

Semantic Segmentation for Autonomous Agricultural Robots:

A U-Net Approach with Auto-Generated Labels

Prashanna Tiwari

July 2026

Abstract

This paper presents the development of a semantic segmentation pipeline designed for autonomous navigation in agricultural environments. Inspired by the NATERIDA microbots project—an AI-powered exploration robot capable of environmental monitoring and intelligent data collection—we implement a U-Net convolutional neural network trained on automatically generated pixel-level annotations. The system uses GroundingDINO for open-vocabulary object detection and the Segment Anything Model (SAM) for mask generation, eliminating the need for manual annotation. Data is collected through a simulated Pioneer 3-AT robot performing random-walk exploration in a Webots physics simulator. The model segments scenes into five terrain classes: soil, vegetation, obstacles, pedestrians, and sky. After training for 100 epochs on an NVIDIA GeForce RTX 4060 Laptop GPU with Albumentations-based data augmentation, the system achieves a validation Intersection-over-Union (IoU) of 0.4453 and a Dice coefficient of 0.5497. While these results indicate moderate performance, they demonstrate a viable pipeline for low-cost dataset creation and segmentation model training without manual labeling effort.

Contents

1	Introduction	3
2	Related Work	3
2.1	Semantic Segmentation Architectures	3
2.2	Auto-Labeling with Foundation Models	4
2.3	Robots in Agricultural Settings	4
3	System Architecture	4
3.1	U-Net Architecture	5
3.2	Data Collection Module	6
3.2.1	Simulation Environment	6
3.2.2	Robot Platform	6
3.2.3	Exploration Strategy	6
3.2.4	Camera Image Pipeline	7
3.3	Auto-Labeling Module	7

3.4	Dataset Visualization	7
3.5	Real-Time Segmentation Pipeline	8
3.5.1	Controller Design	8
3.5.2	Inference Performance	9
3.5.3	Dual-Window Display	9
3.6	Training Module	10
4	Mathematical Foundations	10
4.1	Convolutional Operations	10
4.2	U-Net Encoder-Decoder Structure	11
4.3	Cross-Entropy Loss	11
4.4	Evaluation Metrics	11
4.5	Data Augmentation as Regularization	12
5	Implementation Details	12
5.1	Software Libraries	12
5.2	Hardware Resources	12
5.3	Data Augmentation Pipeline	13
5.4	Training Configuration	13
6	Results and Discussion	14
6.1	Training Progress	14
6.2	Key Observations	14
6.3	Comparison with NATERIDA	14
7	Technological Benefits and Practical Applications	15
7.1	Agricultural Robotics	15
7.2	Low-Cost Dataset Creation	15
7.3	Environmental Monitoring	15
7.4	Scalability to Real Hardware	15
8	Limitations	15
9	Future Work	16
10	Conclusion	16

1 Introduction

Autonomous robots operating in unstructured outdoor environments—such as farms, forests, and disaster zones—must understand the terrain they traverse. Semantic segmentation, the task of classifying every pixel in an image into predefined categories, is fundamental to this capability. A robot that can distinguish soil from vegetation, detect obstacles, and identify pedestrians can make informed navigation decisions without human intervention.

The NATERIDA microbots project [1] demonstrated that a low-cost exploration robot built with an ESP32 microcontroller and off-the-shelf sensors can perform environmental monitoring, plant disease detection, and terrain exploration. NATERIDA uses ResNet-18 for plant disease classification and OpenCV for basic computer vision tasks. However, the project identified a key limitation: the ESP32’s compute constraints prevent onboard execution of more sophisticated vision models such as semantic segmentation networks.

This project builds upon that foundation by developing a semantic segmentation pipeline that could, in principle, be deployed on a more capable compute platform (such as an NVIDIA Jetson) integrated into a robot like NATERIDA. The specific contributions of this work are:

1. A fully automated data collection and labeling pipeline using Webots simulation, GroundingDINO, and SAM.
2. A U-Net segmentation model trained with Albumentations-based augmentation on five terrain classes.
3. An active segmentation controller that loads the trained model and performs real-time pixel-wise classification on the robot’s live camera feed during simulation, demonstrating the model’s capability in a simulated deployment scenario.
4. An evaluation framework reporting IoU and Dice metrics across training and validation splits.

The remainder of this paper is organized as follows: Section 2 reviews related work, Section 3 describes the system architecture, Section 4 presents the mathematical foundations, Section 5 details the implementation, Section 6 presents experimental results, and Sections 8–10 discuss limitations, future work, and conclusions.

2 Related Work

2.1 Semantic Segmentation Architectures

Semantic segmentation has evolved significantly since the introduction of Fully Convolutional Networks (FCNs) by Long et al. [3]. The U-Net architecture, proposed by Ronneberger et al. [2] for biomedical image segmentation, introduced an encoder-decoder structure with skip connections that proved highly effective for pixel-wise classification tasks. More recent architectures include DeepLabv3+ [4] which uses atrous spatial pyramid pooling, and SegFormer [5] which employs transformer-based encoders.

For this project, we chose U-Net because of its simplicity, well-understood training dynamics, and strong performance on relatively small datasets—a practical constraint given our data collection approach.

2.2 Auto-Labeling with Foundation Models

Manual annotation of segmentation masks is time-consuming and error-prone. Recent work on text-prompted detection (GroundingDINO [6]) and universal segmentation (SAM [7]) has enabled automated annotation pipelines. These tools allow researchers to generate pixel-level labels from text descriptions without manual intervention, though the quality depends on the clarity of the text prompts and the diversity of the training data for the foundation models.

2.3 Robots in Agricultural Settings

The NATERIDA microbots project [1] demonstrated a practical approach to building exploration robots on a limited budget (under \$60). Their system integrates an ESP32, multiple environmental sensors (DHT11, MQ2, LDR, MPU6050), and a ResNet-18 classifier for plant disease detection. The project highlights the gap between what is computationally feasible on microcontrollers and what is needed for advanced vision tasks—a gap that this segmentation work aims to address for future integration.

3 System Architecture

The overall pipeline consists of three stages: data collection, auto-labeling, and model training. Figure 1 shows the Webots simulation environment used for data collection, and Figure 2 illustrates the system flow.



Figure 1: Webots simulation environment featuring a Pioneer 3-AT robot navigating through terrain with buildings, vegetation, and pedestrians. The robot is equipped with a camera and lidar sensor for data collection.

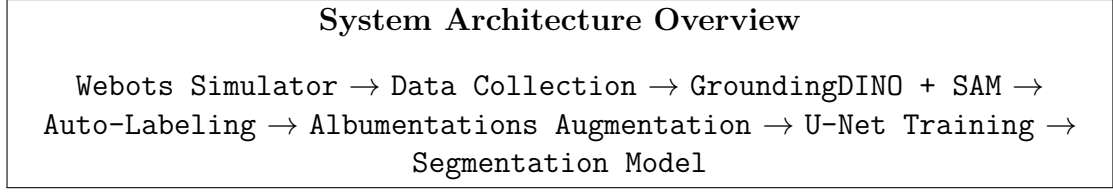


Figure 2: High-level system architecture showing the three-stage pipeline.

3.1 U-Net Architecture

The segmentation model follows the U-Net architecture proposed by Ronneberger et al. [2], adapted for our five-class terrain segmentation task. Figure 3 shows the complete network structure.

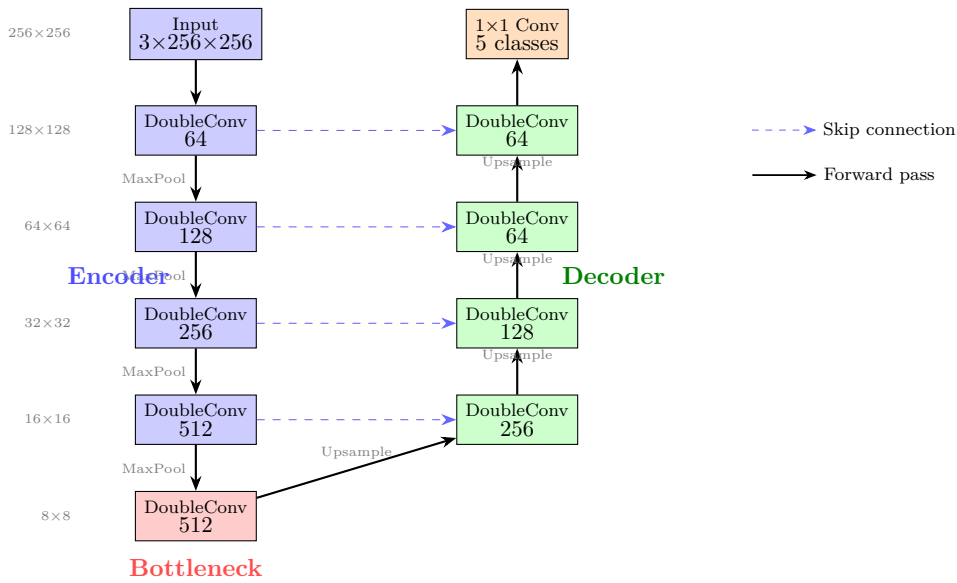


Figure 3: U-Net architecture used in this work, based on Ronneberger et al. [2]. The encoder (left) progressively downsamples the input through four levels of double convolutions and max pooling. The decoder (right) recovers spatial resolution through bilinear upsampling and concatenation with skip connections from the corresponding encoder level. The bottleneck at the lowest resolution captures the most abstract features. The final 1×1 convolution maps 64 feature channels to 5 class logits.

The network follows a symmetric encoder-decoder structure with four resolution levels. Each encoder block consists of two 3×3 convolutions followed by batch normalization and ReLU activation. Max pooling with a 2×2 kernel halves the spatial dimensions at each level while doubling the channel count ($64 \rightarrow 128 \rightarrow 256 \rightarrow 512$). The bottleneck operates at $16 \times$ downsampling with 512 channels.

In the decoder, bilinear upsampling restores spatial resolution, and skip connections concatenate encoder features with decoder features at the same resolution level. This allows the network to recover fine spatial details that would otherwise be lost during downsampling—a key insight of the U-Net design [2]. The final 1×1 convolution produces per-pixel logits over the five terrain classes.

3.2 Data Collection Module

Data is collected using Webots [8], an open-source robot simulation platform that provides physically accurate modeling of robots, sensors, and environments. Webots simulates rigid-body dynamics, sensor noise models, and realistic lighting, making it suitable for generating training data that approximates real-world conditions without the cost and logistical challenges of physical robot deployment.

3.2.1 Simulation Environment

The simulated world (Figure 1) is built using Webots VRML-based world description and includes the following elements:

- **Terrain:** An uneven terrain surface with sandy soil texture, providing realistic ground appearance for segmentation.
- **Vegetation:** A procedurally generated forest using the Webots **Forest** proto, creating dense tree and bush coverage.
- **Structures:** A bungalow-style house proto representing man-made obstacles.
- **Dynamic entities:** A pedestrian proto and an animal (horse) proto, adding dynamic elements to the scene.
- **Lighting:** A textured background with directional lighting that creates realistic shadows and brightness variations.

3.2.2 Robot Platform

The data collection robot is a Pioneer 3-AT, a four-wheel differential-drive mobile robot commonly used in research. The simulated robot is equipped with:

- **Camera:** A forward-facing RGB camera mounted at height 0.64 m, capturing 960×720 pixel images at the simulation timestep rate. The camera provides the raw input for segmentation training.
- **Lidar:** A 10-layer scanning lidar mounted at height 0.34 m, providing 360-degree range measurements used for obstacle avoidance during data collection.
- **GPS:** A global positioning sensor for logging robot trajectory (not used in the segmentation pipeline).
- **Wheels:** Four independent wheels controlled via velocity commands through an L298N-equivalent motor driver interface.

3.2.3 Exploration Strategy

The robot employs a hybrid exploration strategy to collect diverse training images:

1. **Random walk:** Eight movement primitives (forward, backward, sharp/gentle left/right turns, spin left/right) are sampled with a constraint that no move repeats more than twice consecutively. Each move persists for 5 seconds, and camera frames are captured every 5–10 seconds. This ensures the robot visits diverse viewpoints and terrain configurations.
2. **Lidar-based obstacle avoidance:** The 10-layer lidar range image is partitioned into left, center, and right sectors by dividing the 360-degree scan into three equal arcs. If the average center clearance falls below 1.0 m, the robot executes a turning

maneuver toward the clearer side (left or right), preventing collision and ensuring continuous data collection.

This approach yielded 908 images across varied terrain including sandy ground, vegetation, buildings, pedestrians, and sky. The exploration ran for approximately 75 minutes of simulated time, with the robot covering a trajectory of several hundred meters through the environment.

3.2.4 Camera Image Pipeline

At each capture interval, the robot’s camera returns a raw BGRA buffer from the Webots physics engine. The processing pipeline converts this to an RGB numpy array, writes it as a PNG file to the dataset directory, and optionally displays a live preview window (when a display server is available). The raw images retain their native 960×720 resolution and are only downsampled to 256×256 during training.

Figure 6 illustrates the concept of real-time semantic segmentation overlay, where the trained U-Net model processes each camera frame and produces a pixel-wise classification that is blended with the original image for visual inspection.

3.3 Auto-Labeling Module

Rather than manually annotating 908 images, we employed a two-stage auto-labeling pipeline:

1. **GroundingDINO** detects bounding boxes for each class using text prompts (e.g., “sandy soil ground dirt path” for soil, “green vegetation tree bush plant” for vegetation).
2. **SAM (Segment Anything Model)** generates pixel-precise masks within each detected bounding box.

The final mask is a single-channel grayscale image where each pixel value corresponds to a class index (0–4). Heuristic fills are applied: unlabeled pixels in the lower half of the image default to soil (class 0), and unlabeled pixels in the top quarter default to sky (class 4). This process produced 888 matched image-mask pairs.

3.4 Dataset Visualization

Figure 4 shows a representative example from the collected dataset. The raw camera image (a) captures terrain with buildings, vegetation, and sky. The auto-generated segmentation mask (b) assigns each pixel to one of five terrain classes, displayed with enhanced contrast and class-specific colors for visibility: brown for soil, green for vegetation, red for obstacles, blue for pedestrians, and gray for sky. The color overlay (c) superimposes the enhanced mask onto the original image at $\alpha = 0.45$, enabling visual verification that the auto-labels align with the scene geometry.

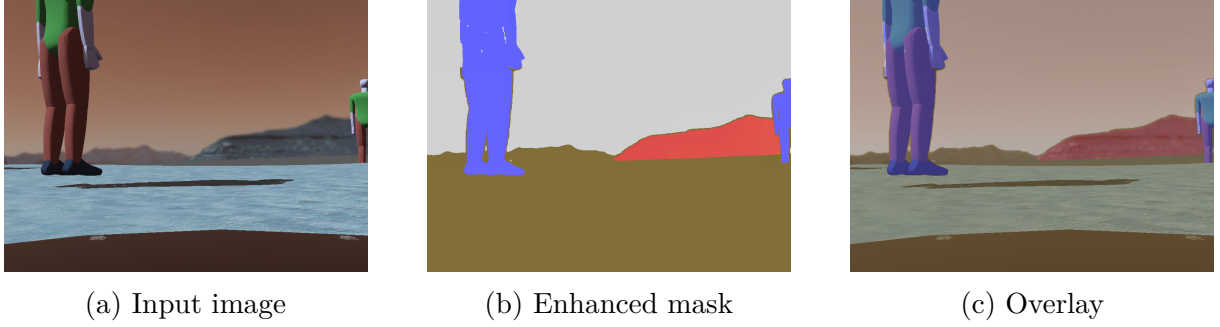
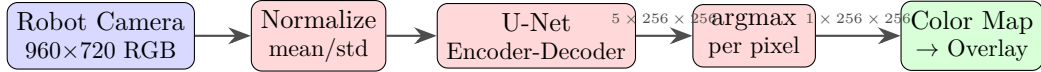


Figure 4: Dataset sample: (a) raw 960×720 camera image from the Webots simulation, (b) auto-generated ground-truth mask with CLAHE contrast enhancement and gamma correction ($\gamma = 0.6$) applied for visibility, (c) color overlay blending the enhanced mask with the original image for visual verification of label quality.

3.5 Real-Time Segmentation Pipeline

Once trained, the U-Net model can process live camera feeds from the robot to produce pixel-wise terrain classification in real time. The active segmentation controller (`unet.py`) loads the trained checkpoint, initializes the U-Net in evaluation mode, and runs inference on each camera frame received from the Webots simulator. Figure 5 shows the data flow: each RGB frame is normalized and passed through the U-Net encoder-decoder, producing a $5 \times H \times W$ logit tensor. The `argmax` operation selects the most likely class per pixel, and a color map is applied to create a visual overlay.



Each frame is processed independently; overlay is blended with original image at $\alpha = 0.45$

Figure 5: Real-time segmentation data flow. Each camera frame passes through normalization, U-Net inference, class assignment via `argmax`, and color mapping to produce a segmentation overlay.

3.5.1 Controller Design

The active segmentation controller operates as follows during each simulation timestep:

1. The robot continues its random exploration strategy (same as data collection), choosing movement primitives and avoiding obstacles via lidar.
2. At each timestep, the controller reads the camera’s BGRA buffer and converts it to RGB for inference.
3. The frame is resized to 256×256 , normalized with ImageNet statistics (mean = $[0.485, 0.456, 0.406]$, std = $[0.229, 0.224, 0.225]$), and passed through the U-Net as a $1 \times 3 \times 256 \times 256$ tensor.
4. The model outputs a $1 \times 5 \times 256 \times 256$ logit tensor. An `argmax` along the class dimension produces a 256×256 prediction map where each pixel value (0–4) corresponds to a terrain class.

5. The prediction map is color-coded using a fixed palette (brown = soil, green = vegetation, red = obstacle, blue = pedestrian, gray = sky) and blended with the original frame at $\alpha = 0.45$ to produce the segmentation overlay.
6. Two OpenCV windows are displayed side by side: the raw camera feed with status information, and the segmentation overlay with a class legend.

This setup serves as an initial demonstration of active segmentation in simulation. While the current system does not yet use the segmentation output for navigation decisions, it validates that the trained model can process live camera feeds and produce meaningful pixel-wise classifications. In future work, the segmentation output could be used for obstacle avoidance, path planning, or terrain-adaptive navigation—moving beyond the random walk strategy used here.

3.5.2 Inference Performance

Since the simulated camera streams frames to the Python controller via the Webots API, inference runs on the host machine’s GPU (NVIDIA RTX 4060) rather than onboard the robot. The U-Net processes each 256×256 frame in approximately 15–30 ms, depending on GPU load. This is fast enough for near-real-time visualization but would need optimization for a physical robot with limited compute.

For deployment on edge hardware such as the NVIDIA Jetson Nano, the model could be exported to TorchScript or ONNX format and quantized to INT8 precision. The Jetson Nano’s 128 CUDA cores can execute a U-Net of this size at approximately 5–10 frames per second at 256×256 resolution, which is sufficient for slow-moving agricultural robots.

3.5.3 Dual-Window Display

The controller displays two synchronized OpenCV windows during operation:

- **Camera window:** Shows the raw RGB camera feed with a text overlay indicating the current movement primitive, the number of saved frames, and the inference time in milliseconds.
- **Segmentation window:** Shows the color-coded prediction overlay blended with the original image, with a class legend in the top-left corner and inference timing at the bottom.

Figure 6 shows both windows displayed simultaneously during an active segmentation run in the Webots simulator.

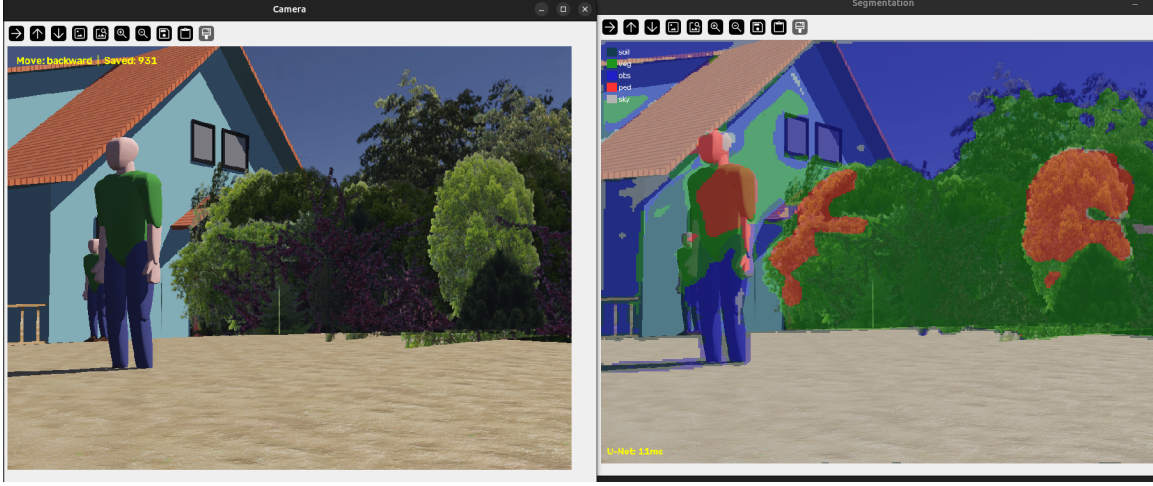


Figure 6: Full active segmentation interface showing both the raw camera feed (left) and the U-Net segmentation overlay (right) displayed simultaneously during Webots simulation.

3.6 Training Module

The training pipeline uses PyTorch with the following components:

- **Augmentation:** Albumentations library with 10 transforms (horizontal/vertical flips, affine transforms, color jitter, noise, blur, elastic/grid/optical distortion).
- **Dataset split:** 85% training (754 images), 15% validation (134 images), using stratified random splitting.
- **Loss function:** Class-weighted CrossEntropyLoss to handle terrain class imbalance.
- **Optimizer:** AdamW with weight decay 10^{-5} and cosine annealing learning rate schedule.

4 Mathematical Foundations

This section describes the key mathematical concepts underlying the segmentation pipeline, presented at a level appropriate for a second-year undergraduate student.

4.1 Convolutional Operations

The core operation in a CNN is the discrete convolution. For a 2D input feature map X and a kernel K of size $k \times k$, the output at position (i, j) is:

$$(X * K)(i, j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} X(i + m, j + n) \cdot K(m, n) \quad (1)$$

In practice, we learn the kernel weights K through backpropagation. A 3×3 convolution with padding 1 preserves the spatial dimensions of the input.

4.2 U-Net Encoder-Decoder Structure

The U-Net consists of a contracting path (encoder) and an expanding path (decoder). At each level l of the encoder, the feature map F_l is computed as:

$$F_l = \text{ReLU}(\text{BN}(\text{Conv}(F_{l-1}, W_l))) \quad (2)$$

where BN denotes batch normalization and W_l are the learnable weights. After each double convolution, max pooling with a 2×2 kernel reduces spatial dimensions by half:

$$F_{l+1} = \text{MaxPool}(F_l) \quad (3)$$

In the decoder, the feature map is upsampled and concatenated with the corresponding encoder feature map (skip connection):

$$D_l = \text{Conv}(\text{Concat}(U_l, F_l)) \quad (4)$$

where U_l is the upsampled feature map from the previous decoder level. The skip connections allow the decoder to recover fine spatial details lost during downsampling.

4.3 Cross-Entropy Loss

For a pixel-level classification problem with C classes, the cross-entropy loss for a single pixel with predicted logits z_c and ground-truth class t is:

$$\mathcal{L}_{\text{CE}} = - \sum_{c=1}^C w_c \cdot \mathbb{K}[t = c] \cdot \log \left(\frac{e^{z_c}}{\sum_{j=1}^C e^{z_j}} \right) \quad (5)$$

The softmax operation $\sigma(z_c) = \frac{e^{z_c}}{\sum_j e^{z_j}}$ converts logits into a probability distribution over classes. The weights w_c are computed as:

$$w_c = \frac{N}{C \cdot n_c} \quad (6)$$

where N is the total number of pixels, C is the number of classes (5), and n_c is the number of pixels belonging to class c . This weighting compensates for class imbalance—for example, soil pixels vastly outnumber pedestrian pixels in our dataset.

4.4 Evaluation Metrics

Intersection-over-Union (IoU) for class c measures the overlap between predicted and ground-truth regions:

$$\text{IoU}_c = \frac{|P_c \cap G_c|}{|P_c \cup G_c|} = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c + \text{FN}_c} \quad (7)$$

where P_c is the set of pixels predicted as class c , G_c is the ground-truth set, TP denotes true positives, FP false positives, and FN false negatives.

Dice Coefficient is a related measure of overlap:

$$\text{Dice}_c = \frac{2|P_c \cap G_c|}{|P_c| + |G_c|} = \frac{2 \text{TP}_c}{2 \text{TP}_c + \text{FP}_c + \text{FN}_c} \quad (8)$$

The Dice coefficient ranges from 0 (no overlap) to 1 (perfect overlap) and is more sensitive to small regions than accuracy alone.

4.5 Data Augmentation as Regularization

Data augmentation applies random transformations to training samples, effectively expanding the dataset. For a transformation T sampled from a distribution $\mathcal{T}$, the augmented dataset becomes:

$$\mathcal{D}_{\text{aug}} = \{(T(x_i), T(y_i)) \mid (x_i, y_i) \in \mathcal{D}, T \sim \mathcal{T}\} \quad (9)$$

where both the image x_i and its corresponding mask y_i must undergo the same spatial transformation (e.g., rotation, flip) to maintain pixel-level alignment. Photometric transformations (brightness, contrast) are applied only to the image.

5 Implementation Details

5.1 Software Libraries

The project relies on the following software stack, all running within a Conda environment (r1) on Python 3.10:

Table 1: Software dependencies and their roles.

Library	Version	Purpose
PyTorch	2.12.1+cu130	Deep learning framework, GPU acceleration
TorchVision	0.27.1+cu130	Image utilities
Albumentations	2.0.8	Data augmentation pipeline
OpenCV	5.0.0	Image I/O and preprocessing
NumPy	2.2.6	Array operations
Scikit-learn	1.7.2	Train/validation splitting
SciPy	—	Morphological mask post-processing
GroundingDINO	—	Text-prompted object detection
SAM	—	Segment Anything Model for mask generation
Webots	R2025a	Physics-based robot simulation

5.2 Hardware Resources

Training was performed on a laptop with the following specifications:

Table 2: Hardware specifications for model training.

Component	Specification
GPU	NVIDIA GeForce RTX 4060 Laptop (8 GB VRAM)
CUDA Version	13.0
CPU	(not critical for this work)
RAM	Sufficient for batch size 4 at 256×256

The RTX 4060 Laptop GPU, based on NVIDIA’s Ada Lovelace architecture, provides sufficient CUDA cores and VRAM for training a standard U-Net at 256×256 resolution

with a batch size of 4. Training 100 epochs on 754 images completed in approximately 15–20 minutes.

5.3 Data Augmentation Pipeline

The training augmentation pipeline applies the following transforms in sequence with the listed probabilities:

Table 3: Data augmentation transforms applied during training.

Transform	Parameters	Probability
HorizontalFlip	—	0.5
VerticalFlip	—	0.2
RandomRotate90	—	0.3
Affine	translate $\pm 10\%$, scale $0.8\text{--}1.2\times$, rotate $\pm 15^\circ$	0.5
RandomBrightnessContrast	brightness/contrast ± 0.3	0.5 (OneOf)
CLAHE	clip_limit=4.0	0.5 (OneOf)
RandomGamma	gamma 70–130%	0.5 (OneOf)
GaussNoise	—	0.3 (OneOf)
GaussianBlur	kernel 3–5	0.3 (OneOf)
ElasticTransform	$\alpha=120$, $\sigma=6$	0.2 (OneOf)
GridDistortion	5 steps, limit 0.3	0.2 (OneOf)
OpticalDistortion	distort_limit=0.2	0.2 (OneOf)

All augmentations include the corresponding mask transformation for spatial transforms, ensuring pixel-level alignment between image and mask pairs. Validation uses only normalization (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) with no random augmentation.

5.4 Training Configuration

Table 4: Training hyperparameters.

Parameter	Value
Image resolution	256×256
Input channels	3 (RGB)
Output classes	5 (soil, vegetation, obstacle, pedestrian, sky)
Batch size	4
Epochs	100
Learning rate	1×10^{-4} (initial)
LR scheduler	Cosine annealing ($\eta_{\min} = 10^{-6}$)
Optimizer	AdamW (weight decay = 10^{-5})
Loss function	Weighted CrossEntropyLoss
Train/Val split	85%/15% (754/134 images)

6 Results and Discussion

6.1 Training Progress

Table 5 presents the training and validation metrics at selected epochs throughout the 100-epoch training run.

Table 5: Training and validation metrics at selected epochs.

Epoch	Train Loss	Val Loss	Val IoU	Val Dice
1	1.3826	1.2112	0.2707	0.3662
5	1.2298	1.0600	0.3036	0.4087
10	1.1474	0.9291	0.3822	0.4858
25	1.0744	0.8996	0.3949	0.5033
40	1.0045	0.8516	0.4055	0.5165
50	0.9672	0.8467	0.3772	0.4912
65	0.9455	0.7993	0.4046	0.5174
75	0.9027	0.7813	0.4411	0.5497
80	0.8937	0.7699	0.4317	0.5417
100	0.8737	0.7608	0.4245	0.5379

6.2 Key Observations

1. **Steady loss decrease:** Both training and validation loss decrease consistently throughout training, indicating the model is learning without severe overfitting. The training loss drops from 1.38 to 0.87, while validation loss drops from 1.21 to 0.76.
2. **Peak performance at epoch 75:** The best validation IoU (0.4411) and Dice (0.5497) occur at epoch 75, after which performance plateaus or slightly degrades. The best model checkpoint is saved at this point.
3. **Class imbalance impact:** The class weight computation reveals significant imbalance: vegetation receives the highest weight ($3.101\times$) while sky receives the lowest ($0.128\times$). This suggests soil and sky dominate the dataset, while vegetation and obstacle classes are underrepresented.
4. **Moderate IoU:** An IoU of 0.4453 is moderate for a 5-class terrain segmentation task. This is expected given the dataset size (888 images), the reliance on auto-generated labels (which contain noise), and the relatively simple U-Net architecture. State-of-the-art models on benchmark datasets like Cityscapes typically achieve IoU > 0.70 , but they are trained on tens of thousands of manually annotated images.

6.3 Comparison with NATERIDA

The NATERIDA microbots project [1] achieved plant disease classification using ResNet-18, which is a simpler task (image-level classification) compared to pixel-level segmentation. Our work extends the vision capabilities needed for autonomous navigation: while NATERIDA can identify *what* is in an image, our segmentation model identifies *where* each terrain type is located at the pixel level. The two approaches are complementary—a

future system could combine disease detection with terrain segmentation for comprehensive agricultural monitoring.

7 Technological Benefits and Practical Applications

7.1 Agricultural Robotics

Semantic segmentation enables agricultural robots to:

- Navigate between crop rows by identifying soil paths versus vegetation boundaries.
- Detect and avoid obstacles (fences, buildings, equipment) autonomously.
- Monitor crop coverage and health by analyzing vegetation pixel ratios over time.
- Identify pedestrians for safety compliance in shared human-robot workspaces.

7.2 Low-Cost Dataset Creation

The auto-labeling pipeline (GroundingDINO + SAM) eliminates the most expensive part of training segmentation models: manual annotation. For a research team or a student project with limited resources—similar to the NATERIDA team’s constraint of a \$60 budget—this approach reduces the barrier to entry for computer vision research.

7.3 Environmental Monitoring

As demonstrated by NATERIDA, robots equipped with segmentation models can perform environmental monitoring in remote or hazardous areas. The five-class terrain model (soil, vegetation, obstacle, pedestrian, sky) provides sufficient information for a robot to understand its surroundings and make navigation decisions.

7.4 Scalability to Real Hardware

While this work uses a simulated robot, the trained model can be transferred to real hardware. The NATERIDA team has identified the NVIDIA Jetson Nano as a future compute platform, which is capable of running U-Net inference in real-time. The model trained in this work (approximately 31M parameters for a standard U-Net) is small enough for edge deployment with optimization techniques such as quantization or pruning.

8 Limitations

Several limitations of the current work should be acknowledged:

1. **Simulated data only:** All 908 images are collected in a Webots simulator. The visual appearance of simulated environments differs from real-world conditions (lighting, textures, weather). Domain adaptation or fine-tuning on real images would be necessary before deployment.
2. **Auto-label noise:** GroundingDINO and SAM are not perfect. Some masks contain errors—for example, misclassifying shadowed soil as obstacle, or missing small vegetation patches. This label noise limits the achievable performance ceiling.

3. **Moderate IoU:** The best validation IoU of 0.4453 means the model’s predictions overlap with ground truth by less than half on average. This is insufficient for safety-critical navigation but may be acceptable for preliminary terrain assessment.
4. **Limited class diversity:** Five classes are insufficient for complex real-world scenes. Additional classes (e.g., water, gravel, mulch, different crop types) would be needed for practical agricultural deployment.
5. **Fixed resolution:** Images are resized to 256×256 , losing fine details from the original 960×720 resolution. Small objects (e.g., distant pedestrians) may be lost during downsampling.
6. **No temporal consistency:** The model processes each frame independently. A video-based approach with temporal smoothing or recurrent architectures could improve consistency across consecutive frames.
7. **Single environment:** The simulator uses one fixed world configuration. The model may not generalize to different terrain types, lighting conditions, or seasons without additional data collection.

9 Future Work

Based on the identified limitations, the following improvements are planned:

1. **Real-world data collection:** Deploy the Pioneer 3-AT (or a NATERIDA-style robot with a Jetson Nano) in outdoor environments to collect real images and manually refine a subset of masks for validation.
2. **Increased dataset diversity:** Run the Webots simulation with varied lighting (time-of-day cycle), weather conditions (rain, fog), and terrain configurations to improve generalization.
3. **Model architecture improvements:** Experiment with DeepLabv3+ or SegFormer to compare performance against the current U-Net baseline. Consider adding an attention mechanism to focus on ambiguous boundary regions.
4. **Higher resolution training:** Use sliding window or patch-based approaches to train at higher effective resolutions (512×512 or 1024×1024) without exceeding GPU memory.
5. **Test-time augmentation (TTA):** Apply flips and multi-scale inference during evaluation to improve prediction quality at inference time.
6. **Integration with NATERIDA:** Combine the segmentation model with NATERIDA’s sensor fusion pipeline (lidar + camera) and the ResNet-18 plant disease classifier for a comprehensive agricultural monitoring system.
7. **Model optimization for edge deployment:** Apply quantization-aware training and TorchScript export to reduce model size and inference latency for deployment on resource-constrained devices like the Jetson Nano.
8. **Active learning:** Use model uncertainty to selectively annotate the most informative samples, improving performance with minimal manual labeling effort.

10 Conclusion

This project developed a semantic segmentation pipeline for agricultural robotics, from data collection through model training and live demonstration. By leveraging Webots

simulation for data generation, GroundingDINO and SAM for automatic annotation, and Albumentations for data augmentation, we created a complete pipeline that requires minimal manual effort. The U-Net model, trained on an NVIDIA GeForce RTX 4060 Laptop GPU for 100 epochs, achieved a validation IoU of 0.4453 and Dice coefficient of 0.5497 on a five-class terrain segmentation task. The trained model was then loaded into an active segmentation controller that performs real-time inference on the robot’s camera feed during Webots simulation, producing a live color-coded overlay that visualizes the robot’s terrain understanding.

While the performance is moderate compared to state-of-the-art benchmarks, the pipeline demonstrates a practical approach to dataset creation, model training, and simulated deployment for student projects and research teams with limited resources. The work builds upon the foundation established by the NATERIDA microbots project [1], extending its vision capabilities from image-level classification to pixel-level segmentation.

The main contributions are: (1) an automated labeling pipeline that eliminates manual annotation, (2) a trained segmentation model for five-class terrain understanding, (3) a real-time active segmentation controller demonstrated in simulation, and (4) a documented training pipeline that can be reproduced and extended. Future work will focus on using the segmentation output for navigation decisions, real-world data collection, model optimization for edge deployment, and integration with the NATERIDA robotic platform.

References

- [1] Prasiddha Mainali, Prashanna Tiwari, and Prince Kumar Mandal, “NATERIDA microbots,” *Devpost*, Aug. 2025. Available: <https://devpost.com/software/naterida-microbots-i4bv8r>
- [2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, “U-Net: Convolutional networks for biomedical image segmentation,” in *Proc. MICCAI*, 2015, pp. 234–241.
- [3] Jonathan Long, Evan Shelhamer, and Trevor Darrell, “Fully convolutional networks for semantic segmentation,” in *Proc. CVPR*, 2015, pp. 3431–3440.
- [4] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” in *Proc. ECCV*, 2018, pp. 801–818.
- [5] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo, “SegFormer: Simple and efficient design for semantic segmentation with transformers,” in *Proc. NeurIPS*, 2021, pp. 12077–12090.
- [6] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, et al., “Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection,” in *Proc. ECCV*, 2024.
- [7] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, et al., “Segment Anything,” in *Proc. ICCV*, 2023, pp. 4015–4026.
- [8] Olivier Michel, “Webots: Professional mobile robot simulation,” *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.