TP4: Advanced Vision, Segmentation and 3D Data

DONGMO Prince Williams
Supervisor: Dr. Louis Fippo Fitime

October 19, 2025

Abstract

This fourth Practical Work (TP4) addresses advanced computer vision topics: semantic segmentation with the U-Net architecture and a first introduction to 3D volumetric data processing using Conv3D layers. The exercises emphasize implementation, metric design (Dice, IoU), and engineering practices such as experiment tracking with MLflow.

1 Introduction

TP4 develops practical skills for high-precision segmentation tasks and volumetric data handling. Students implement a simplified U-Net, custom segmentation metrics, and a toy Conv3D experiment while applying MLOps best practices for experiment tracking.

2 Learning Objectives

- Implement a U-Net for binary semantic segmentation and understand skip connections.
- Implement and compare segmentation-specific metrics (Dice coefficient and IoU).
- Integrate MLflow experiment tracking (naming convention, custom metrics, artifacts).
- Understand Conv3D operations and the engineering trade-offs for volumetric data.

3 Part 1: Segmentation and MLOps Best Practices

3.1 Semantic Segmentation and U-Net

Semantic segmentation outputs a per-pixel label map. For binary segmentation the model emits an $(H, W, 1)$ tensor of probabilities (sigmoid). The decoder path of U-Net upsamples deep feature maps and recombines them with encoder features through concatenation-based skip connections. This mechanism restores fine spatial details lost during downsampling and differs from ResNet-style additive residual connections which primarily facilitate gradient flow.

3.2 Loss functions and class imbalance

Medical segmentation often has extreme class imbalance (tiny foreground). Pixel-wise BCE is dominated by background pixels. Dice loss (1 - Dice) optimizes region overlap directly and is therefore better suited; a common practical choice is to combine BCE and Dice to balance pixel-wise and region-wise objectives.

3.3 Experiment tracking (MLflow)

We adopt a strict run naming convention to keep experiments identifiable. Suggested format: `Model-Optimizer-Loss-YYY` (example: `UNet-Adam-BCE+Dice-20251019-run1`). Custom metrics such as Dice and IoU are implemented as Keras functions and logged both via Keras history and manually to MLflow as needed. Model architecture and artifacts (weights) are saved as MLflow artifacts.

4 Part 2: Semantic Segmentation on Medical Data

4.1 U-Net Implementation

The provided implementation follows the classical U-Net pattern:

- Core conv block: Conv2D -¿ BatchNorm -¿ ReLU -¿ Conv2D -¿ BatchNorm -¿ ReLU.
- Encoder: three downsampling stages with MaxPool, doubling filters each stage.
- Bottleneck: highest-capacity conv block.
- Decoder: Conv2DTranspose upsampling, concatenation with encoder features, conv blocks.
- Output: 1x1 Conv layer with sigmoid activation for binary masks.

A code listing for the main script is available in the repository:
`src/unet_segmentation.py`

4.2 Metrics: Dice and IoU

The Dice coefficient used is:

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|}$$

IoU (Intersection over Union) is calculated as

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|}.$$

Both metrics have been implemented as Keras-compatible functions and logged to MLflow.

4.3 Toy training and example results

A fast verification run was executed on synthetic data to validate the pipeline and MLflow logging. Because this uses random data (no real medical images), the absolute metric values are not indicative of real performance. Example outputs from a 1-epoch dummy run (for pipeline validation):

- Dice (example): approximately 0.09
- IoU (example): approximately 0.045

These values reflect the randomness and sparsity of the synthetic masks and only demonstrate that the metric functions and logging work end-to-end.

5 Part 3: 3D Convolutions and Volumetric Data

5.1 Conv3D concepts

Conv3D kernels have shape (k_d, k_h, k_w) and slide over depth, height and width, capturing inter-slice context. This is important for CT/MRI where anatomical structures span multiple slices.

5.2 Engineering trade-offs

Conv3D layers are memory- and compute-heavy. To design efficient 3D models consider:

- Reducing input depth or using patch/sliding-window inference.
- Smaller kernels (3x3x3) and fewer filters.
- Hybrid 2D/3D strategies (2D per-slice encoder + shallow 3D fusion).

5.3 Demonstration: small Conv3D block

An example Conv3D block and a simulated MLflow experiment are provided in src/conv3d_experiment.py.The

6 Conclusion

TP4 provided hands-on experience with semantic segmentation (U-Net), implementation of segmentation metrics (Dice, IoU), and an introduction to Conv3D for volumetric data. The repository includes minimal working examples, MLflow logging, and a written response to theoretical questions.

Annexes

Lien du depot Github : https://github.com/DPW2005/deep_learning.git
Lien du document latex : https://www.overleaf.com/project/68d392e5a266d8f7159dd874