Practical Semantic Segmentation with Deep Learning

A Step-by-Step Guide for Beginners

Book 1 | Practical Computer Vision with Deep Learning

Dr. Ali Khan | **Dr. Somaiya Khan**

March 2026

Chapter 1

Introduction to Semantic Segmentation

Computer vision systems aim to interpret and understand visual information from images. Early approaches focused primarily on image classification, where a model assigns a single label to an entire image. While useful, classification provides only a coarse understanding of visual content.

In many real-world applications, it is not sufficient to know what is present in an image. It is necessary to determine *where* objects and regions are located and how they are distributed across the entire scene.

Semantic segmentation addresses this requirement by assigning a class label to every pixel in an image. This pixel-wise prediction enables detailed scene understanding and supports a wide range of practical applications across medicine, autonomous systems, remote sensing, and industry.

1.1 What is Semantic Segmentation?

Semantic segmentation is the task of partitioning an image into regions such that each pixel is assigned a semantic label. For example, in a road scene, pixels may be classified as road, vehicle, pedestrian, building, sky, or vegetation.

Unlike object detection, which produces bounding boxes around individual objects, semantic segmentation produces a dense prediction map that describes the class of every pixel in the image. Unlike instance segmentation, which distinguishes between separate instances of the same class, semantic segmentation assigns the same label to all pixels belonging to a given category; two cars in the same image receive the same label. To better understand the differences between the major computer vision tasks, Figure 1.1 compares image classification, object detection, and semantic segmentation using the same scene.

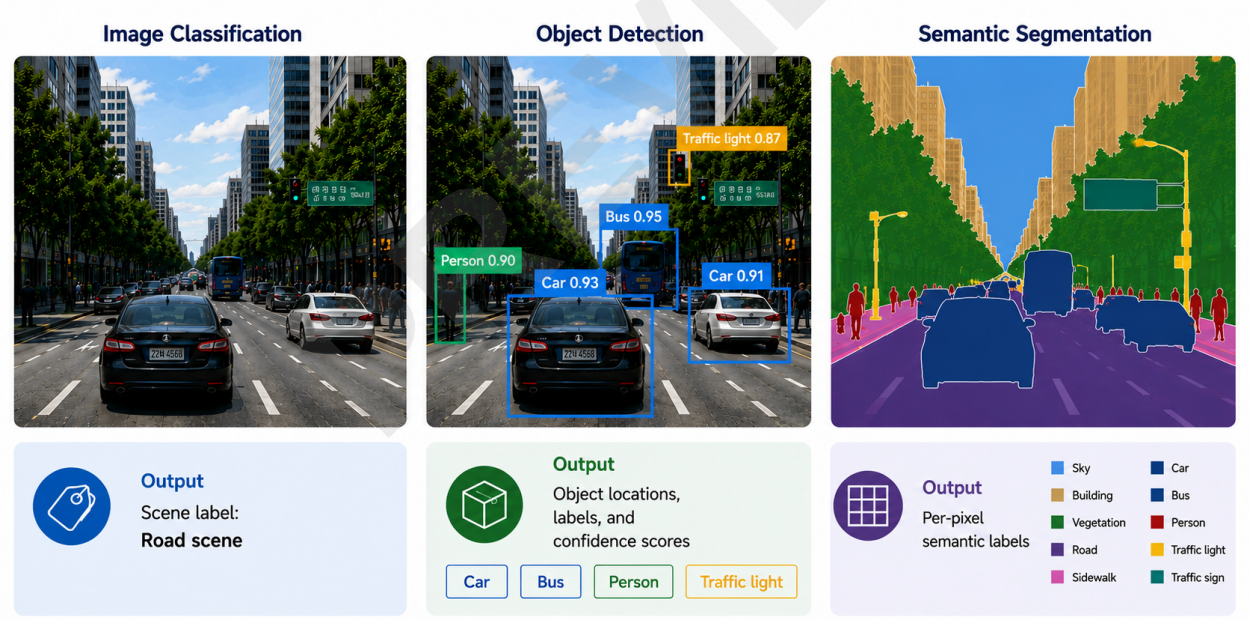

Figure 1.1: *Comparison of common computer vision tasks. Classification assigns one label per image, object detection localizes objects using bounding boxes, and semantic segmentation assigns a class label to every pixel.*

As shown in Figure 1.1, classification provides only a single label for the entire image, while object detection additionally identifies object locations through bounding boxes. Semantic segmentation goes one step further by assigning a class label to every pixel, producing a dense output map with the same spatial dimensions as the input image. This pixel-level understanding makes semantic segmentation particularly useful for applications such as autonomous driving, medical image analysis, and environmental monitoring.

1.2 From Classification to Segmentation

To understand segmentation, it is useful to consider how it extends image classification.

Chapter 2

From Classification to Pixel-wise Prediction

In the previous chapter, we introduced semantic segmentation as a pixel-wise prediction task. Unlike image classification, which assigns a single label to an entire image, segmentation requires predicting a class label for every pixel.

To understand how segmentation models are constructed, it is useful to begin with standard image classification networks and examine how they can be adapted to produce dense predictions.

2.1 Limitations of Classification Networks

Convolutional neural networks (CNNs) designed for image classification typically follow a sequence of convolutional and pooling operations, followed by fully connected layers that produce a single output prediction.

While effective for classification, this design introduces two limitations for segmentation:

- spatial resolution is reduced through pooling operations
- fully connected layers discard spatial information

As a result, classification networks are not directly suitable for pixel-wise prediction tasks.

2.2 Preserving Spatial Information

Semantic segmentation requires maintaining spatial structure throughout the network. Each output value must correspond to a specific pixel location in the input image.

To achieve this, segmentation models remove fully connected layers and replace them with convolutional layers. This allows the network to process images of arbitrary size while preserving spatial relationships.

2.3 Fully Convolutional Networks (FCNs)

Fully Convolutional Networks (FCNs) represent a key step in adapting classification networks for segmentation.

In an FCN, all layers are convolutional. The final output is a feature map rather than a single vector, enabling pixel-wise predictions.

Figure 2.1 shows the basic idea behind this transformation. By replacing fully connected layers with convolutional layers, a classification network can generate spatial outputs that preserve location information.

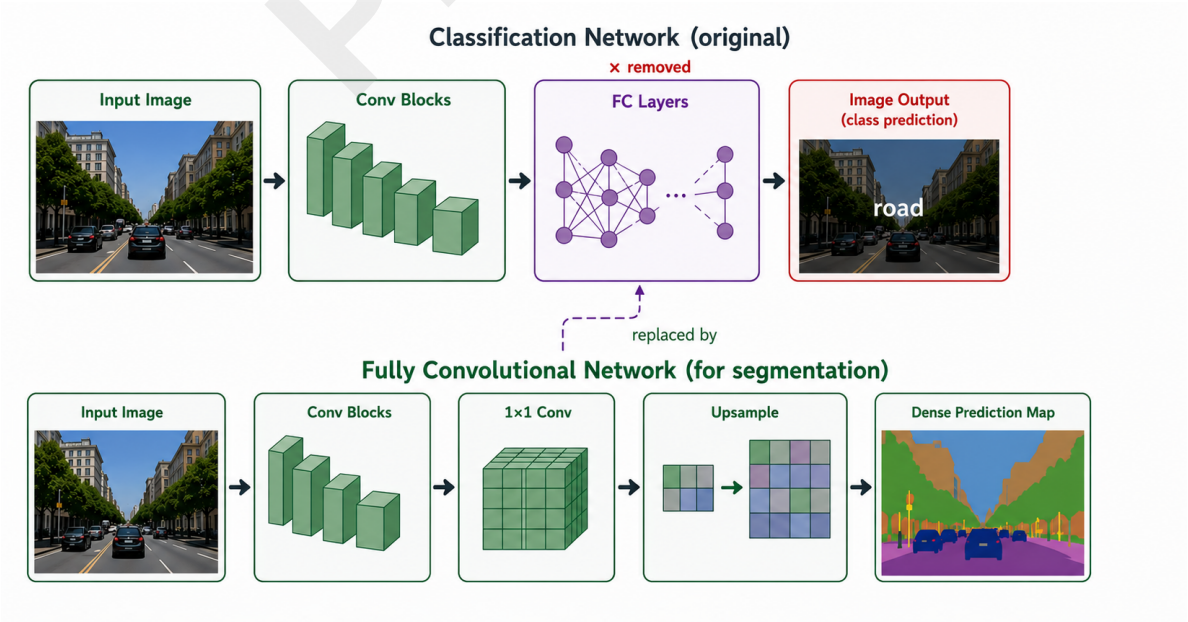

Figure 2.1: *Transformation of a classification network into a fully convolutional network for segmentation.*

Chapter 3

The U-Net Architecture

In the previous chapter, we examined how classification networks can be adapted for semantic segmentation using fully convolutional architectures. While FCNs enable pixel-wise prediction, they often produce coarse outputs because fine spatial detail is lost during the progressive downsampling of the encoder.

The U-Net architecture was introduced by Ronneberger et al. in 2015 to address this limitation directly. Originally developed for biomedical image segmentation, where precise delineation of cell boundaries and organ structures is critical, U-Net has since become the dominant baseline across virtually every segmentation domain. Its defining innovation, skip connections that link encoder and decoder at each spatial scale, remains the central idea in most modern segmentation architectures.

3.1 Overview of U-Net

U-Net follows a symmetric encoder-decoder structure. The name derives from the U-shaped appearance of the architecture when drawn as a diagram: the encoder path descends on the left, the decoder path ascends on the right, and skip connections span horizontally between them.

- the **encoder** progressively reduces spatial resolution and extracts increasingly abstract semantic features
- the **bottleneck** captures the most compressed, highest-level representation of the input
- the **decoder** progressively restores spatial resolution and produces pixel-wise predictions
- **skip connections** link each encoder stage directly to the corresponding decoder stage

Figure 3.1 shows the complete U-Net architecture and the flow of information through the encoder, bottleneck, and decoder.

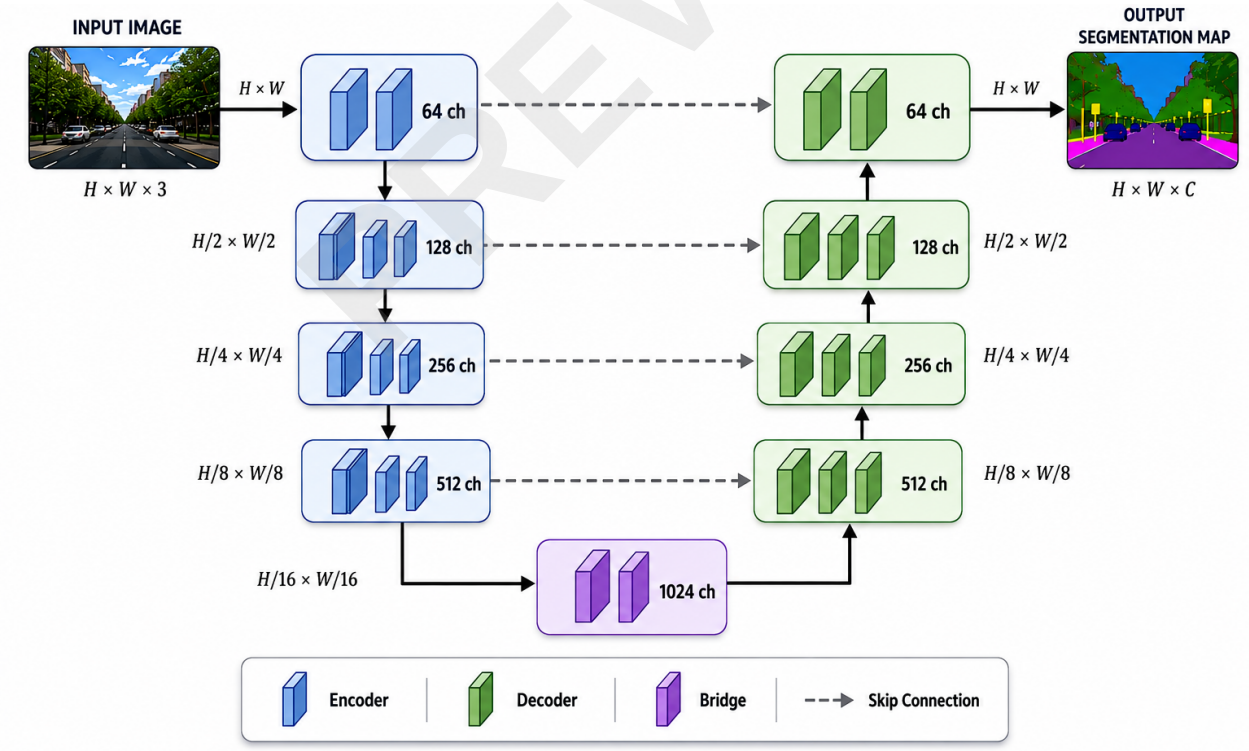

Figure 3.1: *U-Net architecture showing the encoder path (left), bridge (bottom), decoder path (right), and skip connections (horizontal arrows). Resolution labels indicate the spatial dimensions at each stage relative to the input.*

As illustrated in Figure 3.1, each horizontal level of the U corresponds to one spatial resolution. The encoder descends through four resolution levels while the decoder progressively restores the original resolution. Skip

Chapter 4

Dataset Structure and Mask Annotation

Semantic segmentation models rely on pixel-level annotations for training. Unlike classification or detection tasks, where labels are assigned to entire images or bounding boxes, segmentation requires a precise mapping between each pixel in the input image and its corresponding class label.

Understanding how segmentation datasets are structured and how masks are represented is essential for building reliable models.

4.1 Image–Mask Pairs

A segmentation dataset typically consists of pairs of images and corresponding annotation masks.

- the **image** contains the visual input
- the **mask** contains the pixel-wise labels

Each pixel in the mask represents the class of the corresponding pixel in the input image. Figure 4.1 shows a typical image–mask pair used for training a segmentation model.

Figure 4.1: *Example of an image and its corresponding segmentation mask.*

As shown in Figure 4.1, the mask has the same spatial dimensions as the input image. During training, the model learns to predict the correct class label for every pixel by comparing its output with the ground-truth mask.

4.2 Directory Structure

Segmentation datasets are commonly organized using a simple directory structure that separates images and masks.

Listing 4.1: *Typical segmentation dataset directory structure.*

```
dataset/
|-- images/
|    |-- train/
|    |-- val/
|    `-- test/
`-- masks/
     |-- train/
     |-- val/
     `-- test/
```

Chapter 5

Training a Segmentation Model

In the previous chapters, we introduced the U-Net architecture and discussed how segmentation datasets are structured. We now combine these elements to build and train a semantic segmentation model.

Training a segmentation model involves preparing data, defining the network, selecting an appropriate loss function, and optimizing the model parameters through iterative updates.

5.1 Training Pipeline Overview

A typical segmentation training pipeline consists of the following steps:

1. loading image–mask pairs
2. preprocessing inputs and labels
3. defining the model architecture
4. computing loss between predictions and ground truth
5. updating model parameters using an optimizer

This process is repeated over multiple epochs until the model converges. Figure 5.1 summarizes the major stages involved in a typical segmentation training workflow.

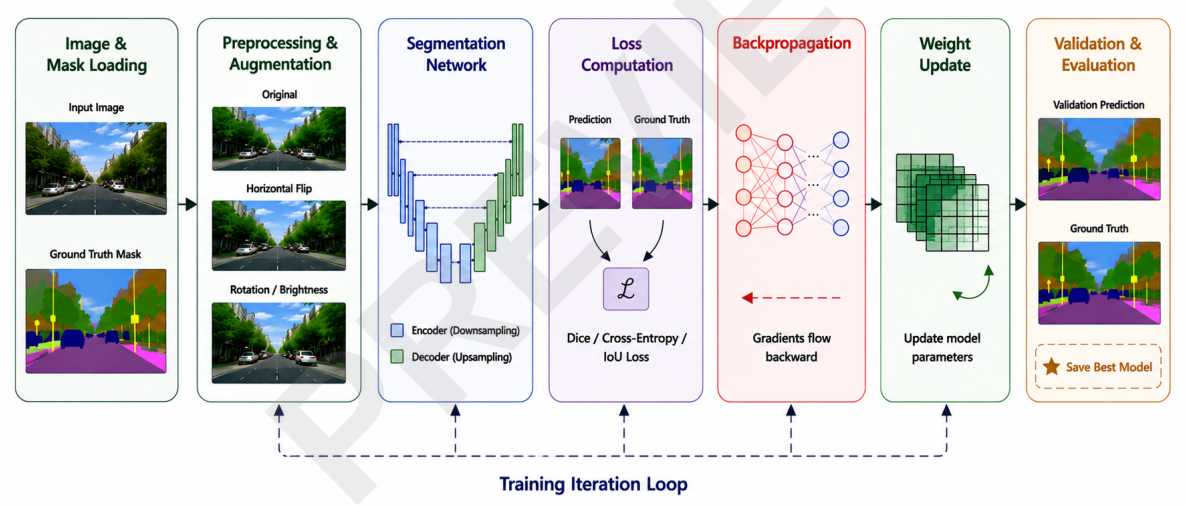

Figure 5.1: *Overview of a typical semantic segmentation training pipeline, showing data preparation, model training, loss computation, optimization, and validation stages.*

As shown in Figure 5.1, training begins with data preparation and proceeds through model training, loss computation, optimization, and validation. The following sections examine each stage in greater detail.

5.2 Data Preparation

Images and masks must be loaded in a consistent manner. Each input image must correspond exactly to its segmentation mask.

Typical preprocessing steps include:

- resizing images and masks to a fixed resolution
- normalizing image pixel values
- converting masks to integer class labels

Care must be taken to avoid altering mask labels during transformations.

Chapter 6

Loss Functions for Semantic Segmentation

Loss functions define how the difference between a model's predictions and the ground-truth annotations is measured during training. They determine what the model is optimized toward and therefore have a direct and significant impact on the quality of the learned segmentation.

In classification, cross-entropy loss applied to a single output vector is almost always sufficient. Segmentation is more demanding. Predictions are made for every pixel, datasets are frequently imbalanced, and small target regions can be overwhelmed by the dominant background class. A well-chosen loss function addresses these specific challenges. A poorly chosen one can cause the model to appear to learn while actually failing entirely on the classes that matter most.

6.1 The Pixel-wise Classification Perspective

Semantic segmentation can be framed as a classification problem applied independently to every pixel. For a multi-class problem with C classes and an image with $H \times W$ pixels, the model produces a tensor of shape (C, H, W), where each position (c, i, j) holds the predicted probability that pixel (i, j) belongs to class c.

The ground-truth mask assigns a single class index to each pixel. The learning objective is to make the predicted probability for the correct class as high as possible at every pixel location.

6.2 Cross-Entropy Loss

Cross-entropy is the standard starting point for segmentation. For a single pixel with ground-truth class c^* and predicted probabilities $\{p_c\}_{c=1}^{C}$, the cross-entropy loss is:

$$\mathcal{L}_{CE}^{pixel} = -\log(p_{c^*}) \tag{6.1}$$

The total cross-entropy loss over all pixels in a batch is the mean of the per-pixel losses:

$$\mathcal{L}_{CE} = -\frac{1}{H \cdot W} \sum_{i=1}^{H} \sum_{j=1}^{W} \log\left(p_{c^*(i,j)}(i,j)\right) \tag{6.2}$$

Cross-entropy has a clear probabilistic interpretation: minimizing it is equivalent to maximizing the log-likelihood of the correct class label at every pixel. It is well-understood, numerically stable, and produces strong gradients early in training.

6.2.1 Limitations of Cross-Entropy

The problem with cross-entropy in segmentation is that it treats every pixel equally, regardless of which class that pixel belongs to. In an image where 90% of pixels are background and 10% are the target class, the total loss is dominated by background pixels. The model learns quickly to predict background everywhere and achieves an apparently low loss, while completely ignoring the target class.

This is not a hypothetical concern. In medical imaging, the lesion or organ of interest may occupy less than 1% of the image. In satellite imagery, a specific land cover class may appear in only a handful of pixels per image. Using unweighted cross-entropy in these situations routinely produces models that score well on overall pixel accuracy while failing entirely on the classes that motivated the segmentation task.

6.2.2 Weighted Cross-Entropy

The most direct remedy is to assign a higher loss weight to underrepresented classes:

$$\mathcal{L}_{WCE} = -\frac{1}{H \cdot W} \sum_{i=1}^{H} \sum_{j=1}^{W} w_{c^*(i,j)} \cdot \log\left(p_{c^*(i,j)}(i,j)\right) \tag{6.3}$$

where w_c is the weight assigned to class c. A common choice is to set weights inversely proportional to class frequency: