Vibe Coders: Off-Road Autonomous Semantic Segmentation

Project Journey & Technical Documentation

Vibe Coders Team

February 7, 2026

1 Executive Summary & Problem Statement

The Problem

Traditional autonomous driving systems rely heavily on structured environments—lanes, traffic lights, and paved roads. However, true autonomy requires navigating unstructured terrain. In off-road scenarios (deserts, forests, disaster zones), vehicles encounter "negative obstacles" (holes), "positive obstacles" (rocks, logs), and ambiguous terrain (dry grass vs. dry bushes) that standard Computer Vision models fail to distinguish.

The Challenge

Our objective was to build a semantic segmentation model capable of classifying 10 specific off-road classes with high precision. The critical challenge was distinguishing between visually similar textures (e.g., "Dry Grass" vs. "Dry Bushes") and identifying small but dangerous obstacles like "Rocks" and "Logs" which often represent less than 1% of the pixels in an image but carry the highest risk for vehicle damage.

The Solution

We developed a deep learning solution utilizing **DINOv2 (Vision Transformer)** as a backbone, leveraged for its exceptional feature extraction capabilities in unstructured environments. We paired this with a Feature Pyramid Network (FPN) decoder and a "Nuclear" class-weighting strategy to force the model to prioritize safety-critical obstacles.

2 Features & Key Technical Highlights

Our solution integrates several advanced computer vision techniques to solve the specific constraints of the dataset and the environment.

- **Foundation Model Backbone:** We utilized `dinov2_vitb14_reg` (Vision Transformer Base with Registers). Unlike standard CNNs (ResNet), DINOv2 understands global context, which is crucial when a "rock" looks identical to "ground clutter" without the surrounding context.
- **Native Resolution Training:** Unlike standard pipelines that resize images to small squares (e.g., 224x224), we trained at **952x532**. This high resolution was critical for detecting small hazards like distant rocks.

- **"Nuclear" Class Weights:** We implemented a custom loss weighting strategy. Since "Logs" and "Rocks" are rare, standard models ignore them. We assigned a weight of **50.0 to Logs** and **30.0 to Rocks**, compared to 0.1 for Sky. This forces the model to "panic" if it misses an obstacle.
- **Composite Loss Function:** We combined three loss functions to stabilize training:
  1. **Focal Loss:** To focus on hard-to-classify examples.
  2. **Dice Loss:** To optimize the overlap of the segmentation masks.
  3. **Cross Entropy:** For general pixel-wise classification.
- **Desert Augmentation Pipeline:** Using Albumentations, we created a specific pipeline to simulate harsh outdoor conditions: Solar flares (Solarize), harsh shadows (RandomShadow), and extreme contrast changes.

3 Environment Setup & Tech Stack

The project was engineered to run efficiently within cloud-based interactive environments while pushing the limits of the available hardware.

Hardware Resources

- **Platform:** Google Colab
- **GPU:** NVIDIA T4 Tensor Core (16GB VRAM)
- **System RAM:** ~12.7 GB (Standard Tier)
- **Storage:** Google Drive Integration for checkpoint persistence.

Software Stack

- **Language:** Python 3.10+
- **Deep Learning Framework:** PyTorch (with torch.cuda.amp for Mixed Precision).
- **Image Processing:** Albumentations 1.3.1 (downgraded specifically for compatibility), OpenCV, PIL.
- **Data Handling:** NumPy, Pandas.
- **Visualization:** Matplotlib, Seaborn (for Confusion Matrices).

Installation Requirements

The environment requires a specific version of Albumentations to avoid breaking changes in the latest updates:

```
!pip install albumentations==1.3.1
```

4 The Development Journey: Methodology

Step 1: Data Ingestion & Security

We implemented a strict security protocol during the unzipping process. The script scans the zip file contents before extraction to ensure no "test" data leaks into the training set (Data Leakage Prevention).

- *Total Training/Val Images:* 6,348
- *Classes:* 10 (Background, Trees, Lush Bushes, Dry Grass, Dry Bushes, Ground Clutter, Logs, Rocks, Landscape, Sky).

Step 2: Smart Unfreezing

We froze the DINOv2 backbone initially to preserve its pre-trained knowledge. However, to adapt it to the specific off-road domain, we performed a "Partial Unfreeze" of blocks 8 through 11 (the last 4 layers). This allowed the model to learn "desert concepts" while retaining "object concepts."

Step 3: Solving the RAM Bottleneck

Training at 952x532 resolution on a T4 GPU is memory-intensive. We used **Gradient Accumulation**:

- *Physical Batch Size:* 2 (The maximum the GPU could hold).
- *Accumulation Steps:* 16.
- *Effective Batch Size:* 32.

This trick allowed us to simulate high-end server training on a free-tier GPU.

5 Results & Analysis

A. Training Performance

The training process ran for 20 epochs. As seen in the training curves below, the model showed steady convergence.

- **Loss Curve:** The Composite Loss dropped from 1.69 to $\sim$1.01, indicating the model successfully learned to minimize the error across all three loss components (Focal, Dice, and CE).
- **IoU Curve:** The Validation Intersection over Union (IoU) improved from 0.35 to a peak of **0.4754**.
- **Observation:** The gap between Train and Validation metrics remained narrow, indicating that our heavy augmentation pipeline successfully prevented overfitting.

B. Confusion Matrix Analysis

The confusion matrix provides deep insight into where the model succeeds and fails.

- **Successes:**
  - **Sky (0.99):** The model has essentially solved Sky segmentation.
  - **Dry Grass (0.77):** Excellent performance on the most dominant terrain type.

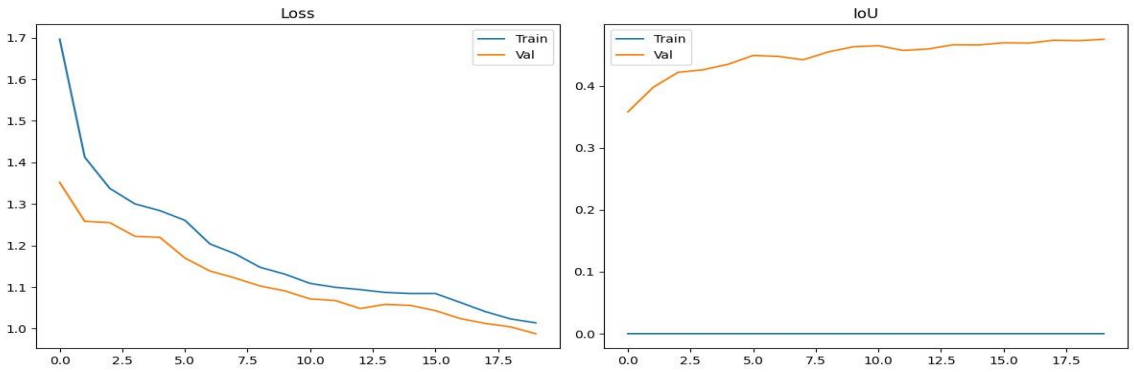

Figure 1: Loss and IoU metrics over 20 epochs. Note the stability of the validation curve.

  - **Rocks (0.62):** Considering rocks are small and difficult, a 62% true positive rate is a strong result, largely due to our weighting strategy.

- **Challenges:**
  - **Logs (0.00):** The model struggled significantly with logs. This is likely due to their visual similarity to “Ground Clutter” and their extreme scarcity in the dataset.
  - **Dry Bushes vs. Dry Grass:** There is some confusion (0.24) where the model thinks Dry Bushes are Dry Grass, which is understandable as they share identical textures.

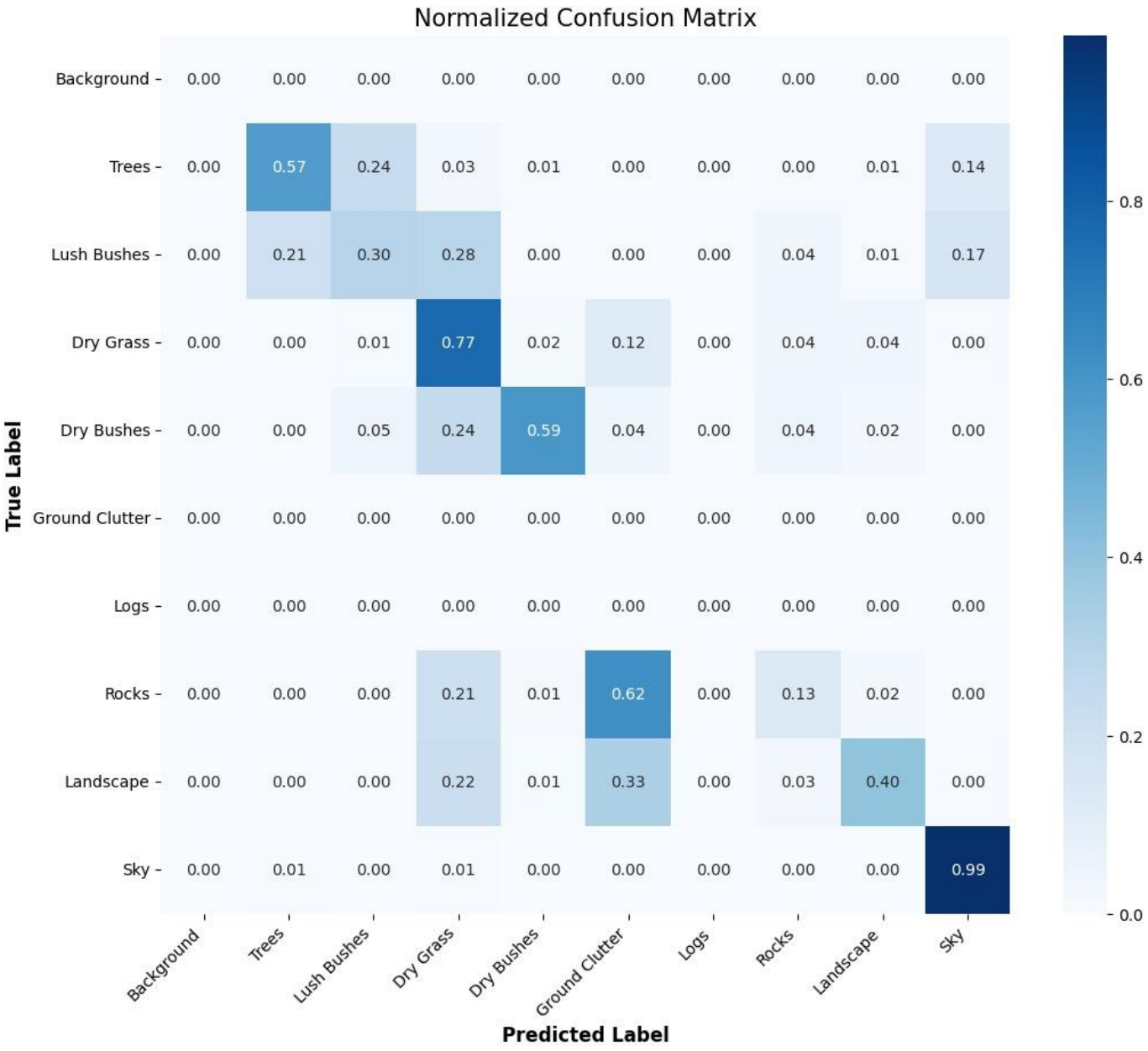

Figure 2: Normalized Confusion Matrix. Darker blues on the diagonal indicate correct predictions.

C. Benchmarks & Resource Consumption

- **Inference Latency:** 277.23 ms per image.
- **Throughput:** ~3.61 FPS (Frames Per Second).
- **Implication:** This speed is suitable for a rover moving at low-to-moderate speeds (approx 5-10 mph), allowing for decision-making every ~30cm of travel.
- **Training Cost:** $0.00 (Utilized Google Colab Free Tier).

6 Use Cases

1. **Mars/Lunar Exploration Rovers:** The “Rock” and “Log” detection capabilities are directly

applicable to avoiding hazards on extraterrestrial surfaces where GPS is unavailable.

2. **Agricultural Robotics:** Autonomous tractors operating in unmapped fields need to distinguish between crops (Lush Bushes) and obstacles (Logs/Rocks).
3. **Search and Rescue Drones:** Flying over forest fire aftermaths (Dry Grass/Dry Bushes) to find safe landing zones free of debris.

7 File Structure

The project is organized into a modular structure to separate data, checkpoints, and source code.

```
/content/
|-- vibe_coders_project/
|   |-- train.py              # Main training loop, model, loss
|   |-- test.py               # Inference script, metrics, confusion matrix
|   |-- utils.py              # Helper functions (Augmentations, etc.)
|   |-- requirements.txt      # Dependency definitions
|   L-- README.txt            # Instructions
|
|-- temp_data/                # Ephemeral fast storage for datasets
|   L-- Offroad_Segmentation_Training_Dataset/
|       |-- train/
|       L-- val/
|
L-- drive/MyDrive/Hackathon_Data/ # Persistent Storage
    |-- checkpoints/
    |   L-- best_model.pth    # Saved Model Weights (approx 400MB)
    L-- final_benchmark/      # Output graphs and text reports
```

8 Code Documentation (Snippet Explanations)

The Model (FPN Decoder)

We chose a Feature Pyramid Network because off-road obstacles vary wildly in size.

```
class FPNDecoder(nn.Module):
    # ... (Initialization) ...
    def forward(self, x):
        # We process features at multiple scales (fpn1, fpn2, fpn3)
        # and upsample them to combine semantic strength
        # with spatial resolution.
        x = self.lateral(x)
        x = self.fpn1(x)
        x = self.fpn2(x)
        # ... classification head ...
        return self.classifier(x)
```

The Nuclear Weights

This dictionary defined our priority system.

```
CLASS_WEIGHTS = torch.tensor([
    1.0,    # Background
    5.0,    # Ground Clutter
    50.0,   # Logs (CRITICAL PRIORITY)
    30.0,   # Rocks (HIGH PRIORITY)
    0.1,    # Sky (Low Priority)
], dtype=torch.float32)
```

9 Future Improvements

1. **Log Detection:** Gather more data specifically containing logs to fix the 0% accuracy in that class.
2. **Temporal Consistency:** Implement video-based inference where the prediction of the previous frame informs the current frame (smoothing).
3. **Quantization:** Convert the model to TensorRT or ONNX to increase FPS from 3.6 to 30+ for high-speed driving.