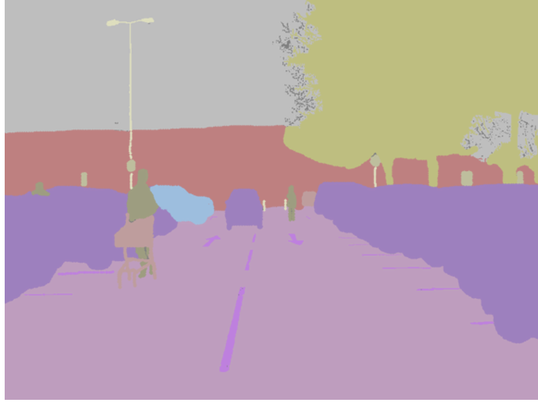
Semantic Segmentation Using Deep Learning

A PROJECT DEVELOPED BY:

MARCO AONDIO
ENRICO BARRETTA
GIANLUCA LICCIARDELLO

Problem Statement



This project aims to perform semantic segmentation on images, assigning a pixel-level class label to each part of the image. The goal is to accurately **classify each pixel** into one of **8 semantic classes**: Sky, Buildings, Poles and Cones, Roads, Pavements, Vegetation, Vehicles and Humans.

Semantic segmentation goes beyond basic image classification by providing a detailed understanding of the scene: it generates a dense, color-coded map where **each pixel is labeled according to the object or region it belongs to**.

To achieve this, we will build and train a deep learning model capable of learning from labeled datasets and correctly predicting the class of every pixel in new, unseen images. The output will be segmented images where **each pixel is visually marked with a color representing its class**, effectively producing a colorized map of the image's semantic content.

Dataset Presentation

For our project, we decided to work with the **CamVid** (Cambridge-driving Labeled Video) dataset, a well-known benchmark for semantic segmentation tasks in computer vision. This dataset consists of **video sequences captured by a dashcam on a moving vehicle**, providing realistic driving scenarios for developing and testing scene understanding models.

We used two complementary datasets derived from these videos:

- The first dataset contains **701 RGB image fragments** (original frames).
- The second dataset contains **701 colored fragments** (ground truth segmentation masks), where each object category is represented by a unique color.

Each class is represented by a specific color in the segmentation masks. Every RGB frame is paired with its corresponding colored mask, enabling precise pixel-level labeling.

Steps to the solution

Verification of the Augmentation

Train/Validation/Test split

Creation of the first model

Choice of the metrics

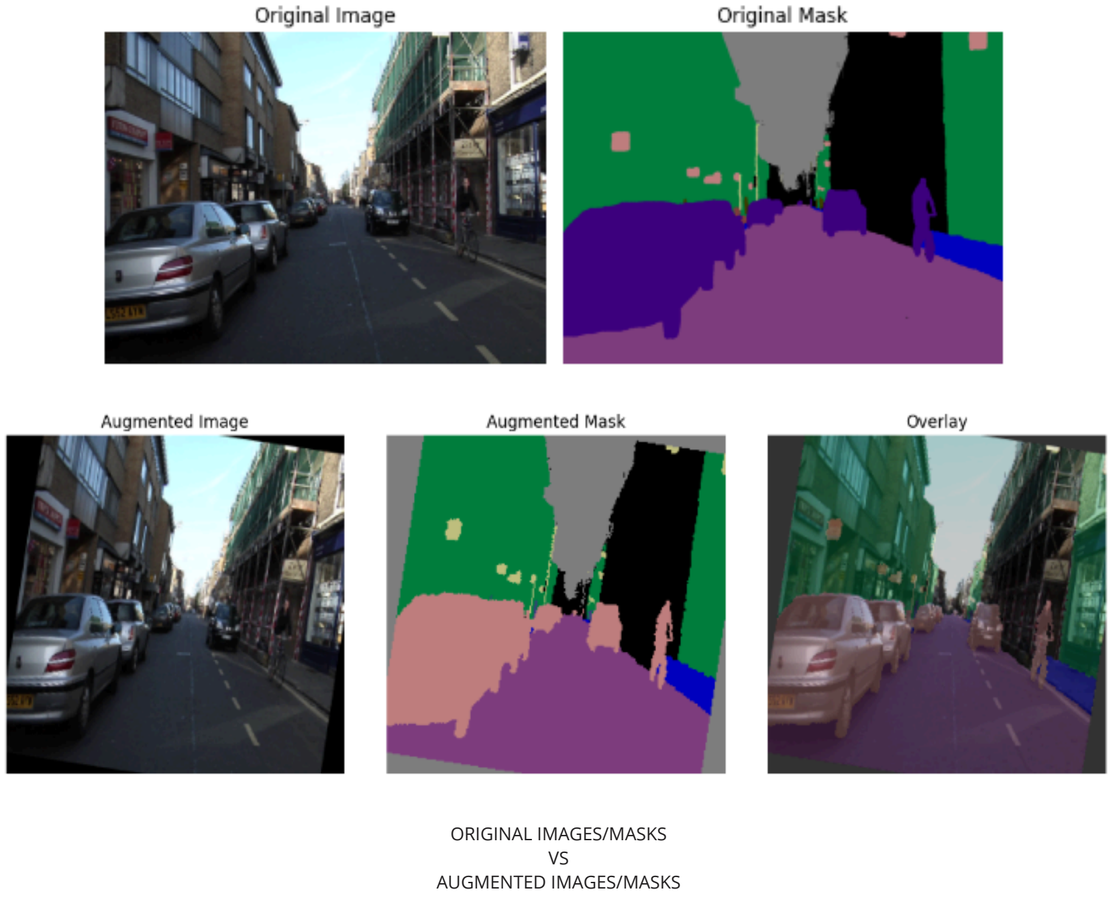
Creation of the weighted mIoU

Aggregation of the smallest classes

Choice of the best model

Test of the best model and performance assessment

Augmentation



ORIGINAL IMAGES/MASKS
VS
AUGMENTED IMAGES/MASKS

For the Augumentation we use:

For images & masks (same params)

- **Random Crop** (p=0.5)
- **Horizontal Flip** (p=0.5)
- **Shift Scale Rotate** (p=0.5)
- **Resize** (keep aspect ratio or pad)

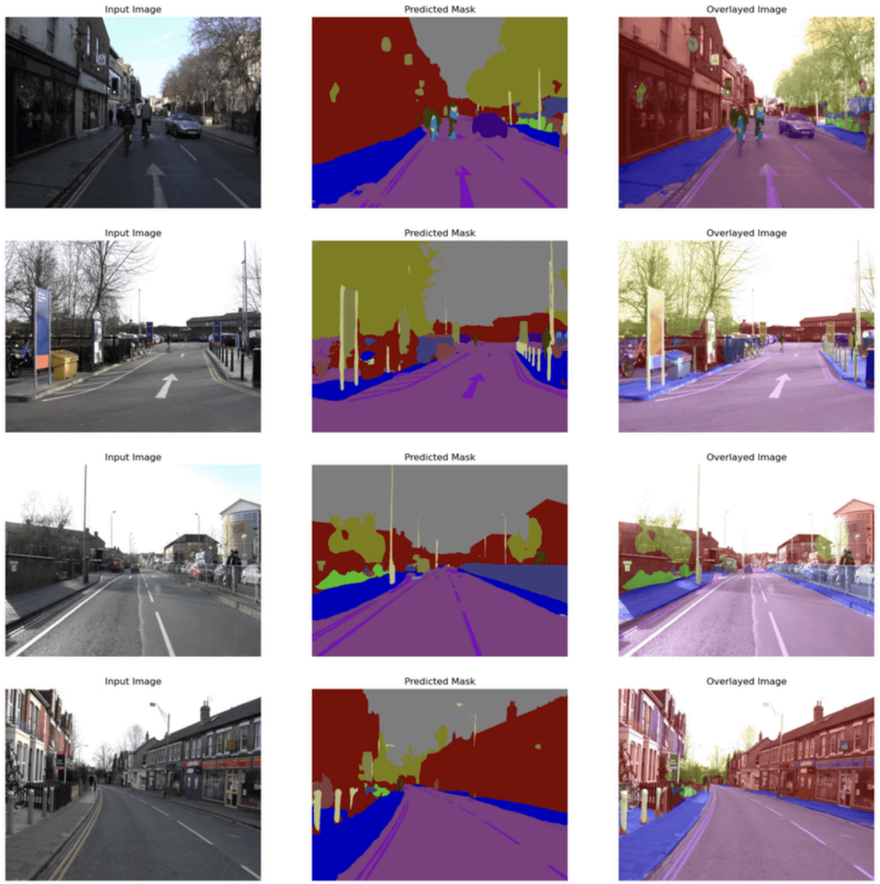
Only for images:

- **Random Brightness Contrast** (p=0.5)

Issues:

- With rotation we introduced noise due to misclassification of the background (categorized as sky).
- In the original masks, some elements (e.g. scaffolding) were not assigned to any class. Consequently, the model also predicts nothing for those pixels.

Splitting the dataset



OVERVIEW SEQUENCES

When splitting the dataset into training, validation and test sets, special care was required because the dataset consists of 4 video sequences fragmented into frames.

A random split at the frame level could place nearly identical frames from the same sequence into the same sets, causing data leakage. This would boost performance metrics without truly testing the model's generalization ability.

To prevent this, we split each video sequence **proportionally** across all three sets. This guarantees that no set is missing content from any sequence, while avoiding overlaps of near-duplicate frames.

Final **split ratio**:

- **60%** of frames for **training**.
- **20%** of frames for **validation**.
- **20%** of frames for **testing**.

First Model

```python
def mod1(pretrained=True, requires_grad=True, num_classes=11):

    model = seg.fcn_resnet50(weights='DEFAULT' if pretrained else None)
    model.classifier[4] = nn.Conv2d(512, num_classes, kernel_size=(1, 1))

    if not requires_grad:
        for param in model.parameters():
            param.requires_grad = False
        for param in model.classifier[4].parameters():
            param.requires_grad = True

    return model
```

FIRST MODEL

Base architecture: **Fully Convolutional Network** (FCN) with a **ResNet-50**, initialized with pretrained ImageNet weights (backbone only).
We discarded the pretrained classifier and replaced the last classification layer (1×1 conv) to match our number of classes. This head is trained from scratch on our dataset.

How it works: The ResNet backbone acts as a feature extractor, producing hierarchical embeddings. FCN replaces the fully connected layers with convolutional layers to output a dense prediction map (semantic segmentation).

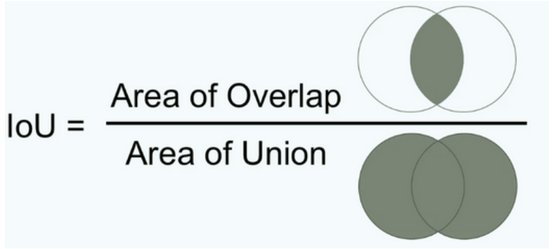
Pros:

- Lightweight, fast.
- Simple architecture, easy to train.

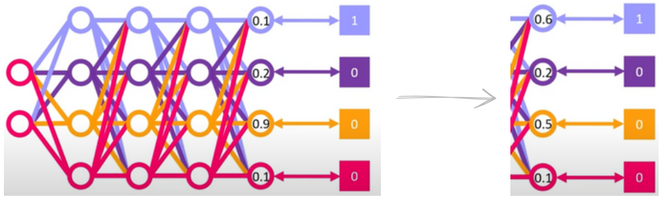
Cons:

- Limited accuracy, especially on fine object boundaries.
- Only basic upsampling → coarse segmentation masks.
- Lower accuracy compared to more advanced architectures.

Chosen Metrics



$$Pixel\ Accuracy = \frac{Correct\ Prediction}{Total\ number\ of\ pixels\ in\ the\ image}$$



Mean Intersection over Union (mIoU)

- Definition: average overlap between predicted mask and ground truth, across all classes.
- Why use it: the standard benchmark metric for semantic segmentation; gives a fair score across all classes, evaluates per-class quality.

Pixel Accuracy

- Definition: percentage of correctly classified pixels over all pixels.
- Why use it: easy to interpret, but can be biased if dataset has dominant classes (e.g. road & sky).

Weighted Cross-Entropy Loss

- Definition: Training objective function that measures the difference between predicted probabilities and the true class label for each pixel, while assigning higher weights to underrepresented classes.
- Why use it: Guides the model during training and handles class imbalance; not an evaluation metric, but indicates learning progress and convergence.

Weighted mIoU

Problem with plain mIoU:

- It gives equal weight to each class.
- If your dataset is imbalanced (e.g. road: millions of pixels, animals: very few), then rare classes dominate the metric's variability (in negative).
- Result: model may look "bad" in mIoU even if it segments dominant classes very well.

Solution → **Weighted mIoU**:

- Instead of averaging all classes equally, weight each class by its pixel frequency.

Result:

- Reflects the real distribution of the dataset.
- Rewards performance on the dominant classes (which matter most for the scene).
- Still reports per-class IoU separately (to check small objects).

Aggregation of small classes

To deal with class imbalance we applied another complementary strategies (over the Weighted mIoU): **Label aggregation**.

We **merged very small classes into larger**, semantically related ones, in order to reduce the difficulty the model has in recognizing rare categories. This reduced the number of classes **from 12 to 8** and led to an improvement in model performance.

Class	Elements	Color
Sky	Sky	
Buildings	Buildings, Bridges, Roofs	
Poles & Cones	Poles, Cones, Fences, Signs	
Roads	Roads, Horizontal Signage	
Pavements	Sidewalks, Traffic Islands	
Vegetation	Bushes, Lawns	
Vehicles	Cars, Buses, Motorbikes, Bikes	
Humans	Pedestrians, Bags, Animals	

CLASS COLOR TABLE

Best Model

Base architecture: **DeepLabV3** with a **ResNet-101 backbone** (deeper than ResNet-50, initialized with pretrained ImageNet weights for the backbone).
We discarded the pretrained classifier layers and replaced both the main and auxiliary heads (1×1 conv) to match our number of classes. These heads are trained from scratch on our dataset.

```python
def build_deeplab101(pretrained=True, requires_grad=True, num_classes=8):

    model = seg.deeplabv3_resnet101(weights='DEFAULT' if pretrained else None)
    model.classifier[-1] = nn.Conv2d(256, num_classes, kernel_size=1)
    if model.aux_classifier is not None:
        model.aux_classifier[-1] = nn.Conv2d(256, num_classes, kernel_size=1)

    if not requires_grad:
        for p in model.backbone.parameters():
            p.requires_grad = False
        for p in model.classifier.parameters():
            p.requires_grad = True
        if model.aux_classifier is not None:
            for p in model.aux_classifier.parameters():
                p.requires_grad = True

    return model
```

LAST MODEL

Key innovations over FCN:

- Uses **Atrous** (dilated) convolutions in the backbone to capture multi-scale context without losing resolution.
- Employs **ASPP** (Atrous Spatial Pyramid Pooling): multiple parallel dilated convolutions with different rates, plus image pooling → combines local and global context.
- Includes both a main classifier and an auxiliary classifier (stabilizes training).

Pros:

- Higher accuracy, especially on object boundaries and small structures.
- Captures both fine details and global scene context.

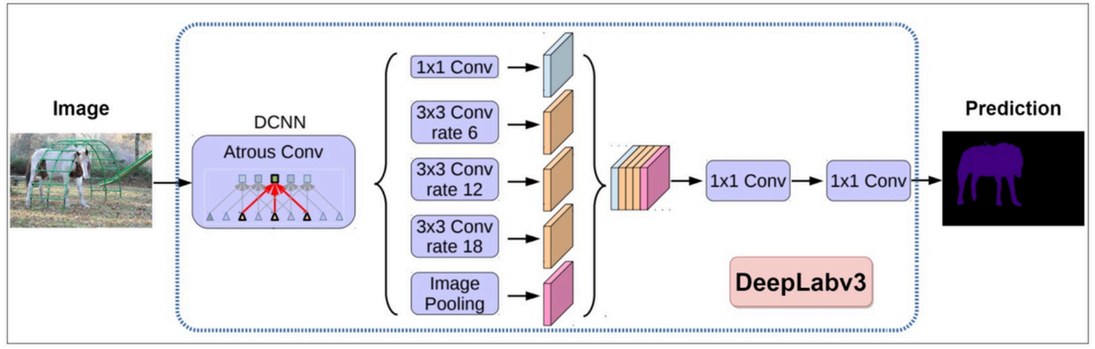
Cons:

- Heavier (ResNet-101 backbone).
- Slower inference compared to FCN-50.

Model Architecture (DeepLabV3 + ResNet-101)

Encoder (**ResNet-101** with atrous convolutions):

- Extracts feature maps while reducing spatial resolution.
- Captures both local patterns and global context.
- Pretrained on ImageNet (backbone), adapted with dilated convolutions for segmentation.



DEEPLABV3 SEMANTIC SEGMENTATION MODEL

HTTPS://LEARNOPENCV.COM/DEEPLABV3-ULTIMATE-GUIDE/

ASPP (Atrous Spatial Pyramid Pooling):

- Parallel convolutions with different dilation rates (6, 12, 18).
- Plus 1×1 convolution and image pooling branch.
- Goal: multi-scale context → recognize both large and small objects.
- ASPP enriches the features with multi-scale context before classification.

Head (**Classifier**):

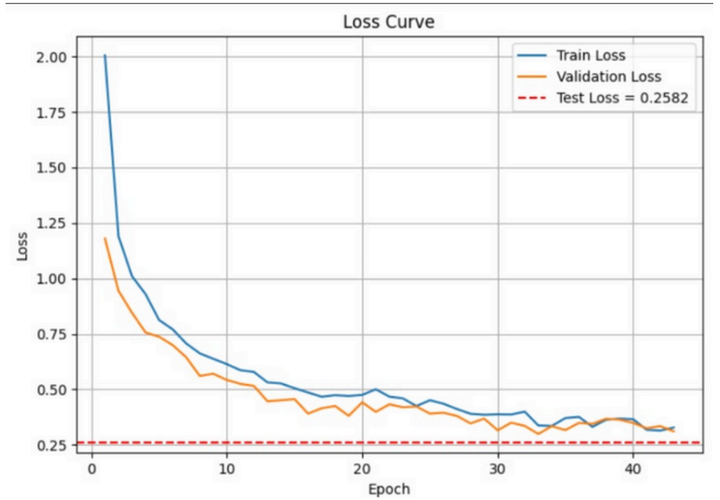
- We replaced the pretrained COCO classifier with new layers (1×1 conv) trained from scratch for our 8 classes.
- Upsampling → restores the original image resolution.

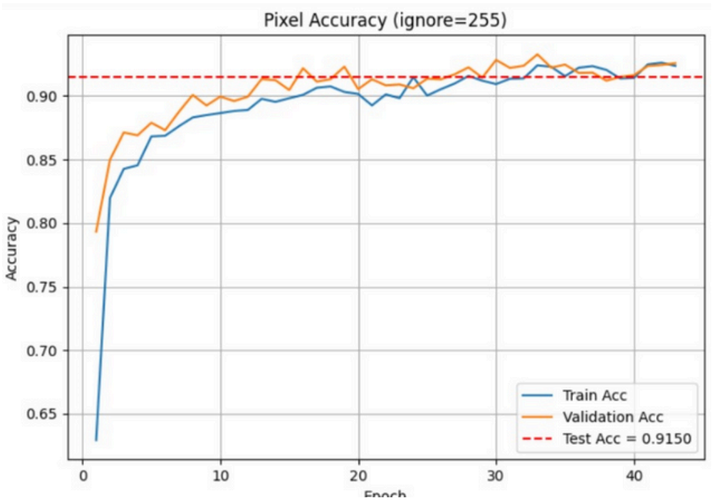
Output:

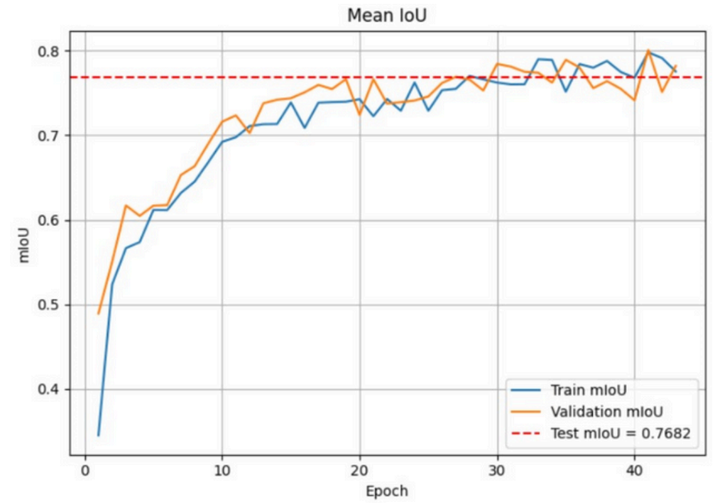
- Segmentation mask: per-pixel prediction, visualized as a color-coded map.

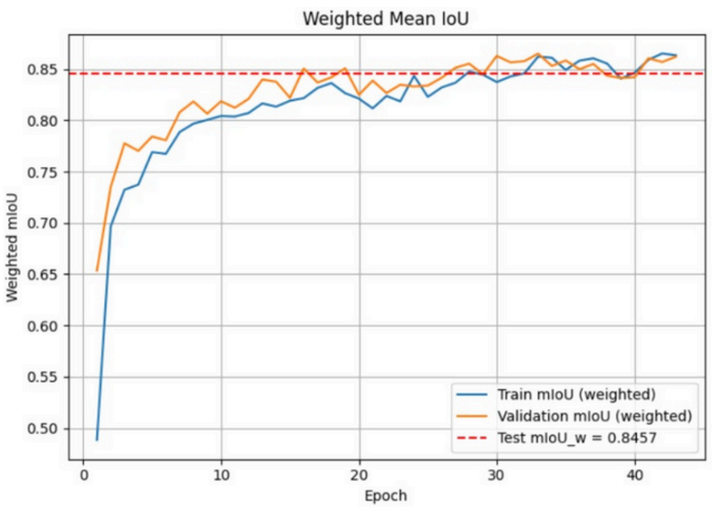
Results

- **Loss**: Both training and validation loss decreased smoothly.
- **Accuracy**: Training and validation accuracy steadily improved, stabilizing above 90%, showing good generalization.
- **mIoU**: Validation mIoU closely follows training, reaching around 0.77 on the test set.
- **Weighted mIoU**: Achieved 0.85 on test data, confirming the model performs well.









Final Consideration

We considered the following options:

Without class aggregation (*link1 google drive*)→ The mIoU stayed low because of minority classes, so we proceeded by merging the smaller classes into the larger ones.

FCN + ResNet50 vs. **DeepLabv3 + ResNet101** (*link2 google drive*) → DeepLabv3, with a deeper architecture, proves more effective and efficient in the long term.

DeepLabv3 with **fewer than 50 epochs vs. 50 epochs** → Training time can be reduced with only a minor loss in efficiency.

DeepLabv3 with **more than 50 epochs** → A possible way to improve results, but the gains are likely small compared to the extra training time.


QUESTIONS