metric (harmonic mean).
  - Useful when there's a tradeoff between precision and recall.

confusion_matrix — scikit-learn 1.6.0 documentation

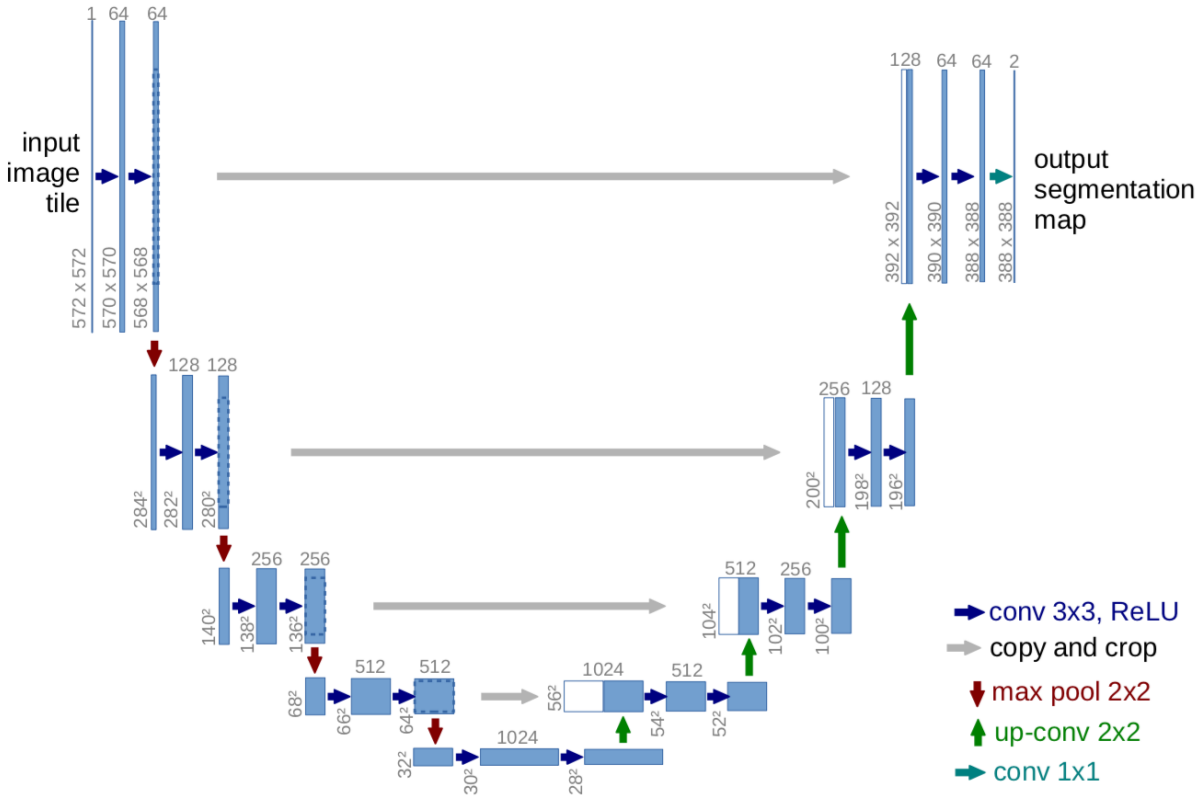
Sommario

- Dataset
- Loss Functions and Evaluation Metrics
- **Training U-Net**
- Training ResNet
- Training with ASPP
- Extra: Transfer Learning

Training U-Net

U-Net

- **U-Shape Architecture (Encoder-Decoder)**
  - **Encoder (Contracting)**: progressively reduce the spatial resolution of the image while increasing the number of channels (feature maps).
  - **Decoder (Expanding)**: reconstruct the original spatial resolution while reducing the number of channels.
- **Skip Connections**
  - data from the encoder is directly concatenated with the data in the decoder. This helps the decoder recover spatial details from the early stages and improves segmentation accuracy.
  - Skip connections are **concatenations** (not summations), meaning information from each encoder level is passed directly to the corresponding decoder level.



U-Net: Convolutional Networks for Biomedical Image Segmentation

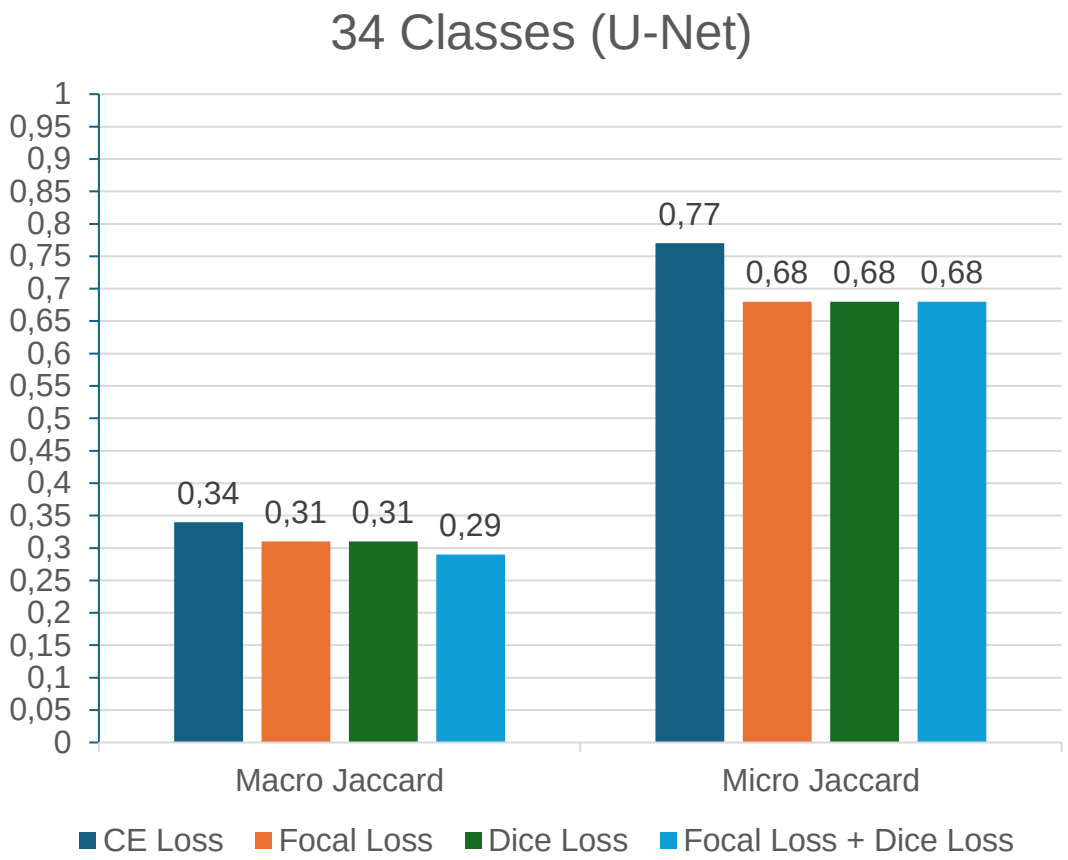
Training U-Net

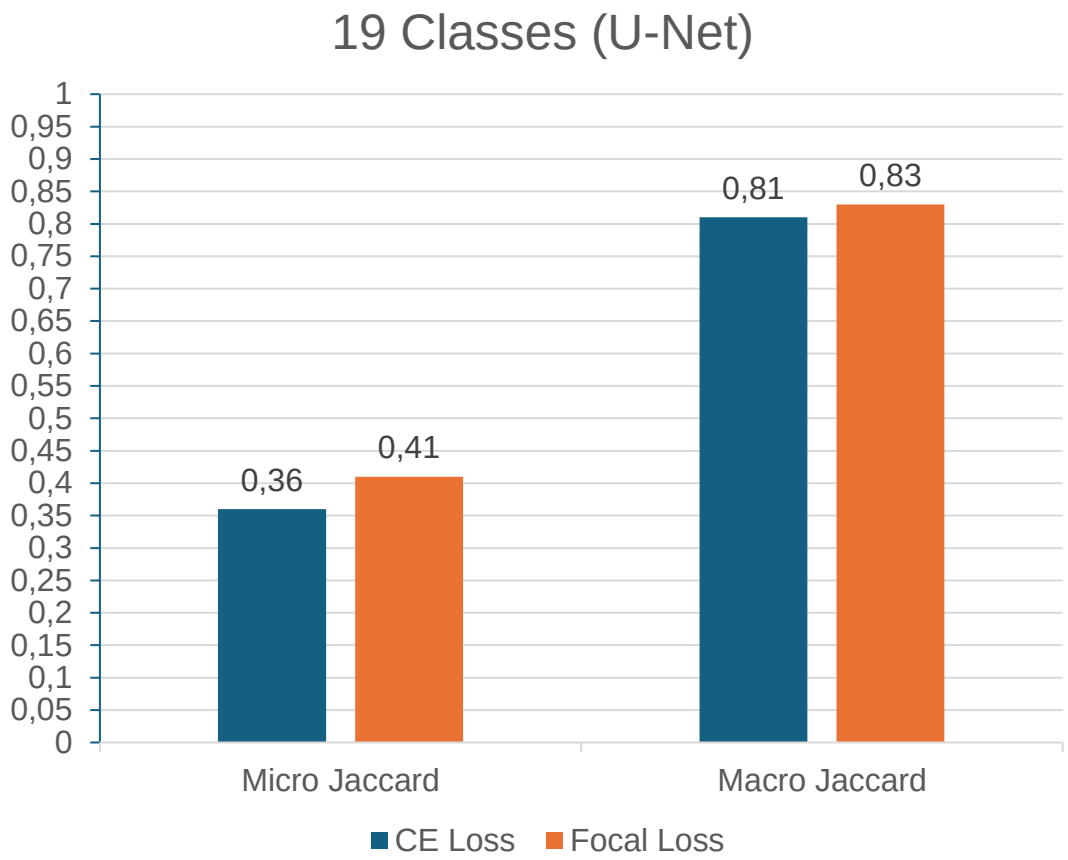
```
class UNet(nn.Module):

    def __init__(self, num_classes):
        super(UNet, self).__init__()
        self.num_classes = num_classes
        self.contracting_11 = self.conv_block(in_channels=3, out_channels=64)
        self.contracting_12 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.contracting_21 = self.conv_block(in_channels=64, out_channels=128)
        self.contracting_22 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.contracting_31 = self.conv_block(in_channels=128, out_channels=256)
        self.contracting_32 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.contracting_41 = self.conv_block(in_channels=256, out_channels=512)
        self.contracting_42 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.middle = self.conv_block(in_channels=512, out_channels=1024)
        self.expansive_11 = nn.ConvTranspose2d(in_channels=1024, out_channels=512, kernel_size=3, stride=2, padding=1, output_padding=1)
        self.expansive_12 = self.conv_block(in_channels=1024, out_channels=512)
        self.expansive_21 = nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=3, stride=2, padding=1, output_padding=1)
        self.expansive_22 = self.conv_block(in_channels=512, out_channels=256)
        self.expansive_31 = nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=3, stride=2, padding=1, output_padding=1)
        self.expansive_32 = self.conv_block(in_channels=256, out_channels=128)
        self.expansive_41 = nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=3, stride=2, padding=1, output_padding=1)
        self.expansive_42 = self.conv_block(in_channels=128, out_channels=64)
        self.output = nn.Conv2d(in_channels=64, out_channels=num_classes, kernel_size=3, stride=1, padding=1)

    def conv_block(self, in_channels, out_channels):
        block = nn.Sequential(nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=3, stride=1, padding=1),
                              nn.ReLU(),
                              nn.BatchNorm2d(num_features=out_channels),
                              nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=3, stride=1, padding=1),
                              nn.ReLU(),
                              nn.BatchNorm2d(num_features=out_channels))
        return block
```

$$O = \frac{\lfloor I + 2P - K \rfloor}{S} + 1$$

$$O = S \cdot (I - 1) - 2P + K + P_{\text{out}}$$

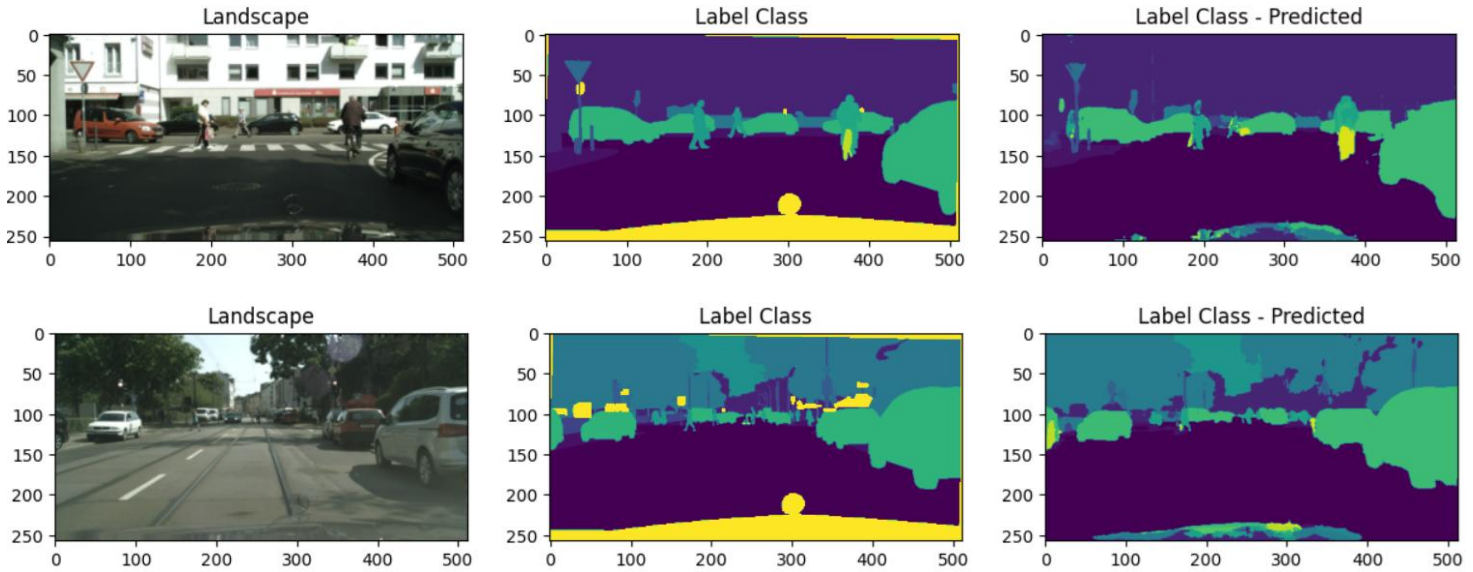
Training U-Net



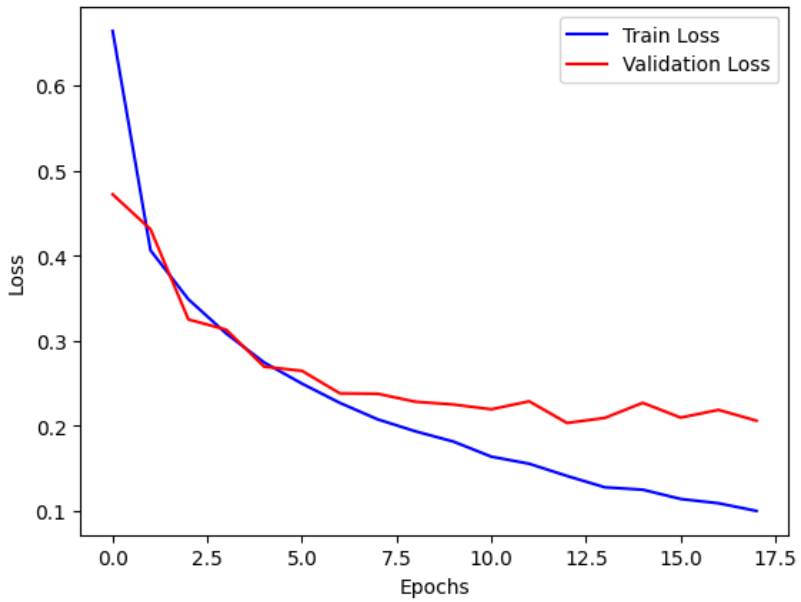


Training U-Net - 19 Classes - Focal Loss

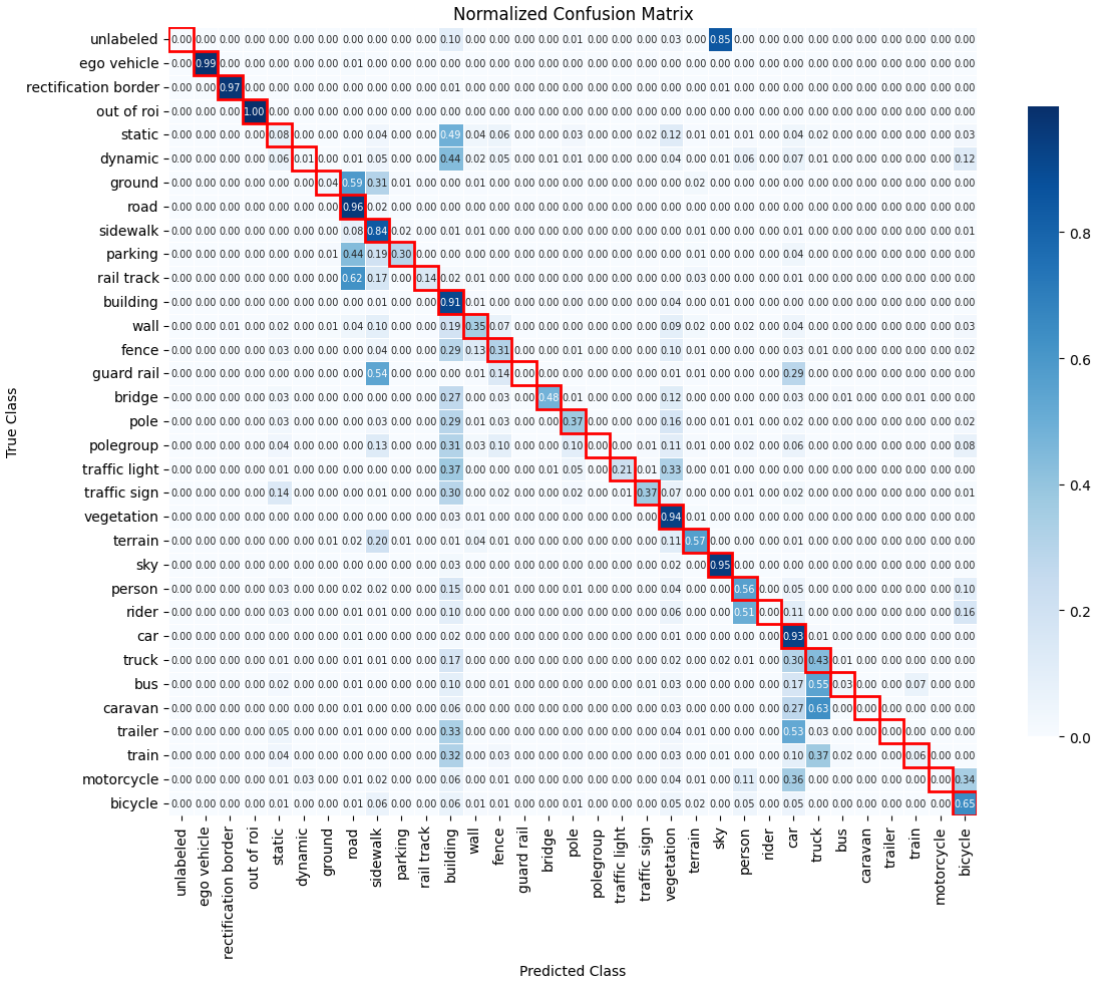
- size = 256x512
- epochs = 20
- criterion = FocalLoss(ignore_index = 255)
- optimizer = torch.optim.Adam
  (model.parameters(), lr=0.00005)

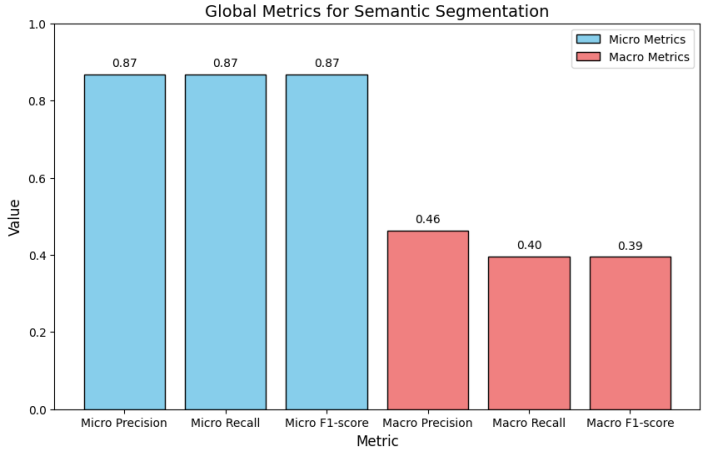


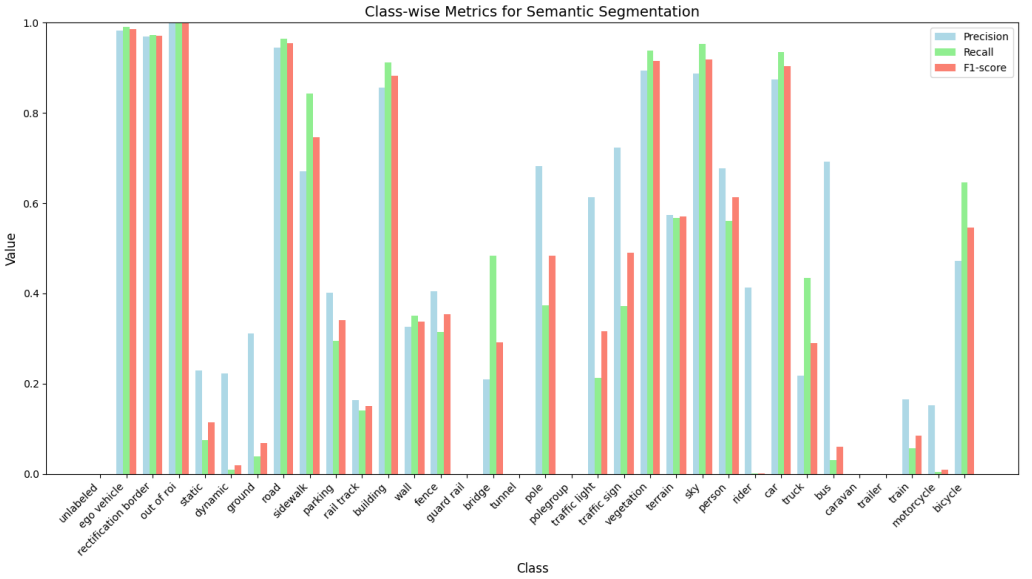

```
Training: 100%|██████████| 618/618 [07:16<00:00,  1.41it/s]
Validation: 100%|██████████| 126/126 [01:04<00:00,  1.95it/s]
Epoch:20/20  Time: 8.36m
Total time: 126.10 m
```



Training U-Net - 34 Classes - CE Loss

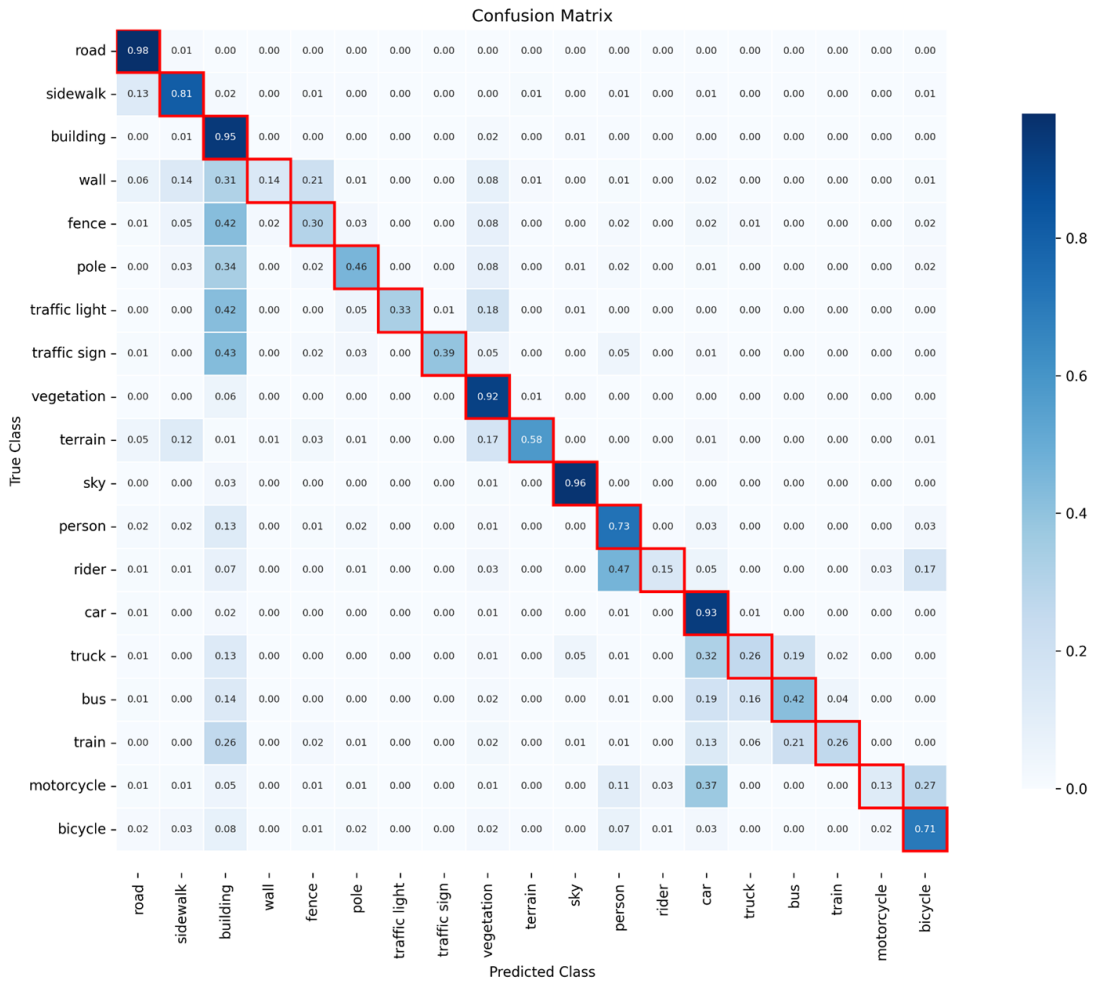


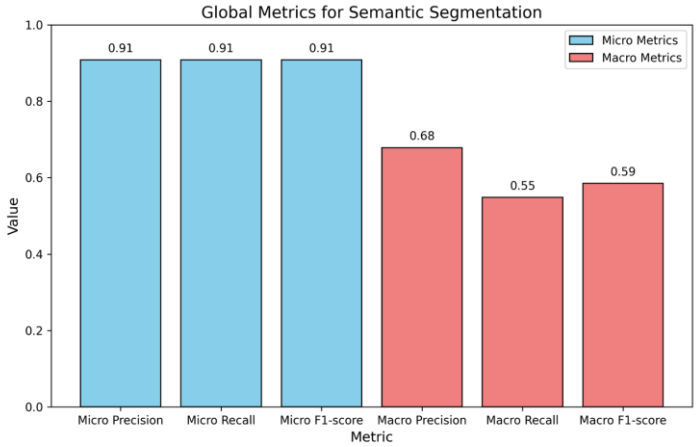


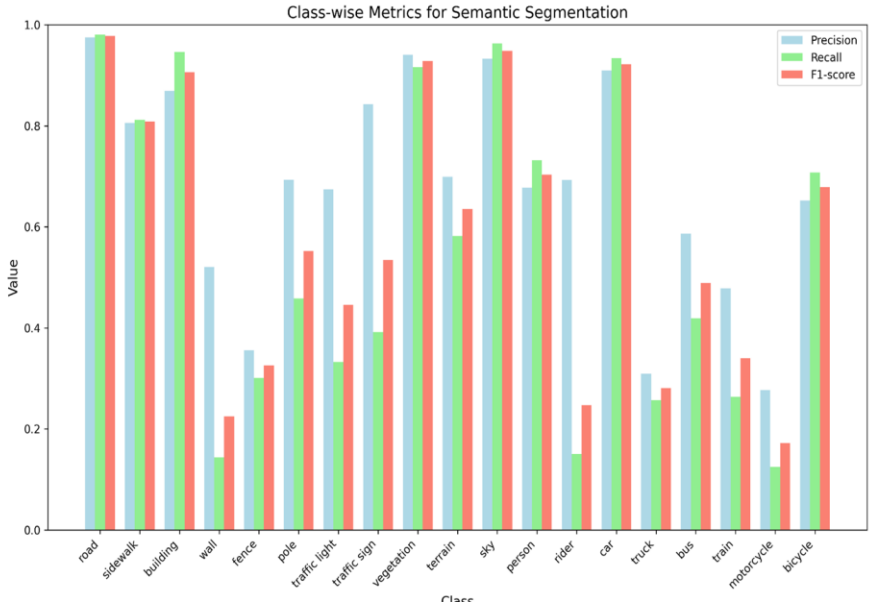


Jaccard macro: 0.34
Jaccard micro: 0.77

Training U-Net - 19 Classes - Focal Loss

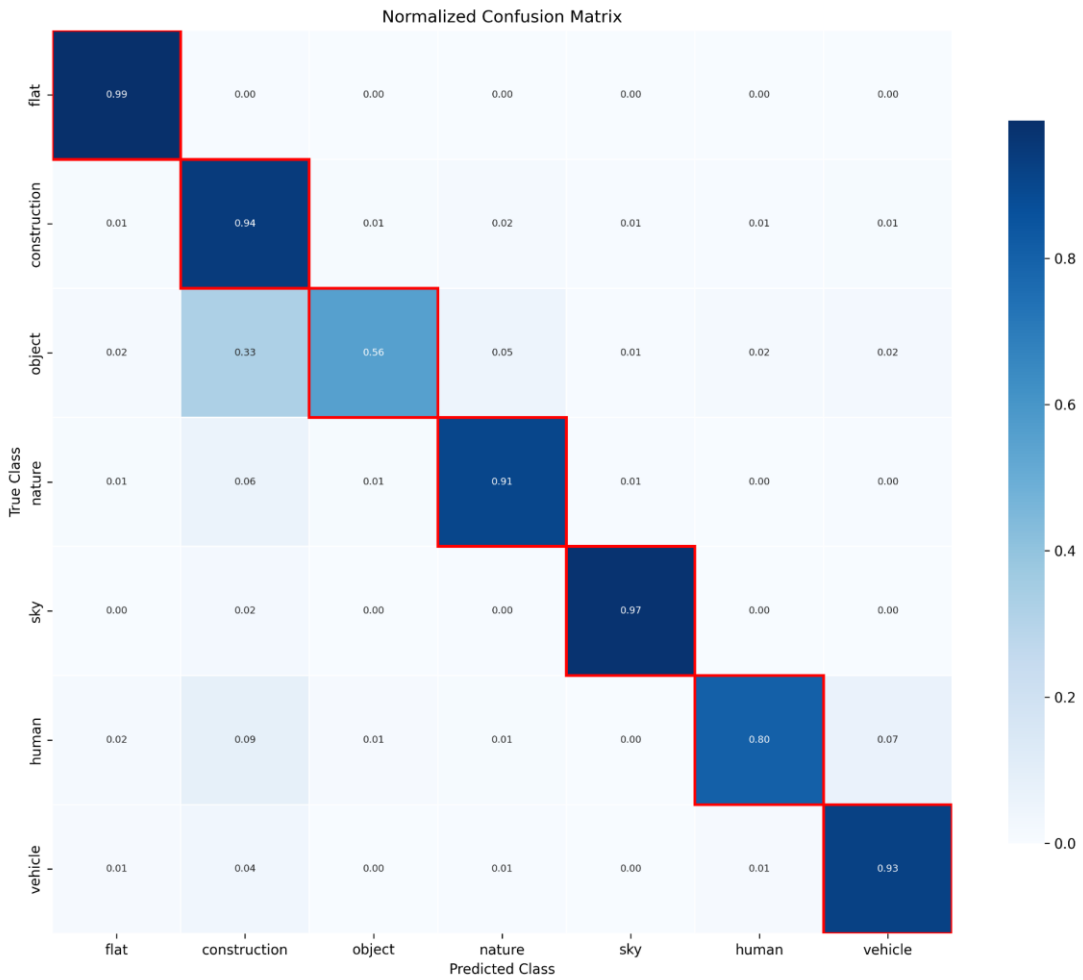


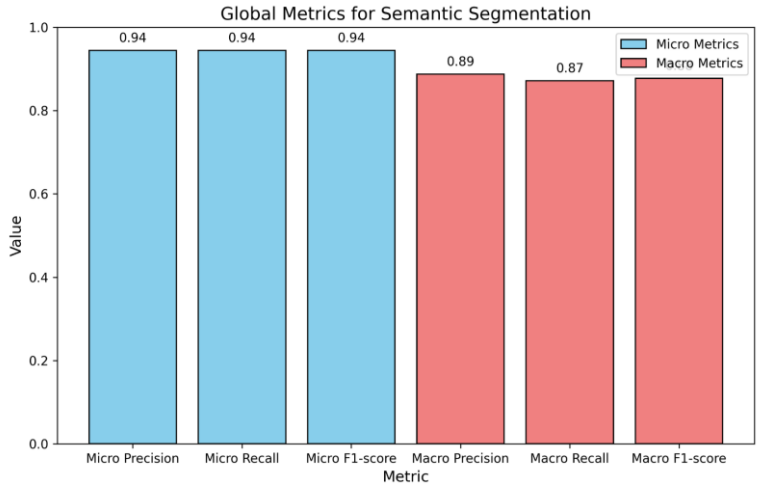


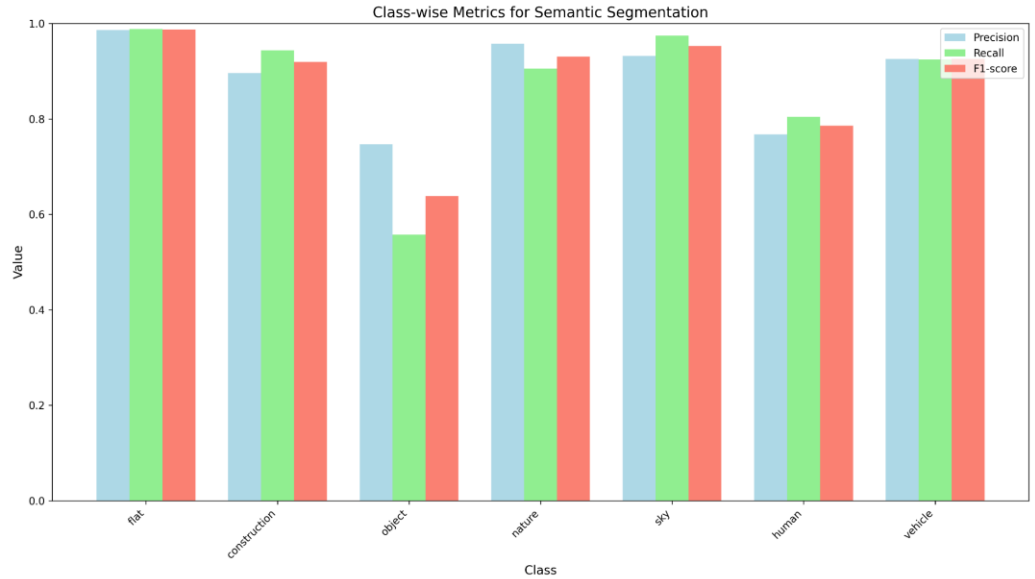


Jaccard macro: 0.41
Jaccard micro: 0.83

Training U-Net - 7 Categories - Focal Loss







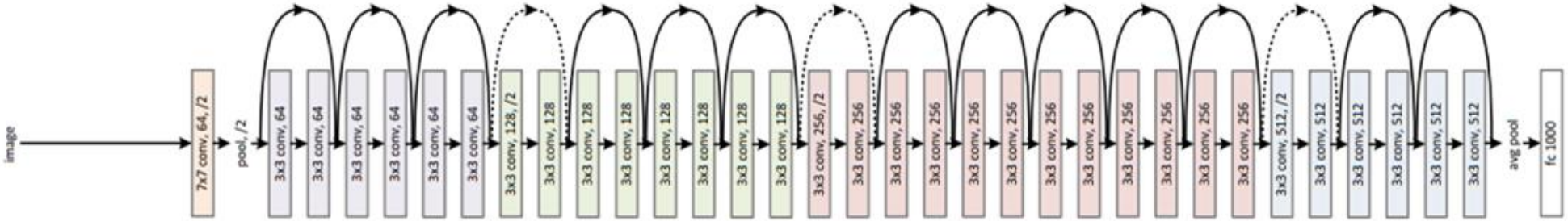
Jaccard macro: 0.77
Jaccard micro: 0.89

Sommario

- Dataset
- Loss Functions and Evaluation Metrics
- Training U-Net
- **Training ResNet**
- Training with ASPP
- Extra: Transfer Learning

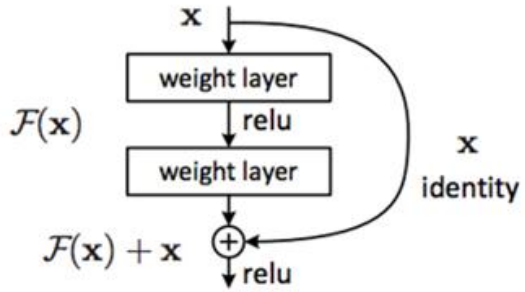
Training ResNet

ResNet



[1512.03385] Deep Residual Learning for Image Recognition

- **Residual Learning**
  - connections "skip" allowing direct information flow from one layer to another.
  - **residual block:**Instead of learning the output directly, the network learns the difference between the input and output, which facilitates training deep networks. $\text{Output} = F(x, \{W_i\}) + x$
  - ResNet is designed to be very deep, with versions (ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152 …) indicating the number of layers.
  - The use of residual connections allows very deep networks to avoid the **vanishing gradients** problem, which commonly occurs in deep networks without such connections.
  - Each residual block consists of a series of convolutions, typically with 3x3 kernels.



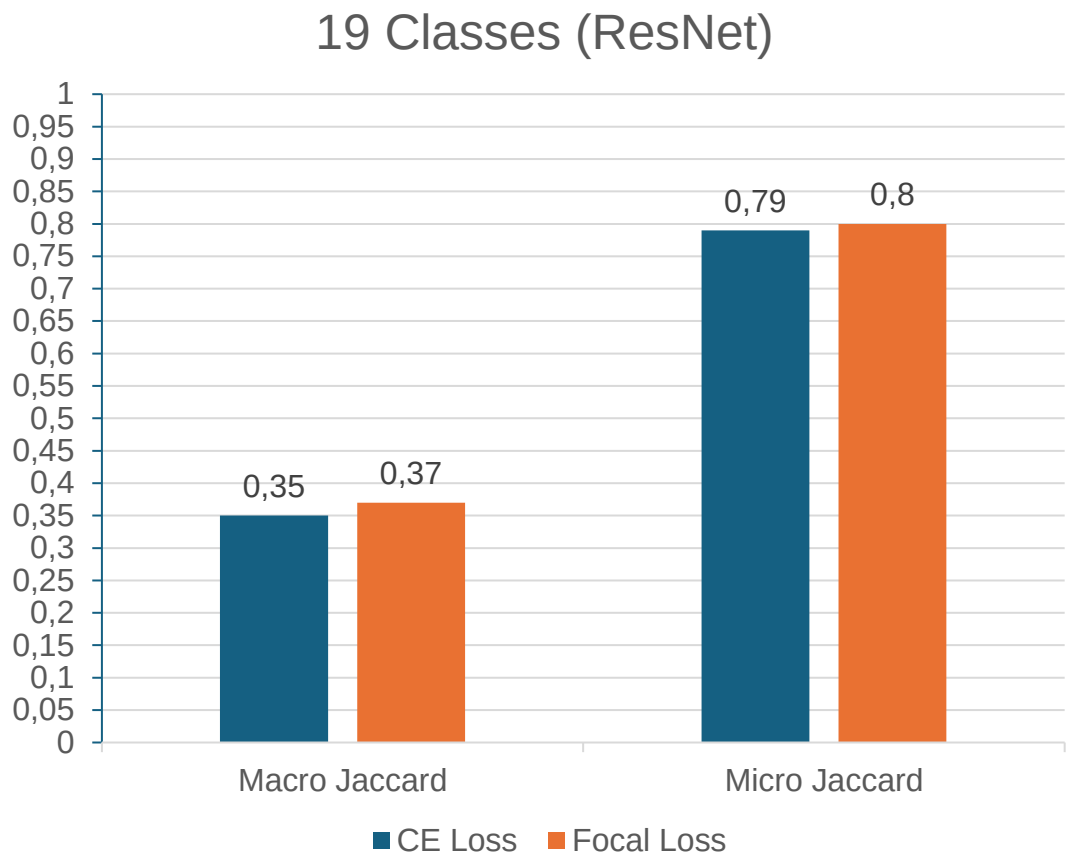
Training ResNet

```python
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=num_classes):
        super(ResNet, self).__init__()

        # Strato iniziale
        self.in_channels = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        # Strati successivi (4 gruppi di blocchi)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)

        # Aggiungi i decoder layers (convoluzioni trasposte o upsampling)
        self.upconv1 = self._decoder_block(512, 256)
        self.upconv2 = self._decoder_block(256, 128)
        self.upconv3 = self._decoder_block(128, 64)
        self.upconv4 = self._decoder_block(64, 64)
        self.upconv5 = self._decoder_block(64, 64)

        # Layer finale per la segmentazione
        self.final_conv = nn.Conv2d(64, num_classes, kernel_size=3, padding=1)

    def _make_layer(self, block, out_channels, num_blocks, stride):
        layers = []
        layers.append(block(self.in_channels, out_channels, stride))
        self.in_channels = out_channels
        for _ in range(1, num_blocks):
            layers.append(block(self.in_channels, out_channels))
        return nn.Sequential(*layers)

    def _decoder_block(self, in_channels, out_channels):
        # Decoder block per l'upsampling (transposed convolution)
        return nn.Sequential(
            nn.ConvTranspose2d(in_channels, out_channels, kernel_size=4, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.BatchNorm2d(out_channels)
        )
```

```python
# Definizione di un blocco ResNet (ResBlock)
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        # Skip connection
        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))   # Primo layer + ReLU
        out = self.bn2(self.conv2(out))         # Secondo layer
        out += self.shortcut(x)                 # Aggiunta della connessione residua
        out = F.relu(out)                       # ReLU finale
        return out
```

```python
def resnet18(num_classes=num_classes):
    """Constructs a ResNet-18 model."""
    return ResNet(BasicBlock, [2, 2, 2, 2], num_classes=num_classes)

def resnet34(num_classes=num_classes):
    """Constructs a ResNet-34 model."""
    return ResNet(BasicBlock, [3, 4, 6, 3], num_classes=num_classes)
```

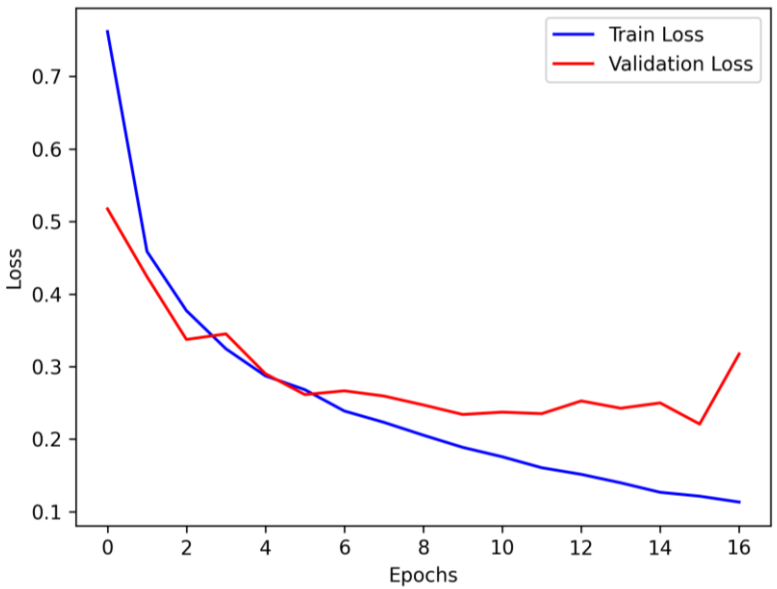
Training ResNet

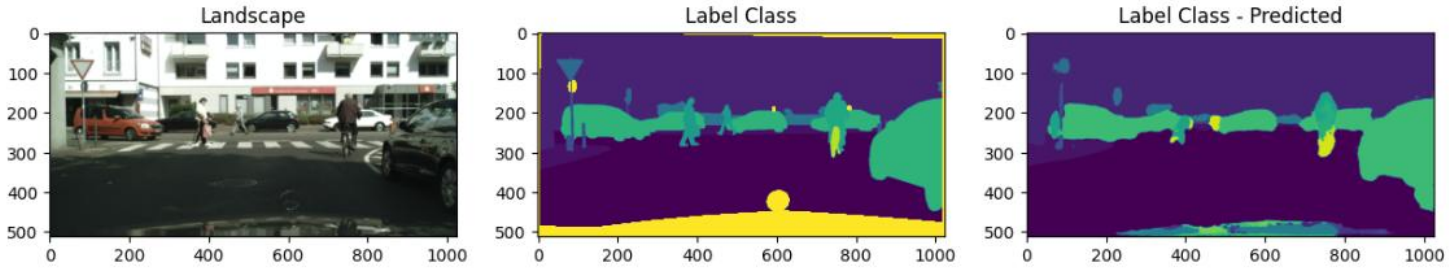


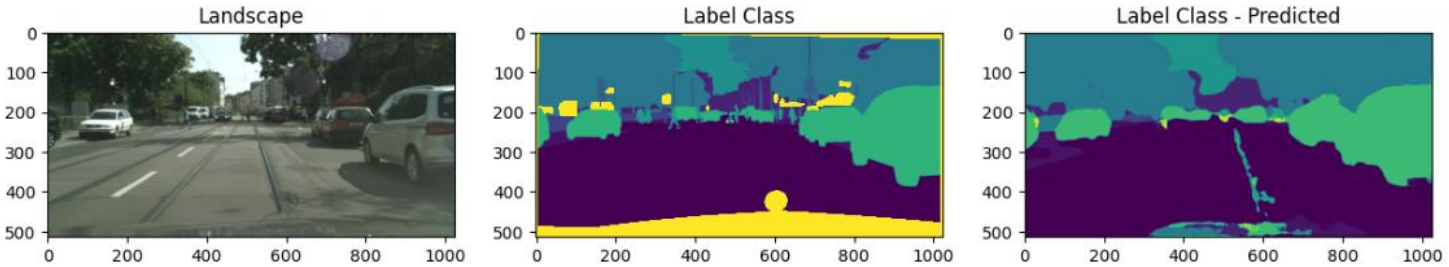
Training ResNet - 19 Classes - Focal Loss

- size = 512x1024
- epochs = 17
- criterion = FocalLoss(ignore_index = 255)
- optimizer = torch.optim.Adam
  (model.parameters(), lr=0.00005)

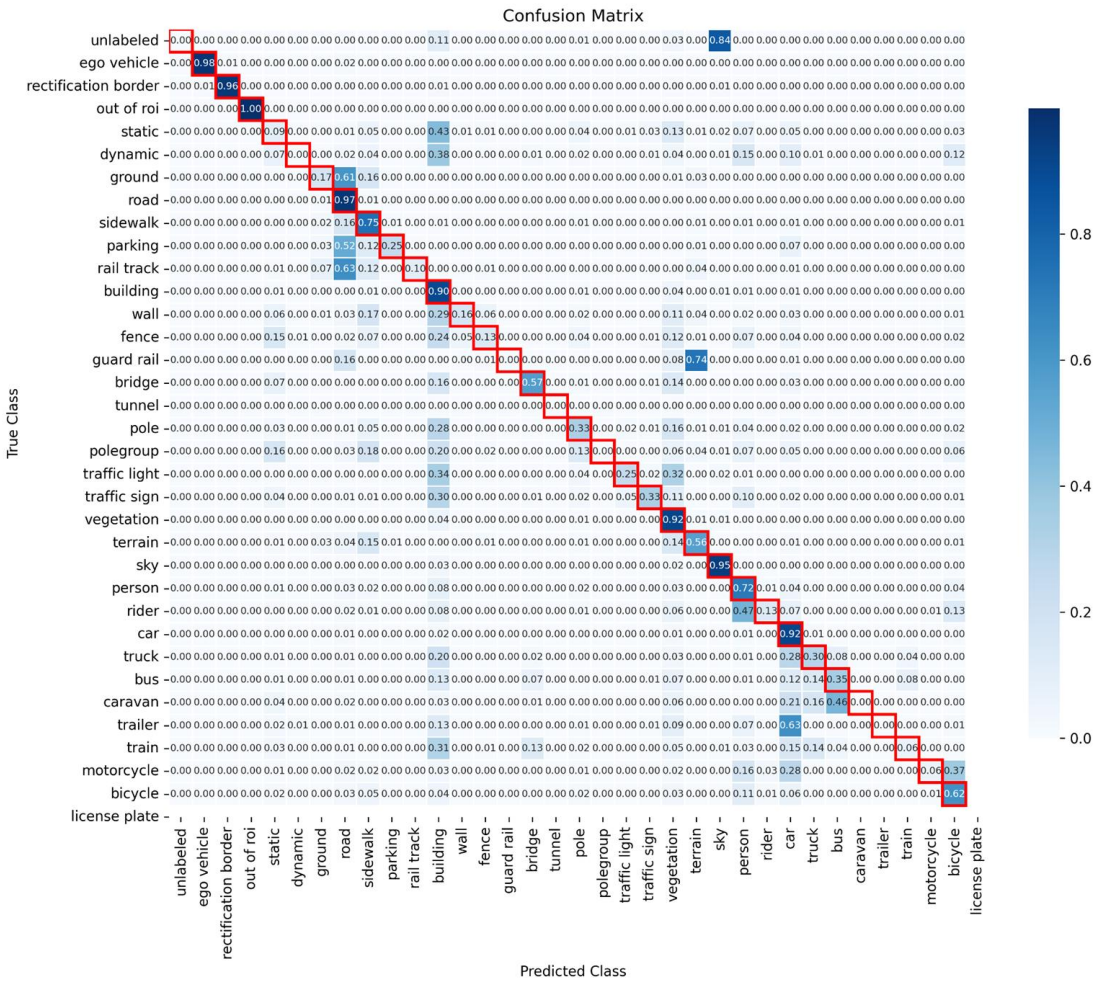
```
Training: 100%|██████████| 618/618 [08:19<00:00,  1.24it/s]
Validation: 100%|██████████| 126/126 [01:22<00:00,  1.53it/s]
Epoch:17/17  Time: 9.70m
Total time: 147.87 m
```

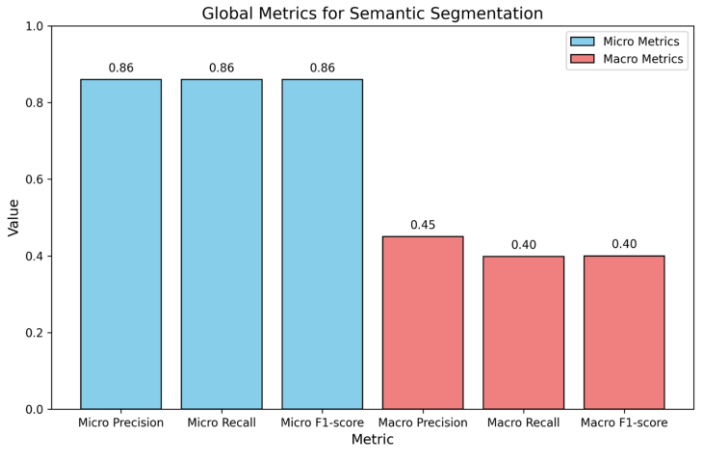


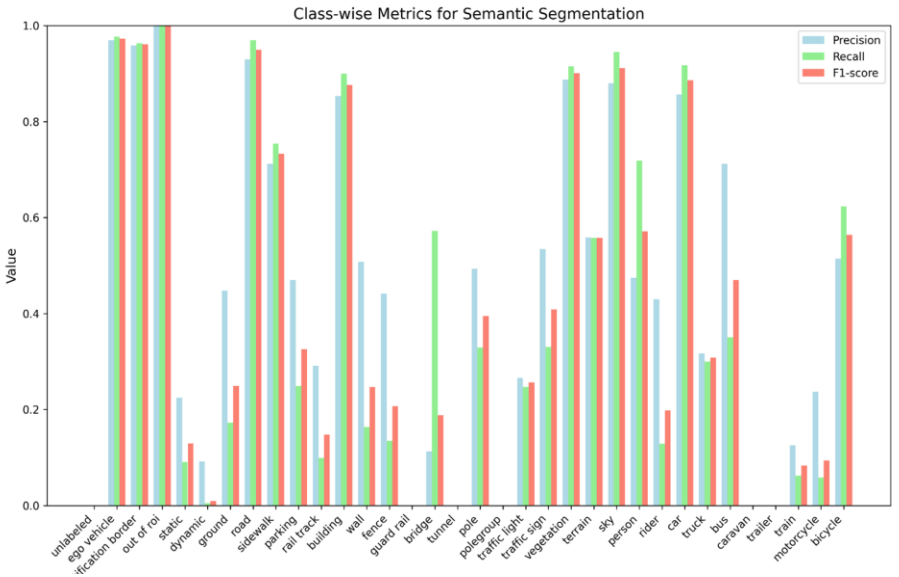




Training ResNet - 34 Classes - Focal Loss

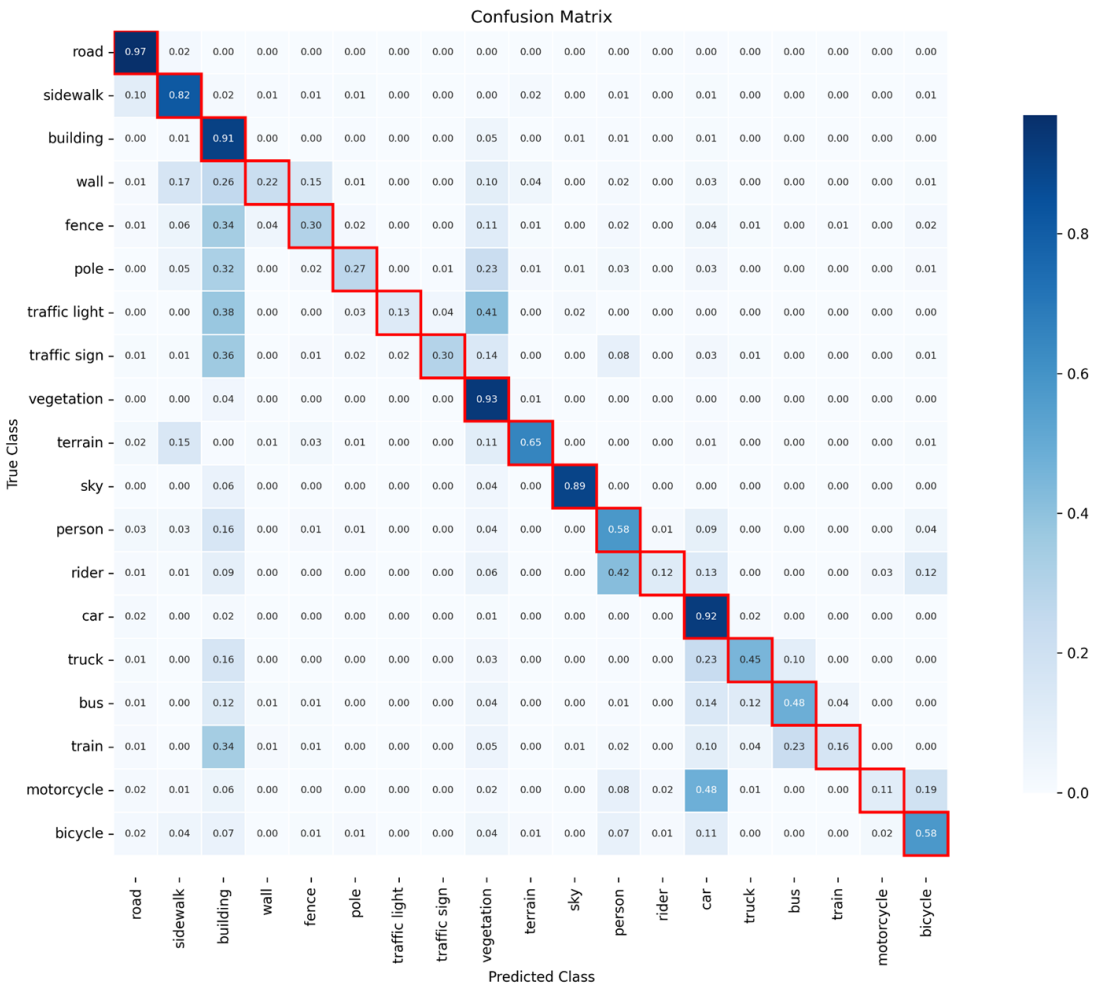


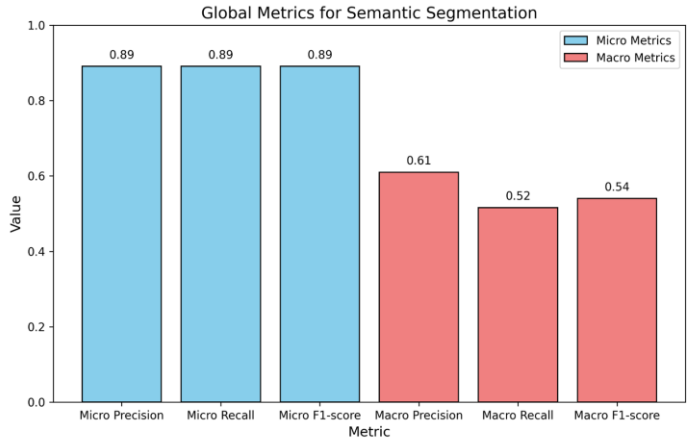


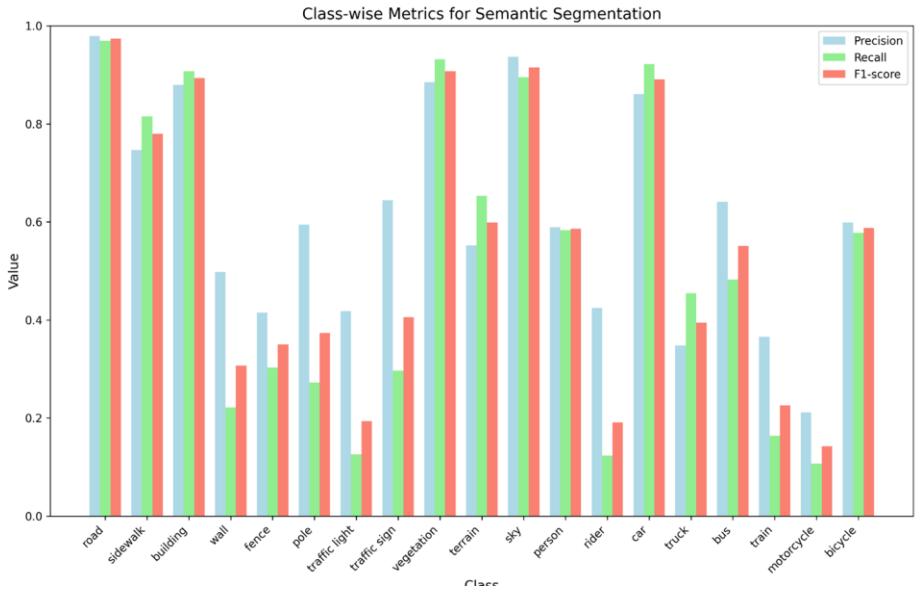


Jaccard macro: 0.33
Jaccard micro: 0.76

Training ResNet - 19 Classes - Focal Loss

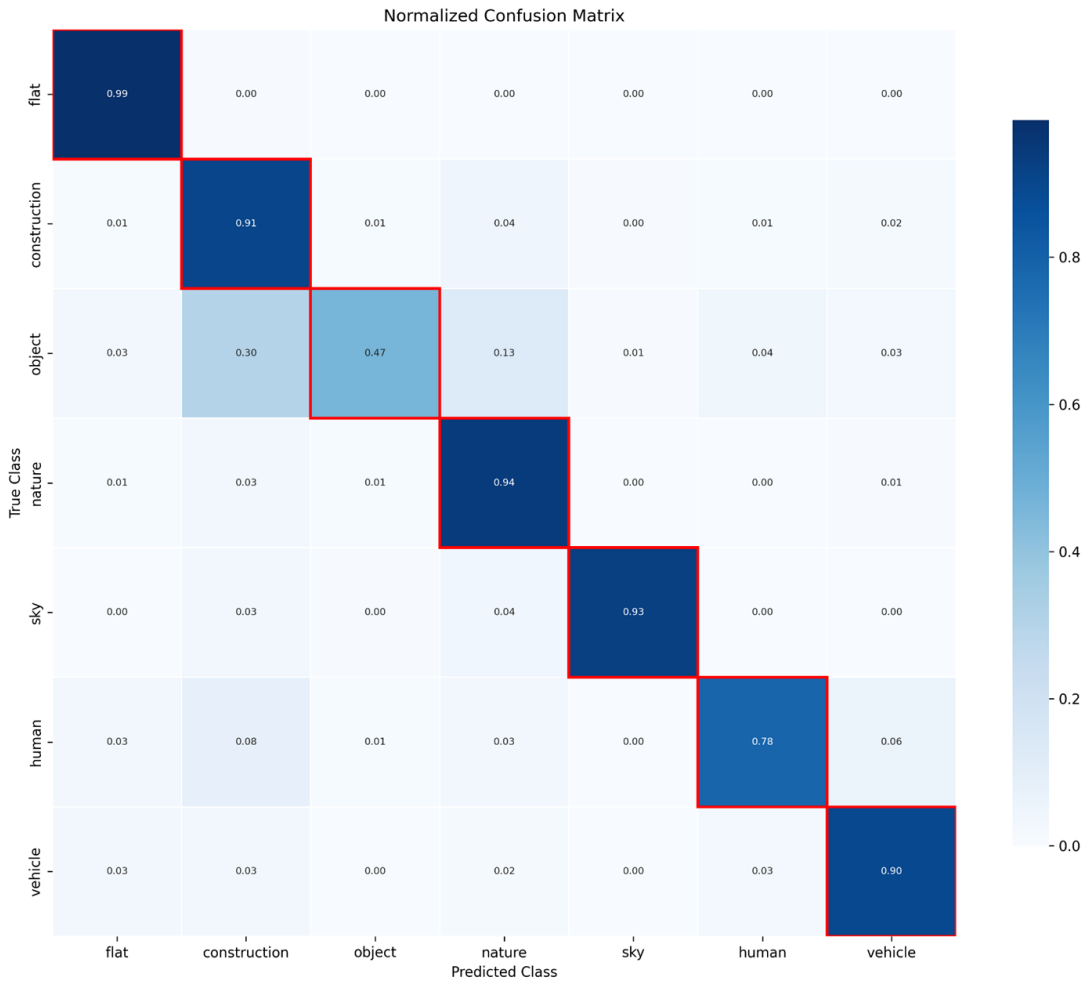


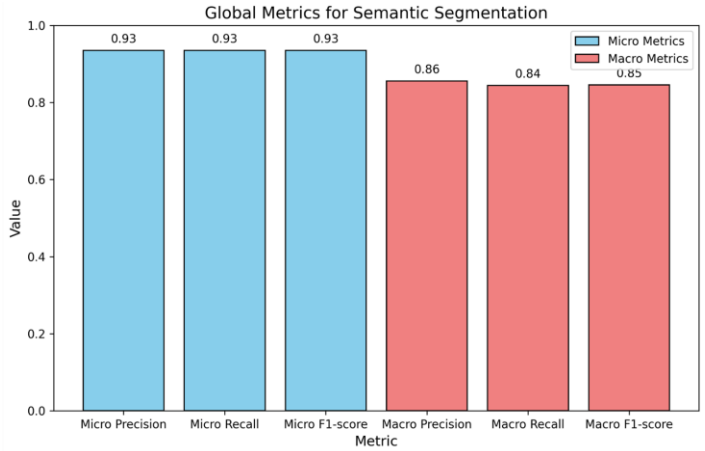


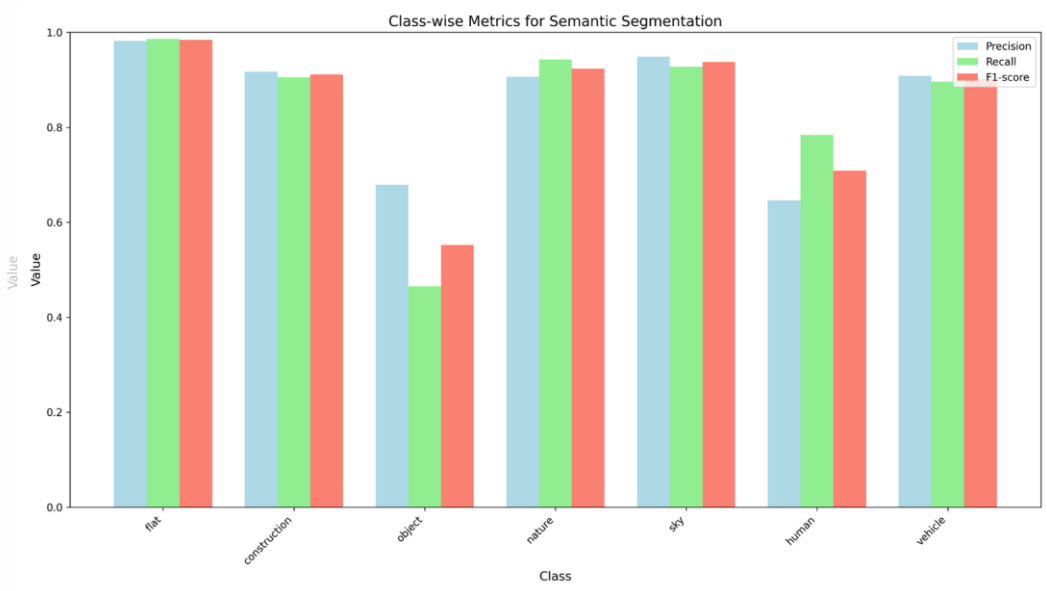


Jaccard macro: 0.37
Jaccard micro: 0.80

Training ResNet - 7 Categories - Focal Loss







Jaccard macro: 0.71
Jaccard micro: 0.88

Sommario

- Dataset
- Loss Functions and Evaluation Metrics
- Training U-Net
- Training ResNet
- **Training with ASPP**
- Extra: Transfer Learning

Training with ASPP

ASPP

Atrous

Atrous Convolution to extract features at multiple scales without increasing the number of parameters or reducing spatial resolution. allow the network to capture contextual information by varying the dilation rates.
Outputs from all the branches are **concatenated**. A **1x1 convolutional layer** is then used to mix these features and generate a unified representation.

Spatial **P**yramid

The **spatial pyramid design** ensures that features are aggregated from:

- Fine-grained details (using smaller dilation rates or no dilation).
- Larger context (using higher dilation rates or global pooling).

Pooling

global average pooling branch, which captures global context by summarizing the entire input feature map into a single vector.

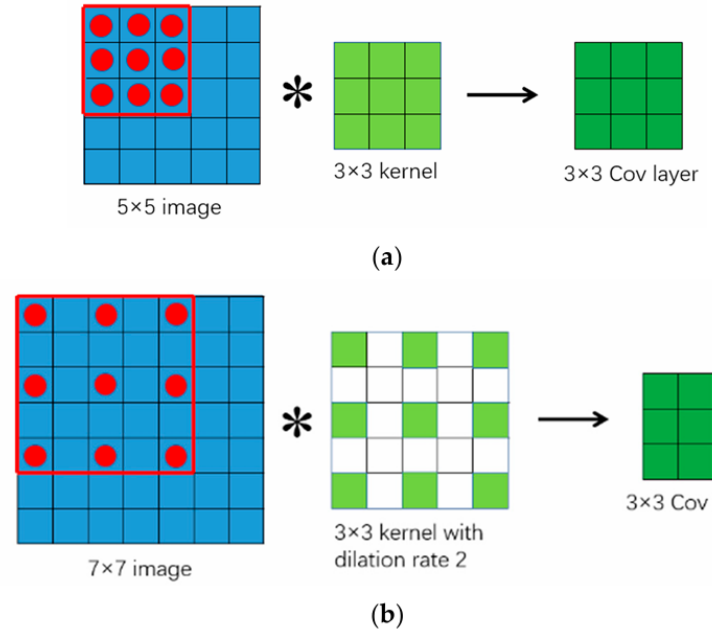


Figure 1. Illustrations of (**a**) convolution and (**b**) atrous convolution.

Si, Y., Gong, D., Guo, Y., Zhu, X., Huang, Q., Evans, J., He, S., & Sun, Y. (2021). An Advanced Spectral–Spatial Classification Framework for Hyperspectral Imagery Based on DeepLab v3+. *Applied Sciences*, *11*(12), 5703. https://doi.org/10.3390/app11125703

Training with ASPP

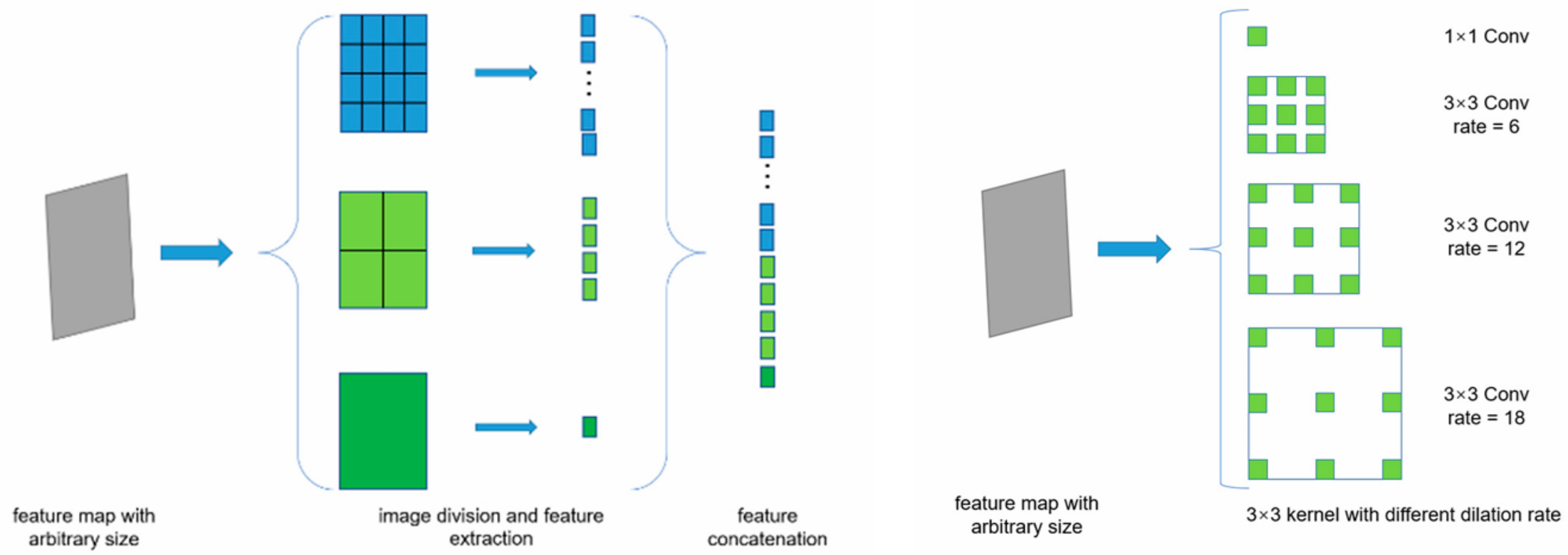


Figure 2. Illustration of (**a**) spatial pyramid pooling (SPP) structure and (**b**) atrous spatial pyramid pooling (ASPP) structure in DeepLab v3+.

Si, Y., Gong, D., Guo, Y., Zhu, X., Huang, Q., Evans, J., He, S., & Sun, Y. (2021). An Advanced Spectral–Spatial Classification Framework for Hyperspectral Imagery Based on DeepLab v3+. *Applied Sciences*, *11*(12), 5703. https://doi.org/10.3390/app11125703

Training with ASPP

[1]

[2]

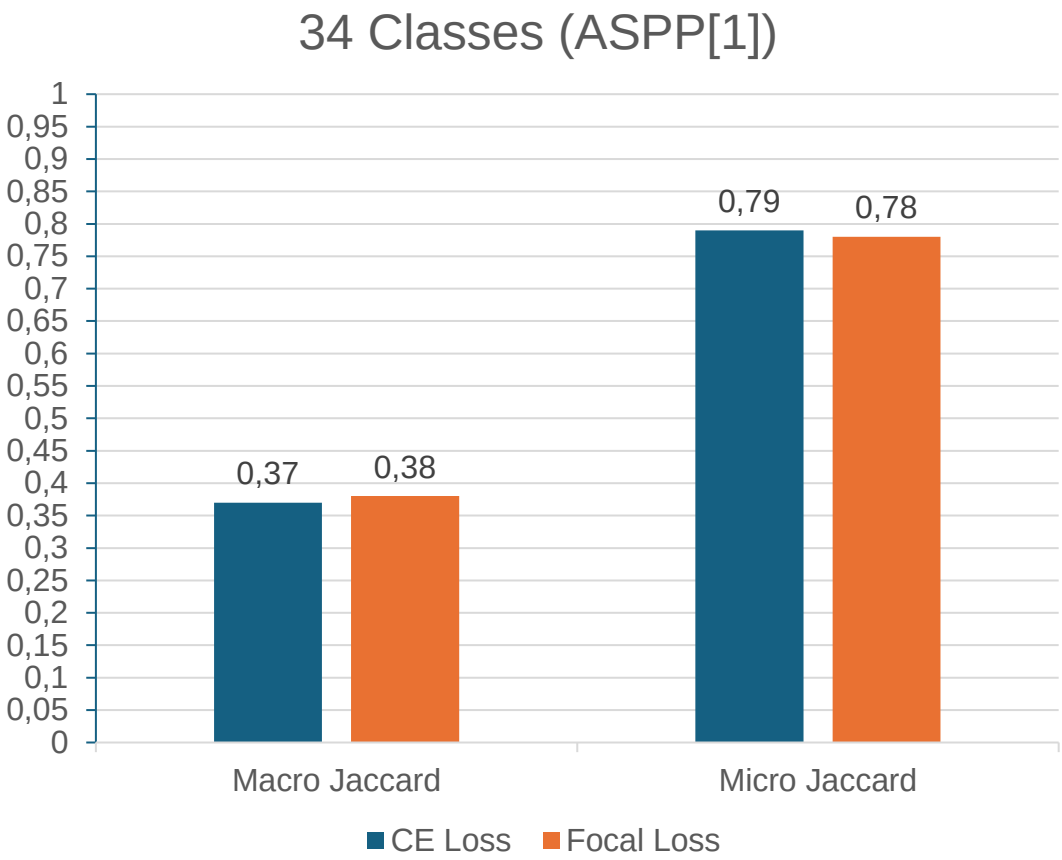
Training with ASPP

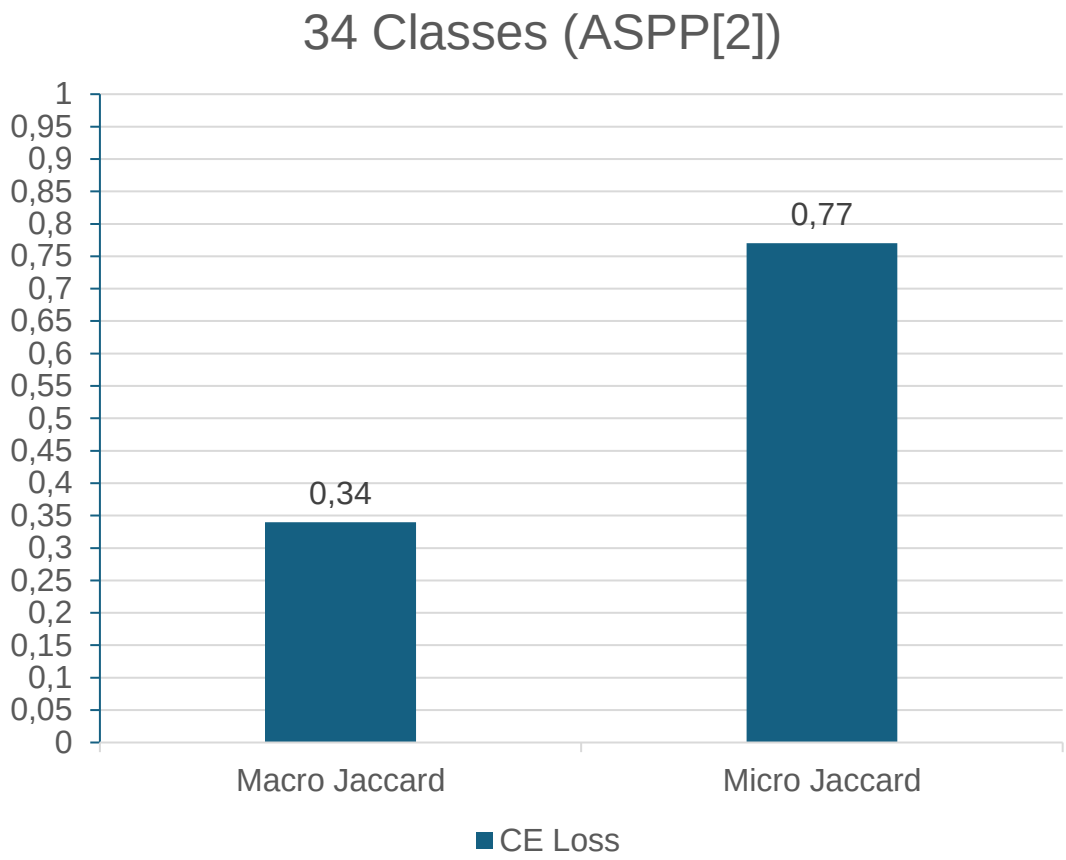
```python
class ASPP(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(ASPP, self).__init__()
        self.atrous1 = AtrousConv(in_channels, out_channels, kernel_size=1, dilation=1)
        self.atrous2 = AtrousConv(in_channels, out_channels, kernel_size=3, dilation=2)
        self.atrous4 = AtrousConv(in_channels, out_channels, kernel_size=3, dilation=4)
        self.atrous8 = AtrousConv(in_channels, out_channels, kernel_size=3, dilation=8)
        self.global_avg_pool = nn.Sequential(
            nn.AdaptiveAvgPool2d(1),
            nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )
        self.conv1 = nn.Conv2d(out_channels * 5, out_channels, kernel_size=1, bias=False)
        self.bn = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        x1 = self.atrous1(x)
        x2 = self.atrous2(x)
        x3 = self.atrous4(x)
        x4 = self.atrous8(x)
        x5 = self.global_avg_pool(x)
        x5 = F.interpolate(x5, size=x.shape[2:], mode='bilinear', align_corners=True)
        x = torch.cat([x1, x2, x3, x4, x5], dim=1)
        x = self.conv1(x)
        return self.relu(self.bn(x))
```

Trade-offs in Dilation Size:

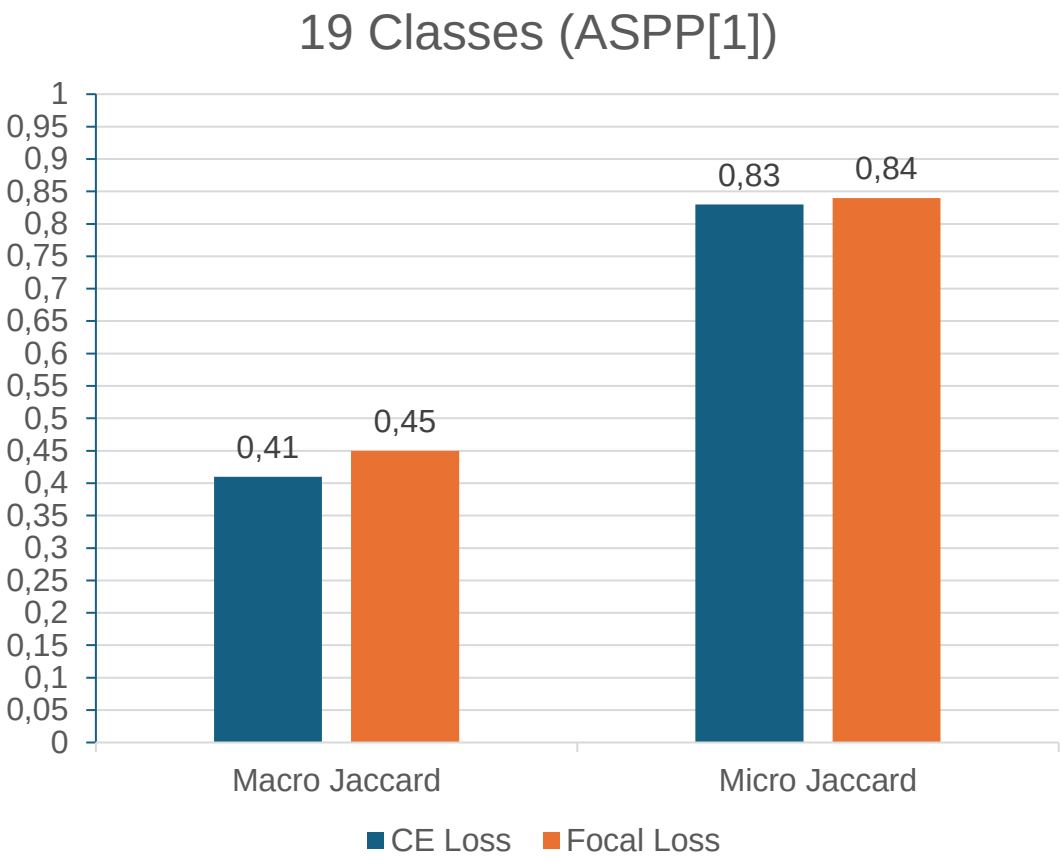
- **Too large dilation:**
  - Captures long-range context but risks losing fine details.
  - May introduce grid artifacts (checkerboard patterns).
- **Too small dilation:**
  - Focuses on local context but misses global context.

Training with ASPP



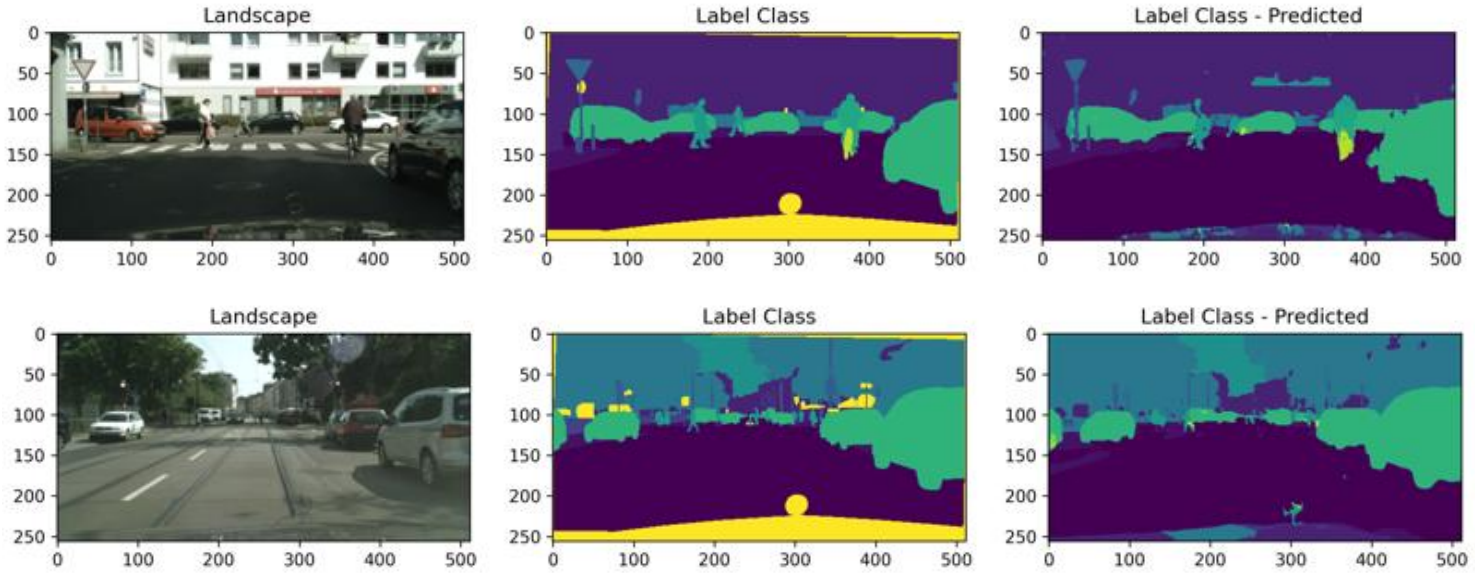


Training with ASPP

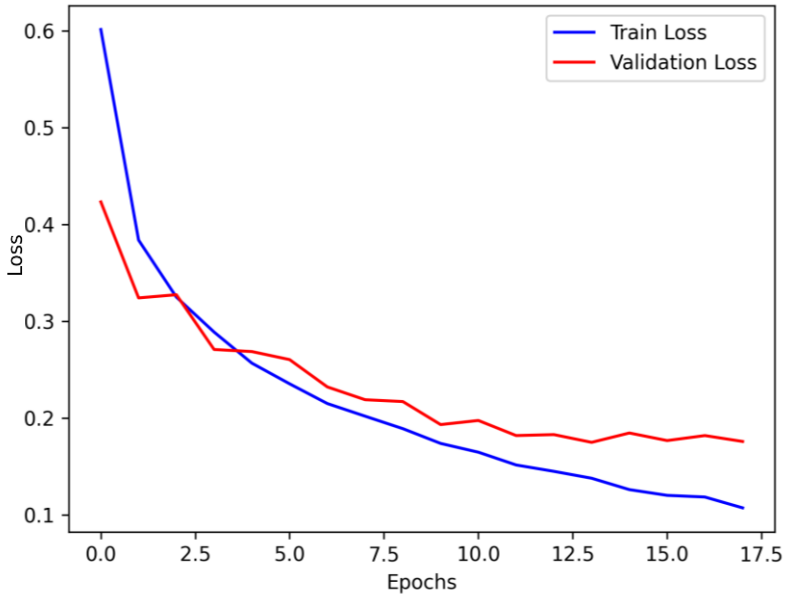


Training with ASPP - 19 Classes - Focal Loss [1]

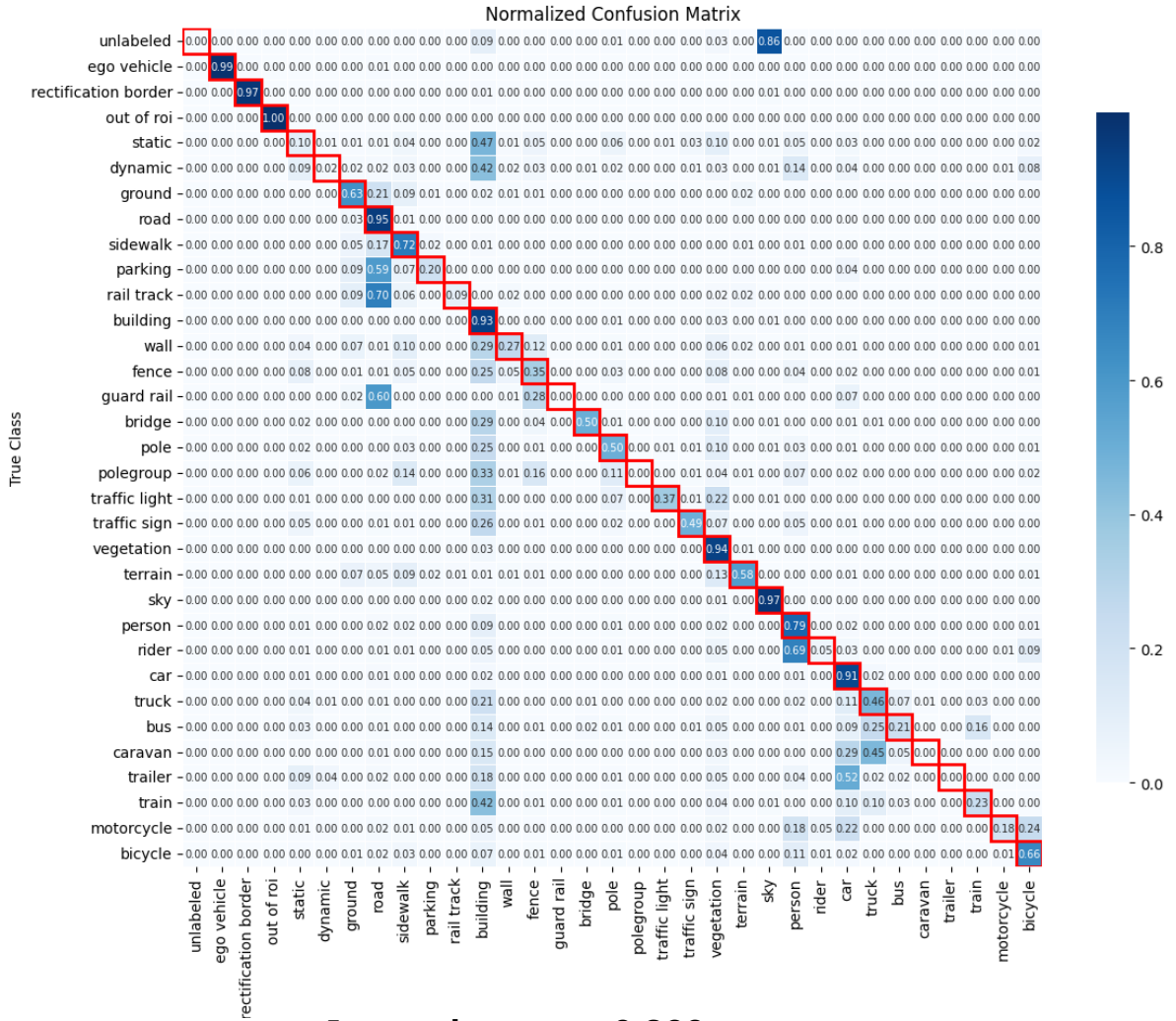
- size = 256x512
- epochs = 18
- criterion = FocalLoss(ignore_index = 255)
- optimizer = torch.optim.Adam
  (model.parameters(), lr=0.00007)

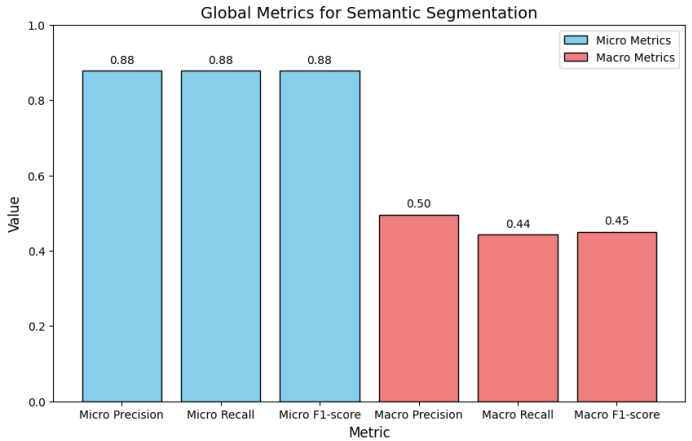


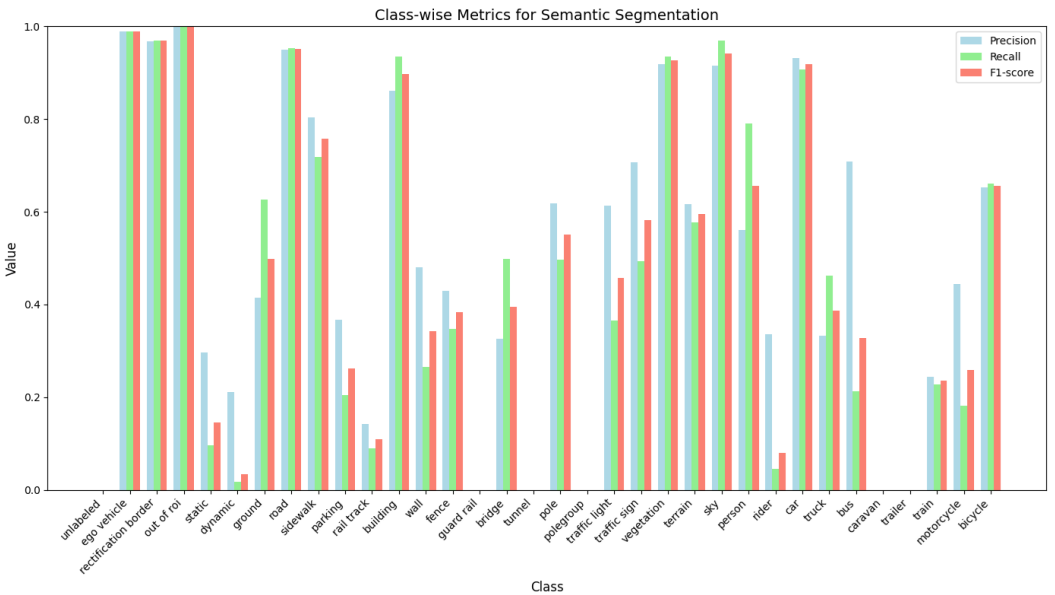

```
Training: 100%|██████████| 618/618 [07:07<00:00,  1.45it/s]
Validation: 100%|██████████| 126/126 [01:03<00:00,  1.98it/s]
Epoch:18/18 Time: 8.18m
Total time: 151.53 m
```



Training with ASPP - 34 Classes - CE Loss [1]

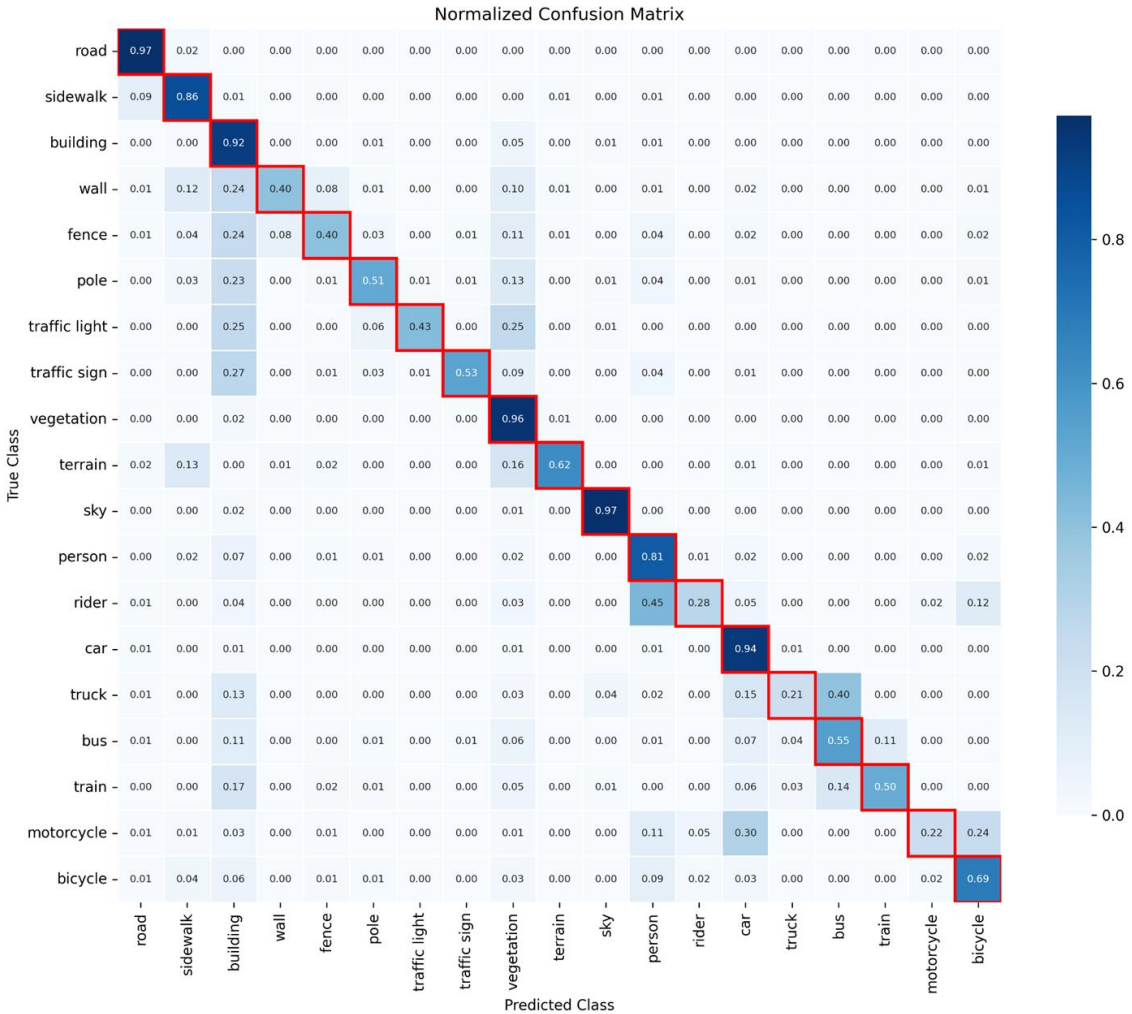




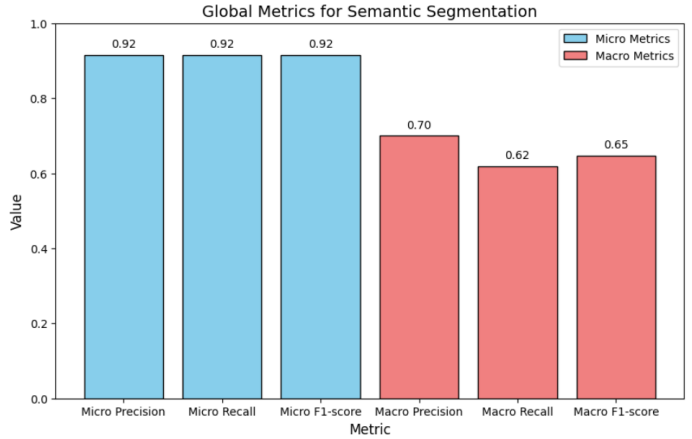


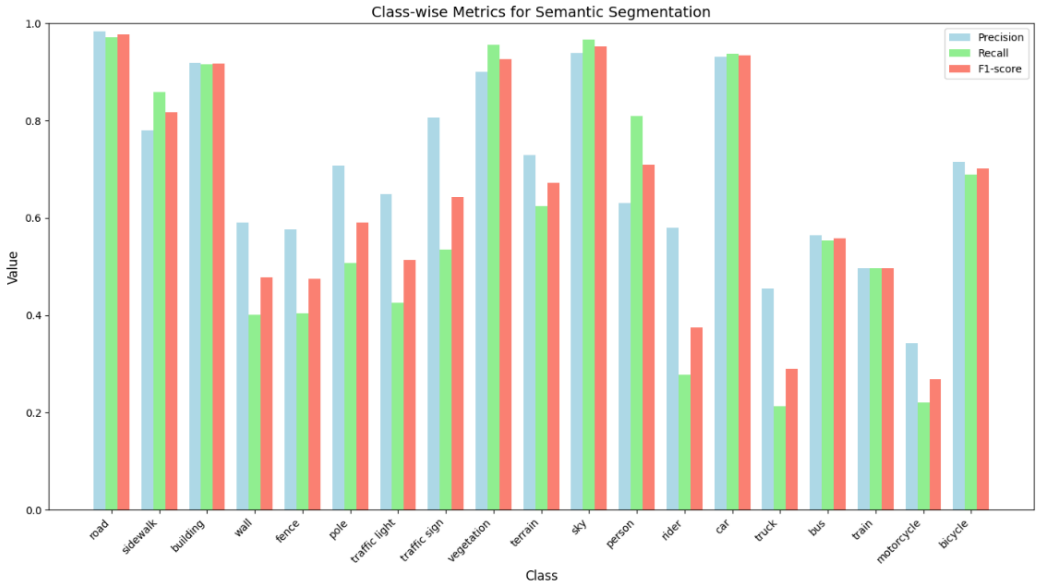
Jaccard macro : 0.366
Jaccard micro: 0.786

Training with ASPP - 19 Classes - Focal Loss [1]

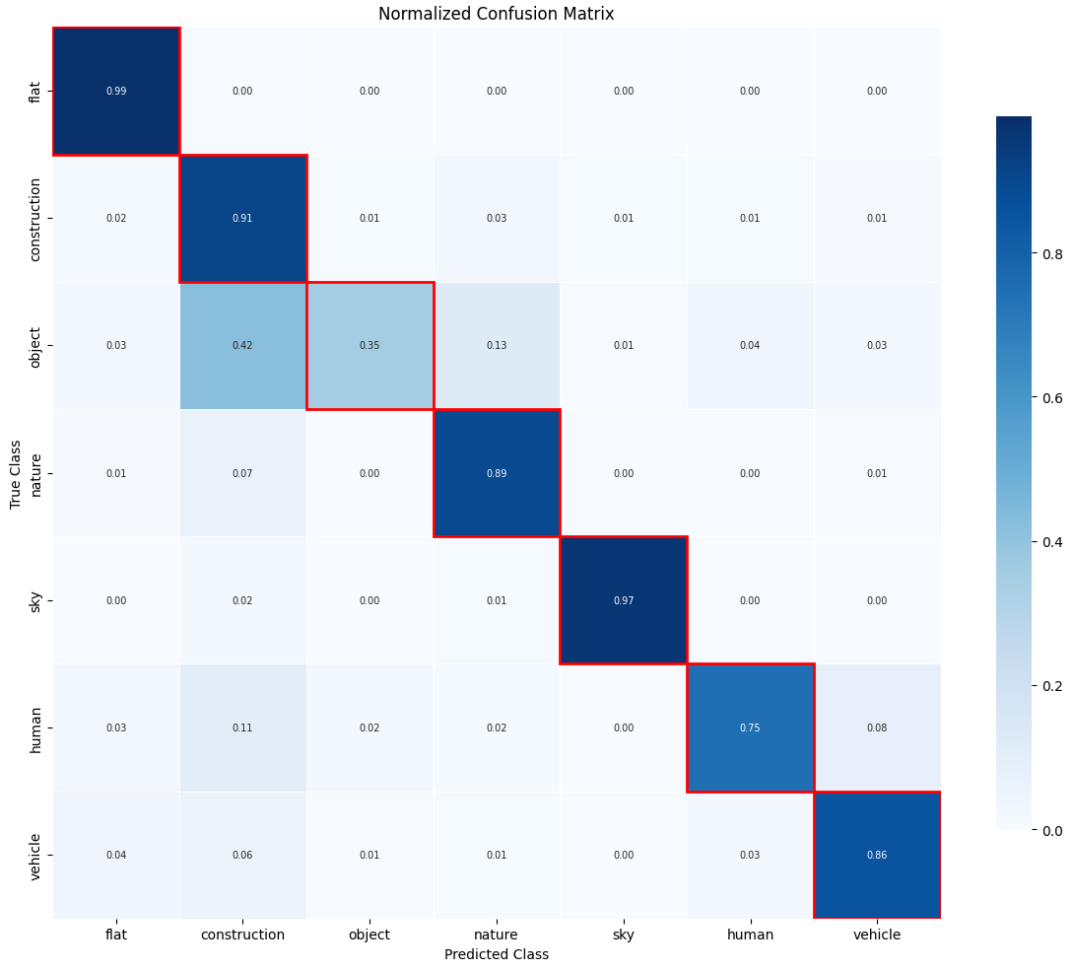


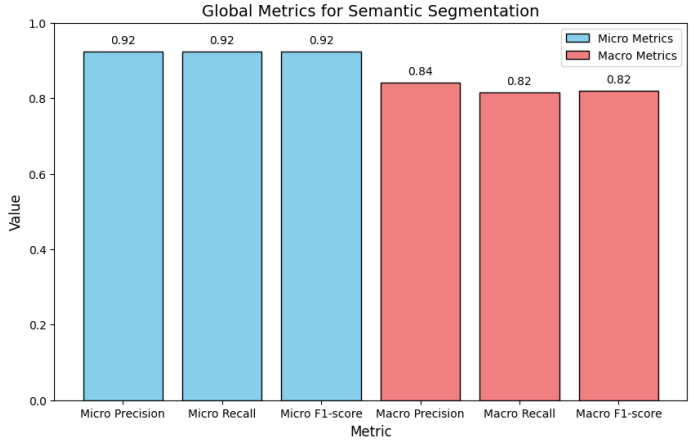
Jaccard macro: 0.45
Jaccard micro: 0.84

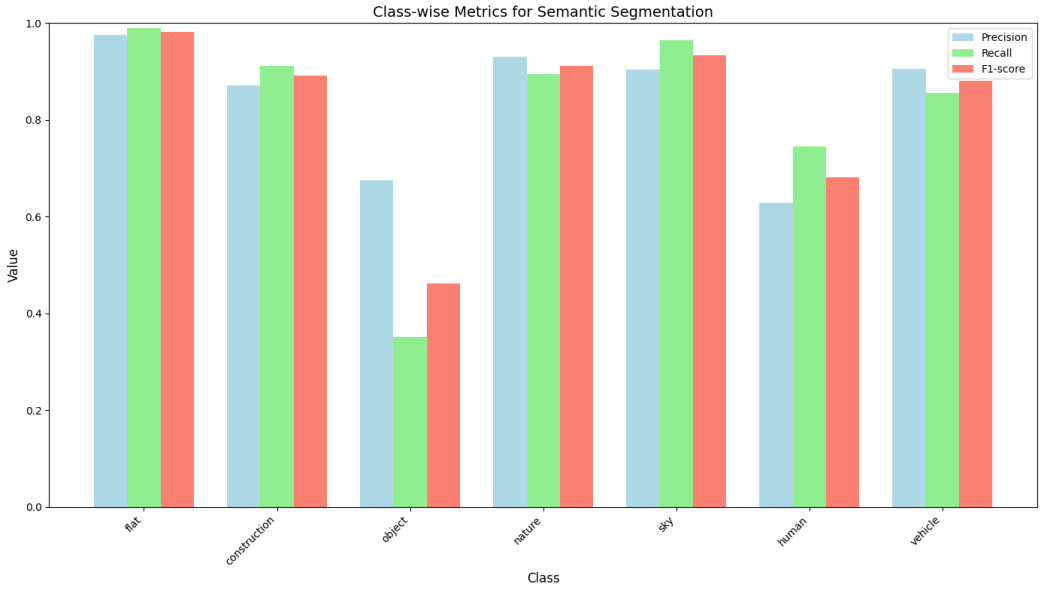




Training with ASPP - 7 Categories - Focal Loss [1]



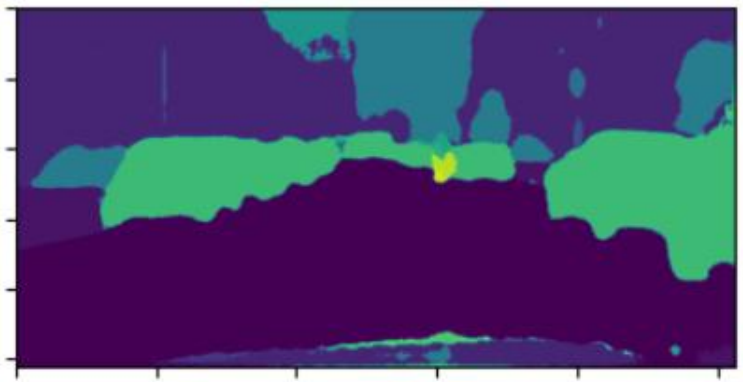


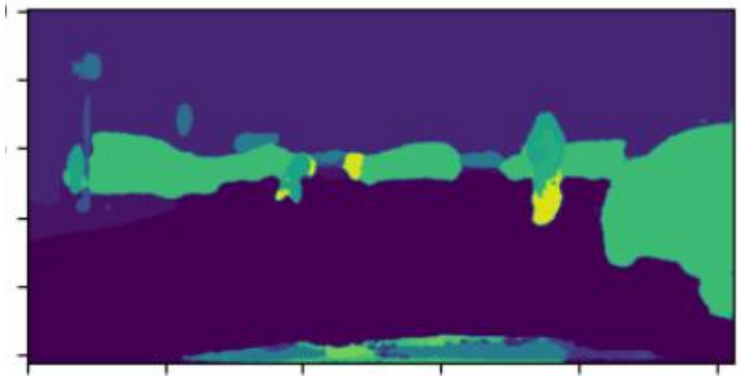


Jaccard macro: 0.70
Jaccard micro: 0.86

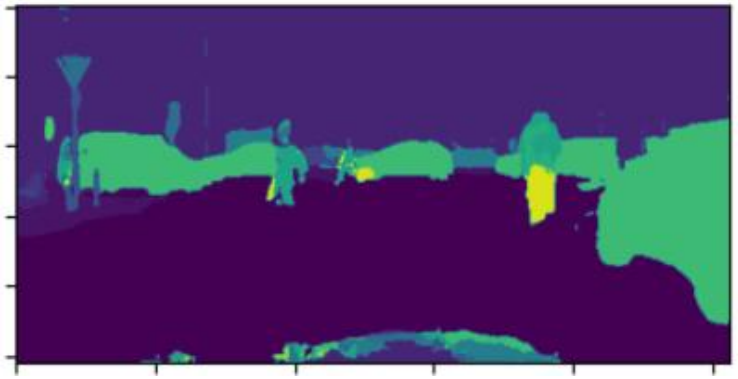
Training comparison

ResNet

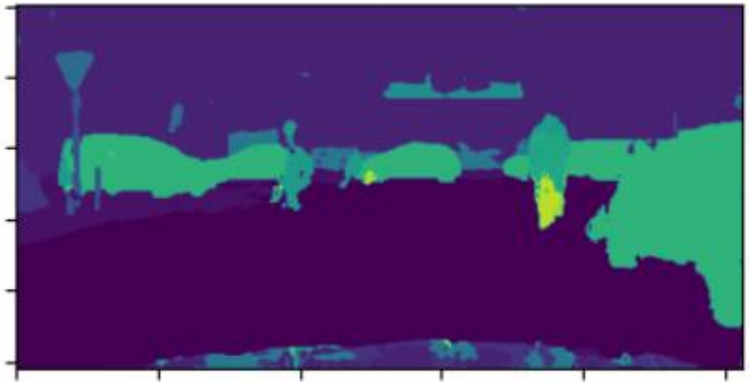




U-Net



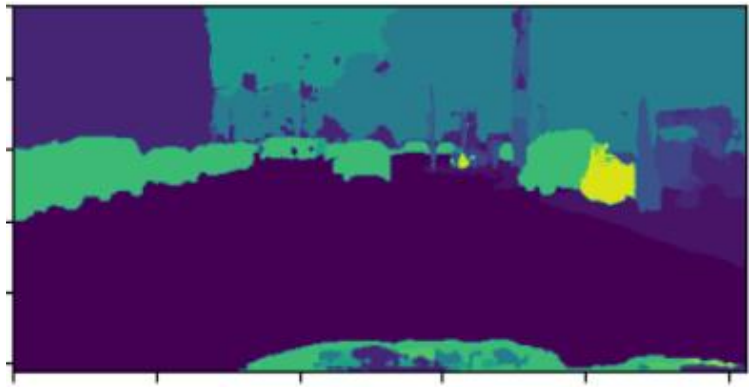
U-Net ASPP



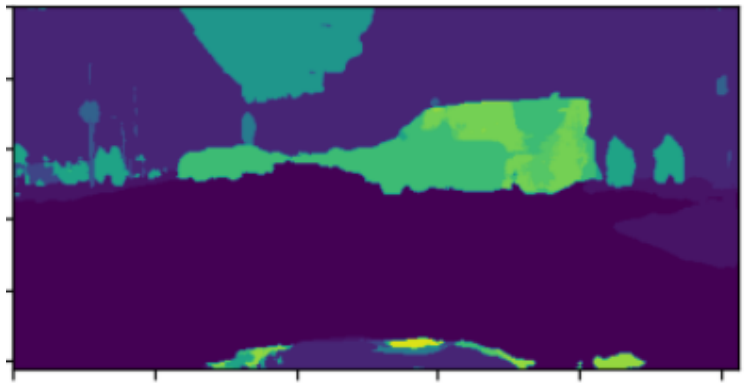
Training comparison

ResNet

U-Net



U-Net ASPP

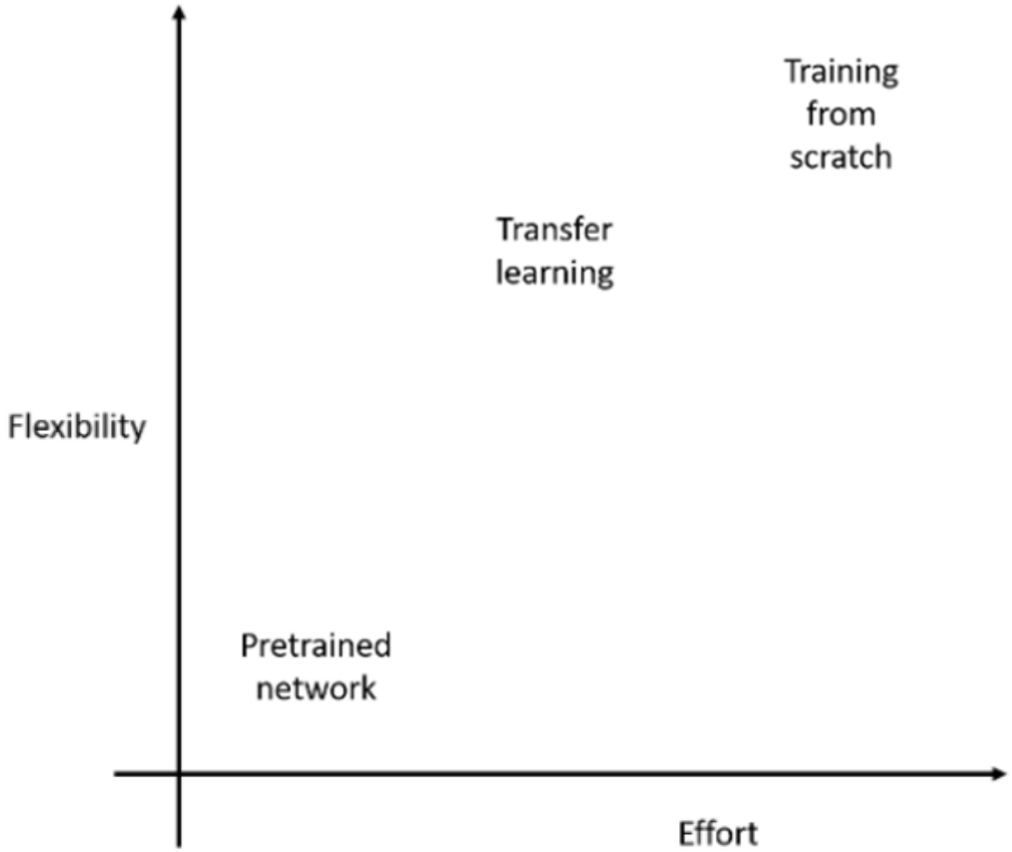


Sommario

- Dataset
- Loss Functions and Evaluation Metrics
- Training U-Net
- Training ResNet
- Training with ASPP
- **Extra: Transfer Learning**

Extra: Transfer Learning

- Pretrained network:
  - It is extremely easy to get started using a pretrained network but there is no flexibility in the way the network operates
- Training from scratch:
  - Starting with just the network architecture and random weights, it is possible to get a network suited to a specific problem. However, achieving reasonable results requires experience with network architecture, a huge amount of training data, and a lot of computer time.
- Transfer Learning:
  - Training requires some data and computer time, but much less than training from scratch. The result is a network suited to a specific problem



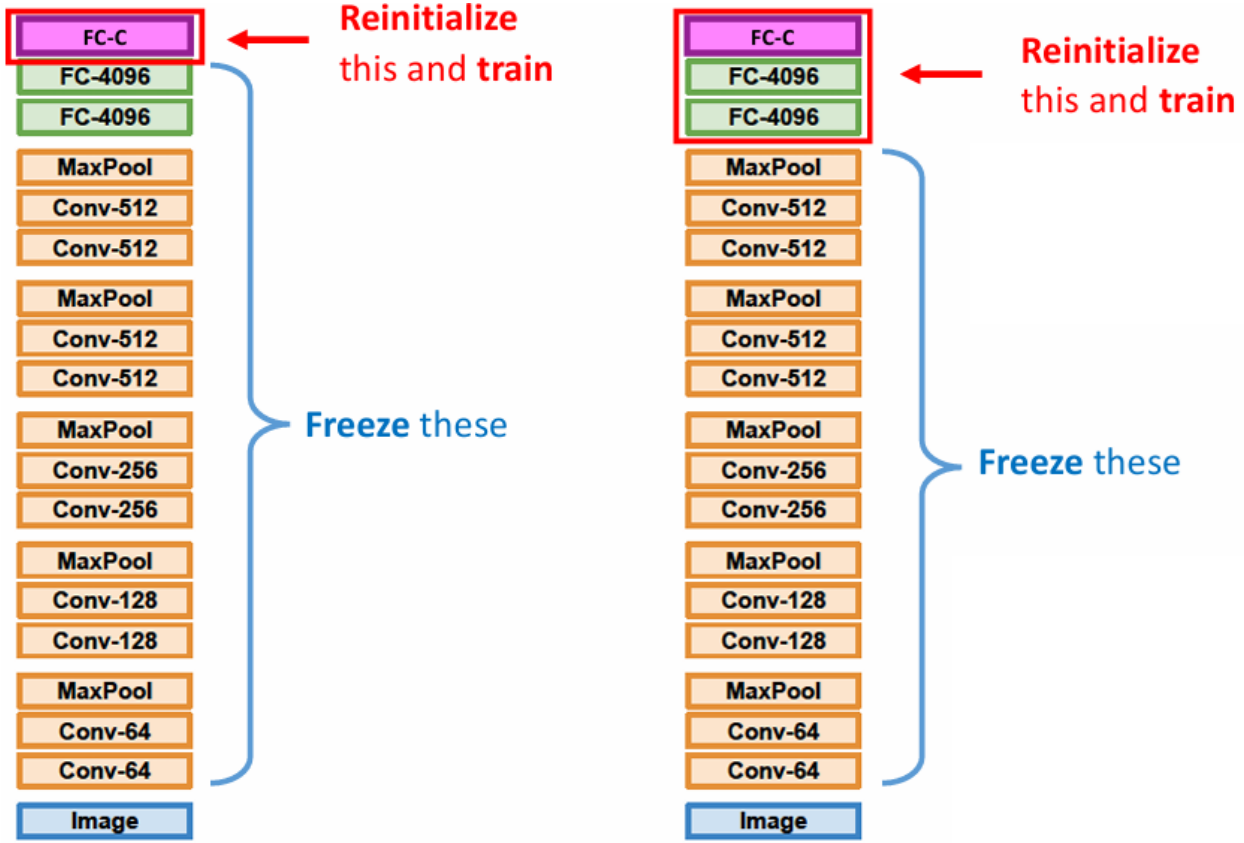
Extra: Transfer Learning

- Fine-Tuning Deeplab
  - robust and well-trained model
  - good dataset

  → freezing more layers

- Porting to SUIM
  - weak and undertrained model
  - bad dataset

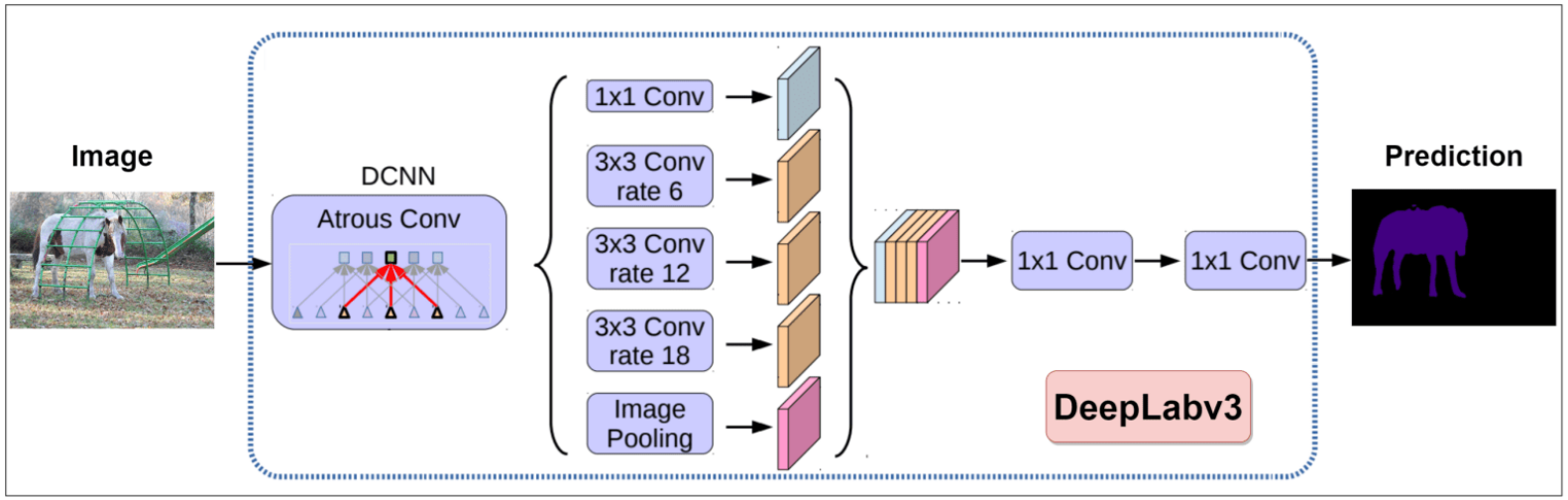
  → freezing few layers



Extra: Transfer Learning

- ResNet-101 DCNN model as the backbone.
- When OS (output stride)=8, the last two blocks (block3 and block4) contain atrous convolution with dilation rate=2 and rate=4, respectively.
- Whereas for OS=16, the dilation rate of just block4 is set to rate=2.
- Output feature map from block4 is passed through the modified ASPP module.
- The outputs from all five sub-modules in the ASPP are concatenated and passed through the pen-ultimate (1×1) convolutional block. This block is for mixing the feature maps acquired by different sub-modules.
- Finally the feature map passes through the final (1×1) convolution before being bilinearly upsampled depending on the OS.

```python
import torch
from torchvision import models
from torchvision.models.segmentation.deeplabv3 import DeepLabHead

# Carica il modello pre-addestrato
model = models.segmentation.deeplabv3_resnet101(pretrained=True)

# Modifica l'ultimo layer per Cityscapes (19 classi)
num_classes = 19
model.classifier = DeepLabHead(2048, num_classes)

# Congelare i parametri del backbone
for param in model.backbone.parameters():
    param.requires_grad = False
```

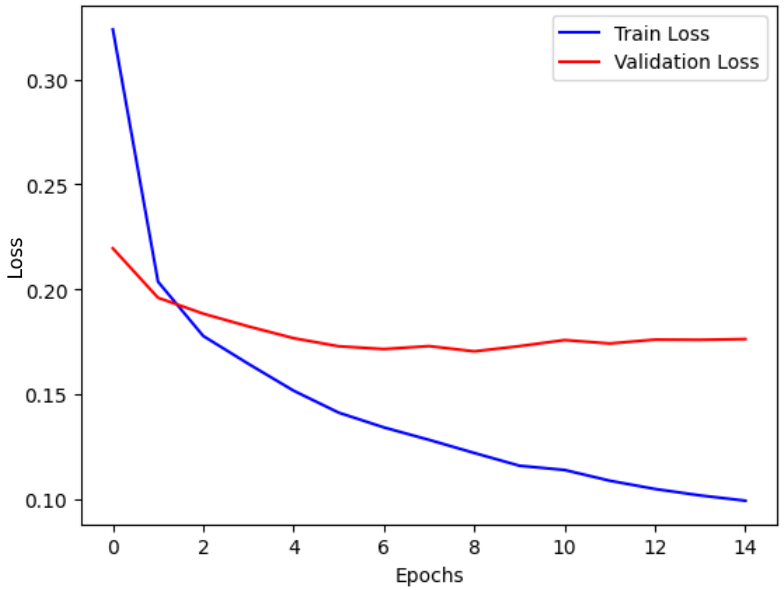


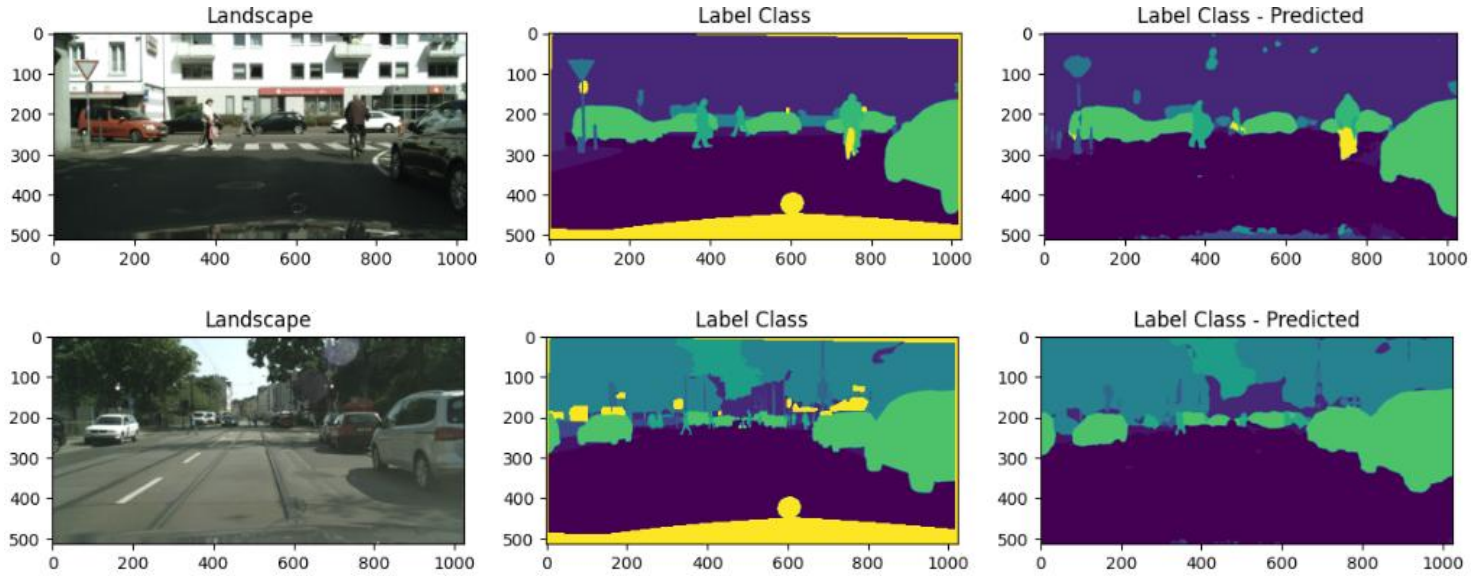
DeepLabv3 & DeepLabv3+ The Ultimate PyTorch Guide

Extra: Transfer Learning

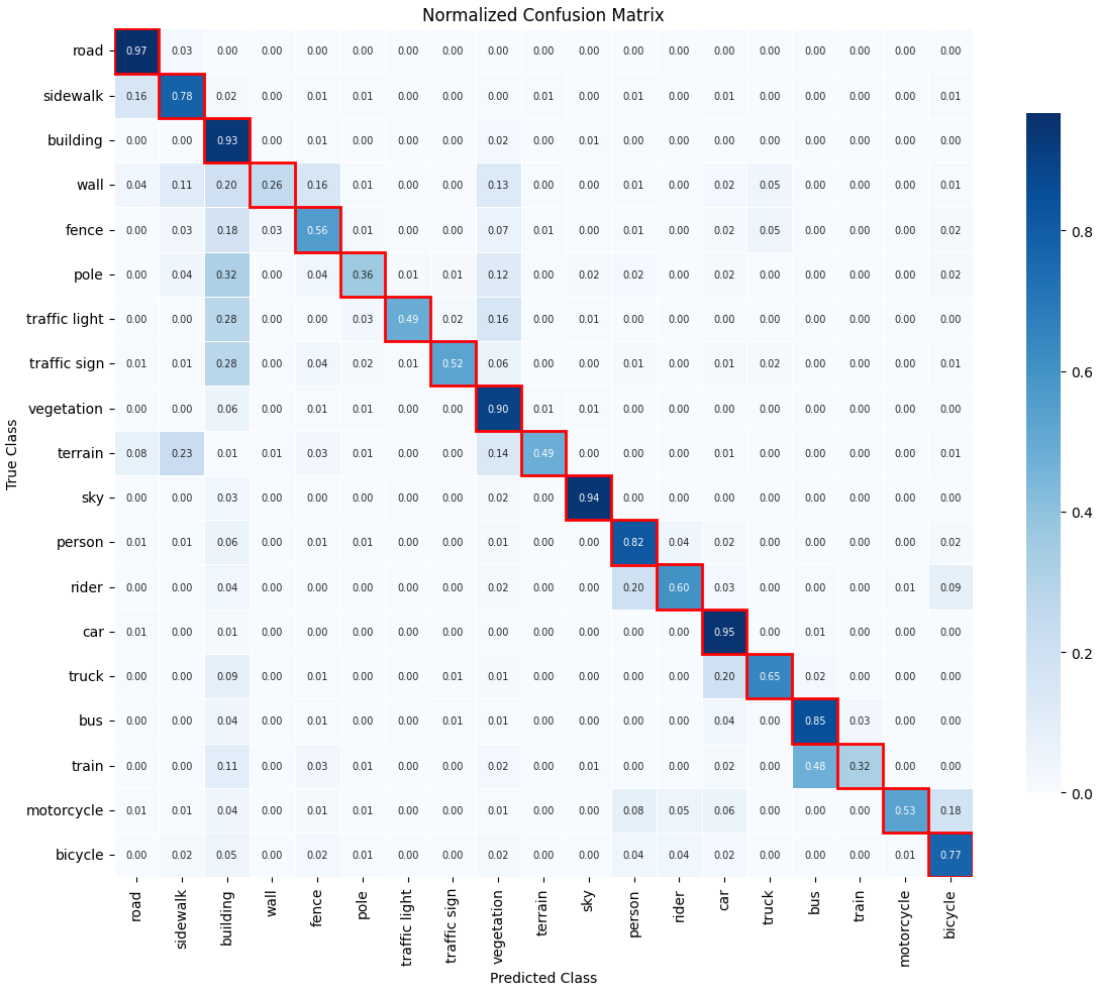
- size = 512x1024
- epochs = 15
- criterion = FocalLoss(ignore_index = 255)
- optimizer = torch.optim.SGD(
  filter(lambda p: p.requires_grad, model.parameters()),
  lr=0.01, momentum=0.9, weight_decay=0.0001)

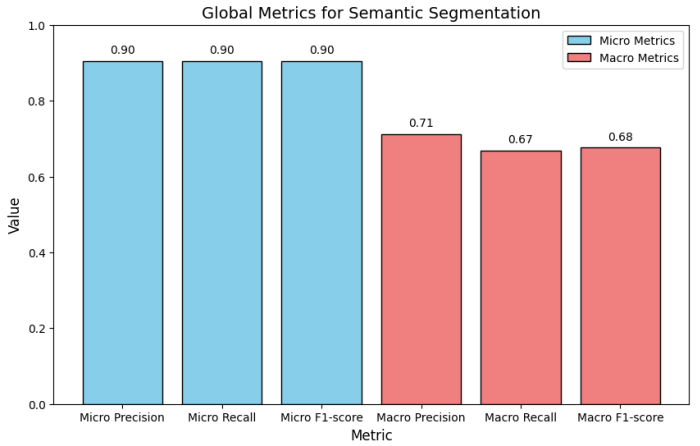
```
Training: 100%|██████████| 618/618 [15:19<00:00,  1.49s/it]
Validation: 100%|██████████| 126/126 [02:34<00:00,  1.23s/it]
Epoch:15/15  Time: 17.90m
Total time: 272.32 m
```

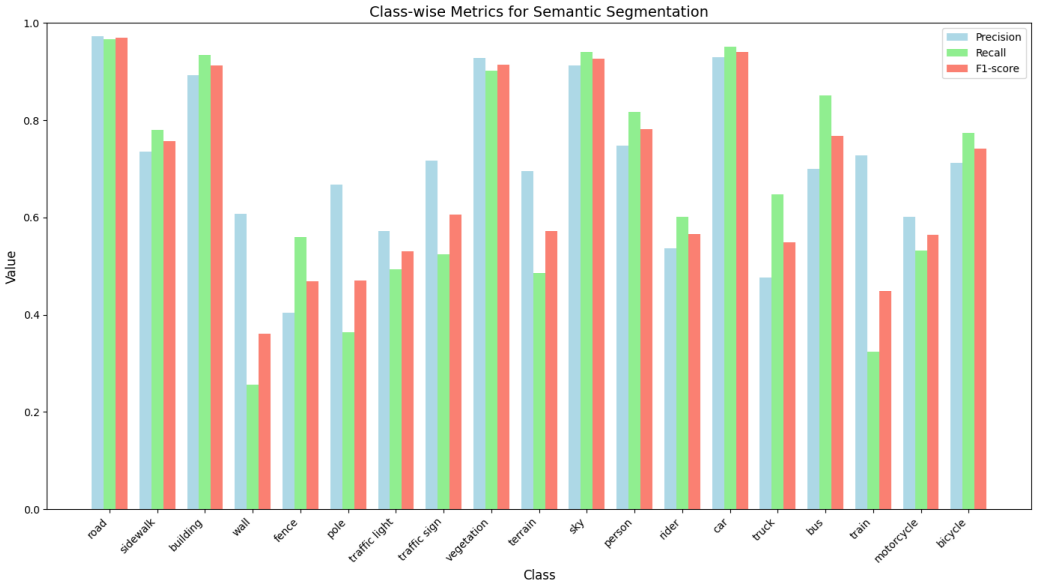




Extra: Transfer Learning







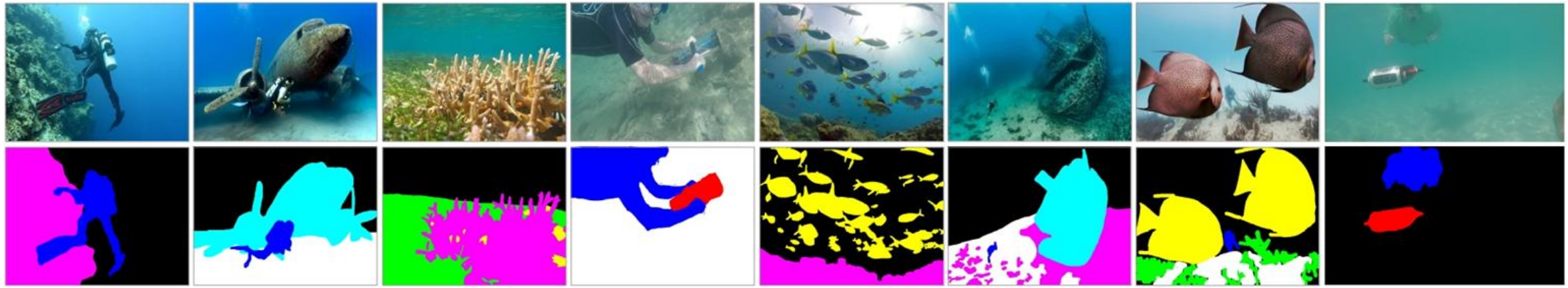
Jaccard macro: 0.465
Jaccard micro: 0.825

Extra: Transfer Learning

SUIM Dataset

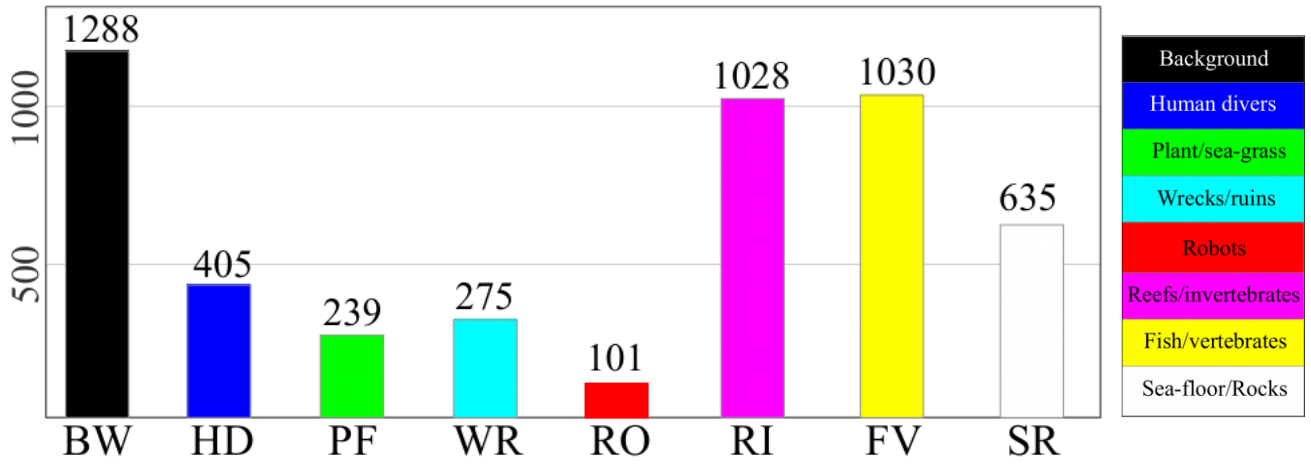
- For semantic segmentation of natural underwater images
- 1525 annotated images for training/validation and 110 samples for testing

Object category	RGB color	Code
Background (waterbody)	000	BW
Human divers	001	HD
Aquatic plants and sea-grass	010	PF
Wrecks or ruins	011	WR
Robots (AUVs/ROVs/instruments)	100	RO
Reefs and invertebrates	101	RI
Fish and vertebrates	110	FV
Sea-floor and rocks	111	SR



arxiv.org/pdf/2004.01241

Extra: Transfer Learning

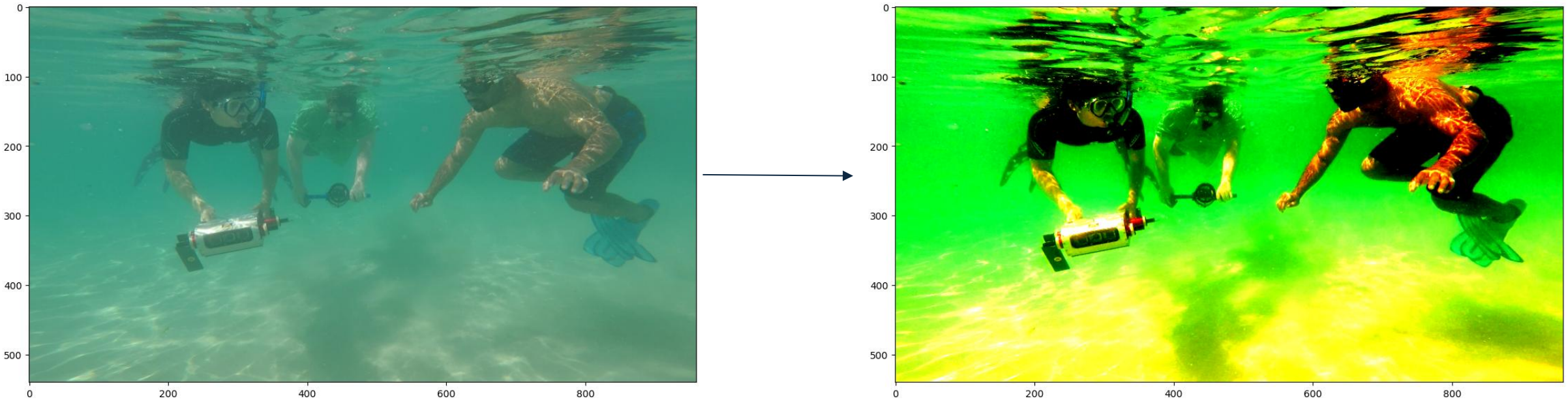


(a) The number of images containing each object category.

	Dimensions	Occurrences
0	(640, 480)	1329
1	(960, 540)	130
2	(1280, 720)	57
3	(590, 375)	31
4	(640, 360)	17
5	(1906, 1080)	17
6	(480, 448)	16
7	(800, 600)	13
8	(1152, 720)	8
9	(910, 435)	6
10	(416, 416)	5
11	(584, 480)	5
12	(1280, 960)	1

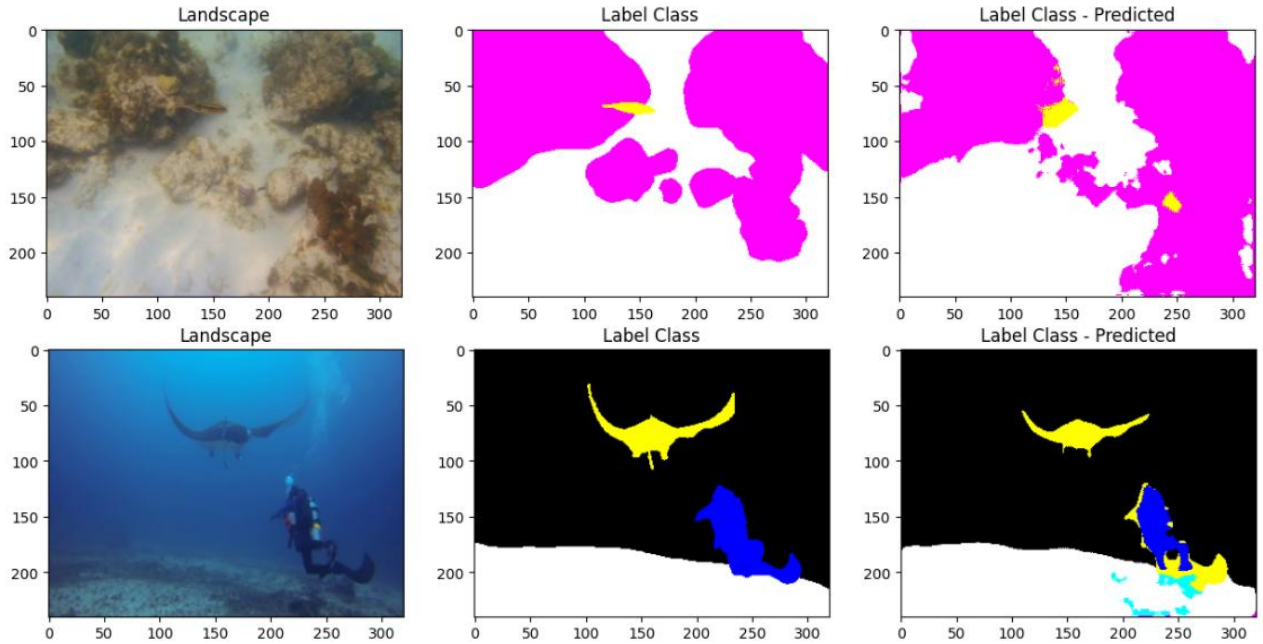
Extra: Transfer Learning

```
T.Normalize(mean=(0.2476039,  0.42255569, 0.48715457), std=(0.16100399, 0.18140797, 0.19568597))
```

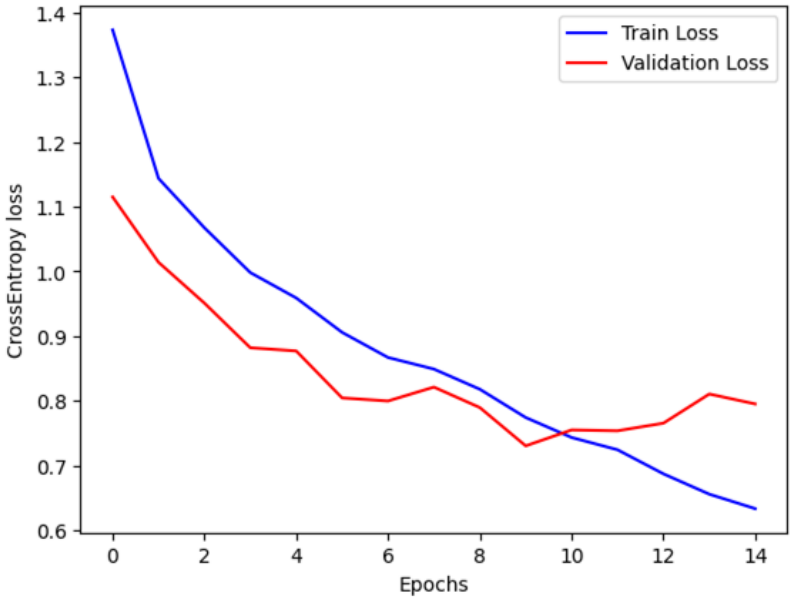


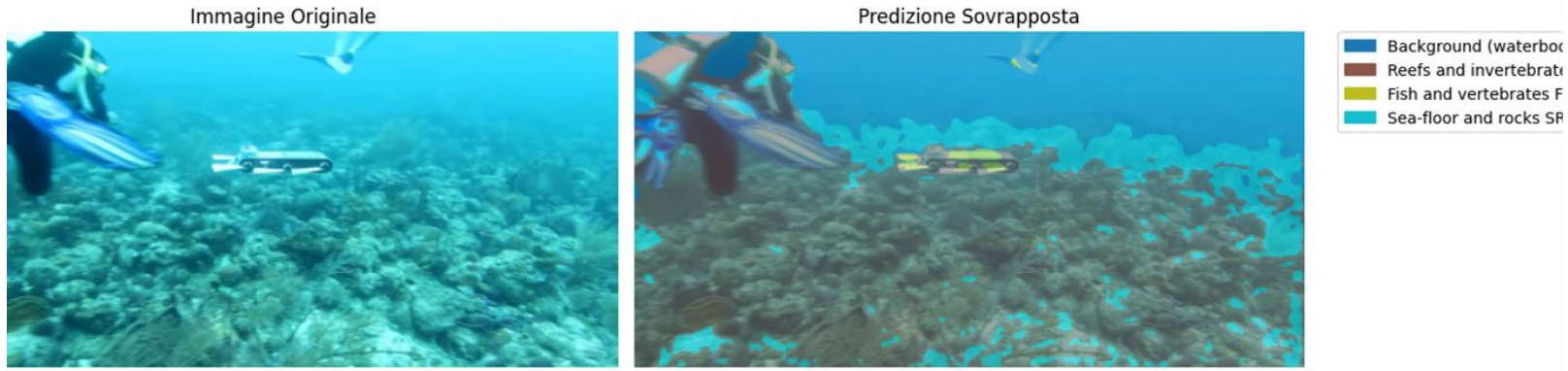
Extra: Transfer Learning

- size = 240x320 (from paper)
- epochs = 15
- criterion = nn.CrossEntropyLoss()
- optimizer = torch.optim.AdamW(model.parameters(), lr=0.0001)

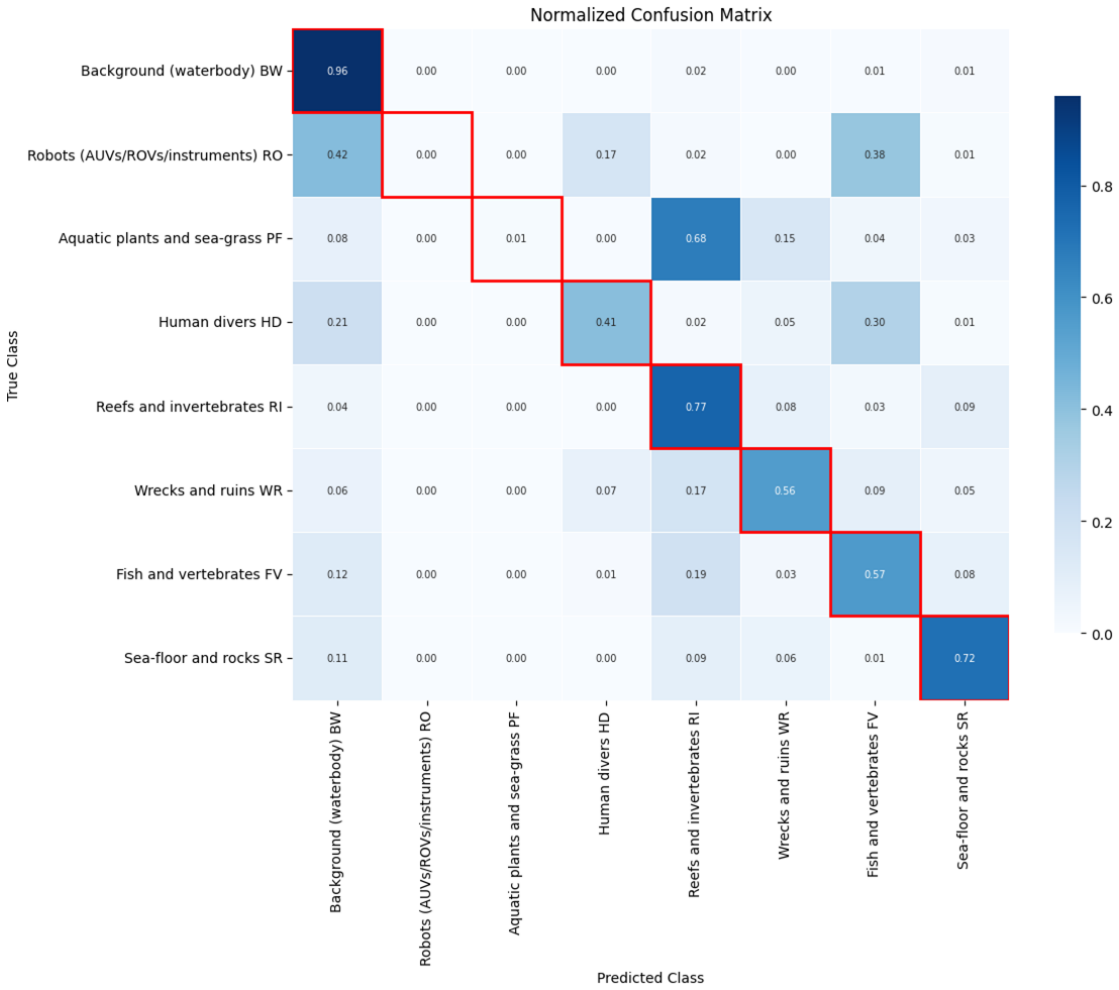


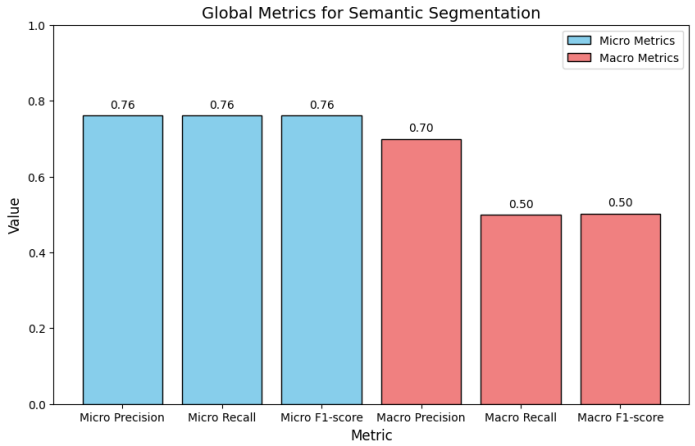

```
Training: 100%|██████████| 172/172 [03:14<00:00,  1.13s/it]
Validation: 100%|██████████| 19/19 [00:17<00:00,  1.12it/s]
Epoch:15/15 Time: 3.53m
Total time: 51.41 m
```

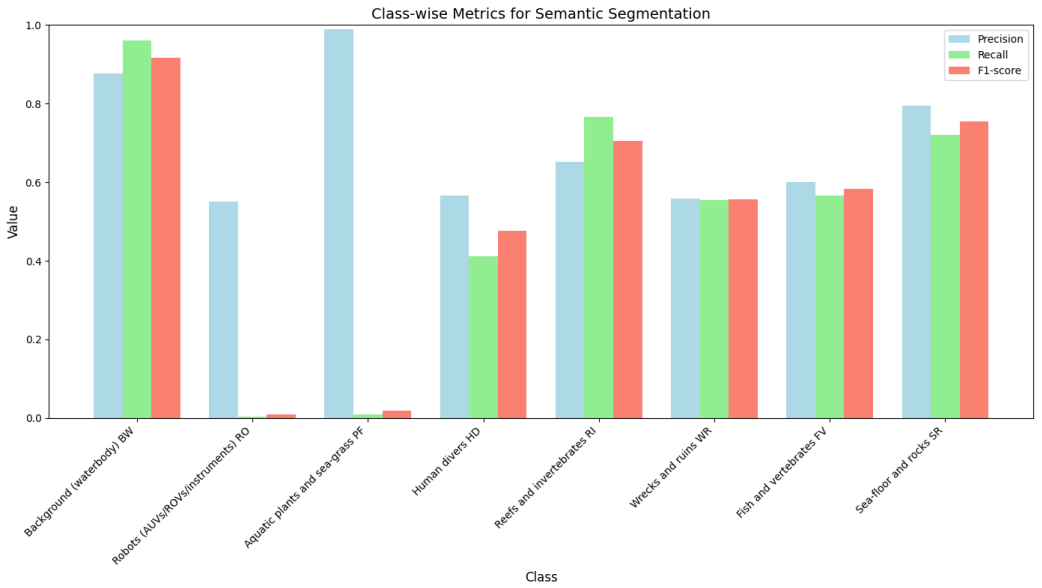




Extra: Transfer Learning







Jaccard IoU macro: 0.358
Jaccard IoU micro: 0.620