UNDERSTORY

Training Guide — How to Teach Understory About Your Forest

This guide walks you through the full process of taking raw scan data from your forest and using it to train a better model. After training, the program will recognize the trees, ground, and plants in your specific forest type much more accurately.

You do not need programming experience to follow this guide. Just follow each step in order.

What is Training and Why Do It?

Understory uses a computer model (like a brain) to look at your point cloud and decide what each point is — ground, tree trunk, leaf, or fallen wood. The model that comes with the program was trained on forests in Australia. Your forest may look very different.

Training means showing the model examples from your forest so it learns to recognize your trees and plants correctly.

Think of it like teaching a child: you show them "this is a tree trunk" and "this is a leaf" many times, and they learn to tell the difference on their own.

Before You Start

You will need:

- A scan of your forest (a .las or .laz file)
- The Understory program installed and working
- About 1-2 hours of time for the first training (less for future ones)

Step 1: Run the Pipeline on Your Scan

First, let the program try to segment your point cloud with the model it already has. Even if the results are not perfect, this gives you a starting point so you don't have to label every point by hand.

1. Open Understory (`./venv/bin/python -m understory`)
2. Click **New Project** and give it a name (example: "Rio Negro Site 1")
3. Click **Open Point Cloud** and select your .las file
4. Go to the **Prepare** tab:
    - Click **Crop Outliers** to remove far-away noise points
    - Click **Save Cloud** to save the cleaned version
5. Go to the **Process** tab:
    - Set your **Plot Radius** (or leave at 0 for the whole cloud)
    - Make sure all 5 pipeline stages are checked
    - Click **Run Pipeline**
6. Wait for the pipeline to finish. You will see progress in the status bar.

When it finishes, the program creates a file called `segmented.las` in your run folder. This file has every point labeled with a class (terrain, vegetation, CWD, or stem) and a **confidence score** — how sure the model was about each label.

Step 2: Open the Label Editor

Now you will check the model's work and fix mistakes.

1. Go to the **Training** tab in the left sidebar
2. Under **Step 3: Review & Correct Labels**, click **Open Label Editor**
3. Browse to the `segmented.las` file from your pipeline run
    - It is in your project folder under `runs/run_XXXX/output/segmented.las`
4. The Label Editor opens showing your point cloud with colors for each class:
    - **Brown** = Terrain (ground)
    - **Green** = Vegetation (leaves, small plants)
    - **Gold** = CWD (fallen wood, dead branches on the ground)
    - **Red** = Stem (tree trunks)

Step 3: Find Mistakes Using Confidence View

The model tells you where it is not sure. Points with low confidence are most likely to be wrong.

1. Press the **C** key to turn on **Confidence Coloring**
    - **Red points** = the model is NOT sure (these are likely wrong)
    - **Yellow points** = the model is somewhat sure
    - **Green points** = the model is very sure (these are likely correct)
2. Look at the red and yellow areas — these are where you need to check the labels
3. You can also click **Select Low Confidence** to automatically select all uncertain points below a threshold (default 0.5)

Press **C** again to go back to normal class coloring when you want to see the labels.

Step 4: Navigate the Point Cloud

Moving around the point cloud is important for checking labels carefully.

- **Scroll wheel** — Zoom in and out
- **Left-click and drag** — Rotate the view
- **Middle-click and drag** — Pan (move sideways)
- **Press F** then **right-click a point** — Set that point as the center of rotation (very helpful for focusing on a specific tree)
- **Home key** — Reset the view to see the whole cloud
- **Top/Front/Right/Iso buttons** — Quick camera angles

Turn on **EDL** (Eye-Dome Lighting) for better depth — it makes it easier to see shapes in the point cloud.

Step 5: Fix Wrong Labels

Now you will correct the mistakes you found.

Hide Classes You Don't Need

Use the **checkboxes** next to each class name to hide classes. For example, if you want to fix stem points that were labeled as CWD:

1. Uncheck **Terrain** and **Vegetation** to hide them
2. Now you can see only CWD and Stem points
3. This makes it much easier to select the mislabeled points

Select and Paint

1. Click the **Box Select** button (or it may already be active)
2. **Draw a rectangle** around the points you want to change by clicking and dragging
3. The selected points turn **white**
4. Choose the correct class:
    - Press **1** for Terrain
    - Press **2** for Vegetation
    - Press **3** for CWD
    - Press **4** for Stem
    - Or click the class radio button and then click **Paint Selected**
5. The points change to their new color

Tips for Faster Editing

- **Focus on the big mistakes first** — If a whole tree trunk is labeled as vegetation, fix that before worrying about individual points
- **Use confidence view** (C key) to find problem areas quickly
- **Use Select Low Confidence** to grab many uncertain points at once, then paint them to the correct class
- **Undo mistakes** with **Ctrl+Z** — you can undo as many times as you need
- **Turn layers on and off** to isolate what you're working on
- You do NOT need to fix every single point. Even 80% correct training data helps the model learn.

Step 6: Save Your Corrected Labels

When you are happy with your corrections:

1. Click the **Save Labels** button in the toolbar
2. Choose where to save the file and give it a name (example: `rio_negro_site1_corrected.las`)
3. Save it somewhere you can find it easily

This saved file is your **training data** — it has the correct labels that you verified and fixed.

Step 7: Split Your Data for Training and Validation

This is an important step that many people skip — and it makes a big difference.

Why You Need Two Sets of Data

When training, the model needs two separate sets of data:

- **Training data** — the examples the model learns from (like a textbook)
- **Validation data** — a separate set the model is tested on but never learns from (like an exam)

If you only have training data and no validation data, you have no way to tell if the model is actually learning useful patterns or just memorizing your specific point cloud. The validation data acts as a reality check.

The training data and validation data must be different. You cannot use the same file for both.

How to Split Your Data

You have two good options:

Option A: Use Different Scans (Best)

If you have multiple scan plots from your forest:

- Put most of them in `data/train/` (for training)
- Put 1-2 of them in `data/validation/` (for validation)

For example, if you have 5 corrected scans:

- 4 go in `data/train/`

- 1 goes in `data/validation/`

Option B: Split One Scan (Good Enough)

If you only have one corrected scan, open it in the Label Editor and save two copies:

1. Load your corrected scan
2. Use **Box Select** to select roughly 20-30% of the point cloud (for example, one side or corner)
3. Save that selection as `my_forest_validation.las`
4. Undo the selection, then select the remaining 70-80%
5. Save that as `my_forest_training.las`

Then:

- Import `my_forest_training.las` into `data/train/`
- Manually copy `my_forest_validation.las` into `data/validation/`

You can find the validation folder at: `data/validation/` inside your Understory installation directory.

What Happens If You Skip Validation?

Training will still run, but:

- You will not see the orange validation line on the loss chart
- You will have no way to detect overfitting (the model memorizing instead of learning)
- You might train for too long and end up with a worse model

Even a small validation set is much better than none.

Step 8: Import Training Data

Now bring your corrected files into the training system.

1. Go back to the **Training** tab
2. Under **Step 1: Import Training Data**, click **Import Files**
3. Select your corrected .las file(s) — these go into `data/train/`
4. The files are copied to the training data folder

For validation data, manually copy your validation .las file(s) into the `data/validation/` folder inside the Understory installation directory.

Note: The `example.las` files that come with the program are just placeholders so the folders are not empty. You can delete them once you have your own data. They will not interfere with training if left in place, but removing them avoids mixing unrelated data into your model.

More training data = better results. If you have scans from multiple plots in the same forest type, correct and import all of them. Even 2-3 corrected scans make a big difference.

Step 9: Configure Training

Under **Step 4: Configure Training**, set these options:

Setting	What It Does	Suggested Value
Model filename	Name for your new model	`my_forest_model.pth`
Epochs	How many times the model studies the data	100-400 for fine-tuning
Learning rate	How fast the model learns (smaller = more careful)	0.000025 (the default is good)
Training batch size	How many samples at once	2-10 (higher if you have VRAM to spare)
Device	Use GPU or CPU	`cuda` if you have a GPU
Load existing model	Start from the existing model (recommended)	Checked (yes)
Class weights	Give extra attention to rare classes	Auto (recommended)

Important Settings Explained

Load existing model weights — Keep this checked. This means the model starts with what it already knows and just learns the new things about your forest. This is called "fine-tuning" and it is much faster and better than starting from zero.

Class weights: Auto — In most forests, there are many more ground and vegetation points than CWD or stem points. Without class weights, the model might ignore the rare classes. "Auto" fixes this by

paying extra attention to the classes with fewer points.

Epochs — Each epoch is one full pass through your training data. More is not always better — training too long causes overfitting (see the loss chart section below). Recommended epochs:

Scenario	Epochs	Why
Fine-tuning with existing model (recommended)	100-400	The model already knows forests. It just needs to learn your specific trees.
Training from scratch (no existing model)	1000-3000	Starting from zero takes much longer.
Lots of training data (5+ scans)	200-600	More data means the model can learn longer without overfitting.
Very little data (1 scan)	50-150	With limited data, the model overfits quickly. Stop early.

Training batch size — This controls how many training boxes the model processes at once. Higher values use more GPU memory but can be slightly faster. A batch size of 2 works on most GPUs. If you have a GPU with lots of VRAM (24 GB+), try 4-10.

Step 10: Train the Model

1. Click **Start Training**
2. Watch the **training console** at the bottom of Step 5 — it shows detailed output from each epoch including loss and accuracy
3. The **GPU monitor** shows your GPU utilization and memory usage during training
4. The **loss chart** updates in real time (see below for how to read it)
5. You can **Pause** training to check on things and **Resume** when ready
6. You can **Stop** training at any time — the model is saved after every epoch, so you do not lose progress

Training takes from a few minutes to several hours depending on:

- How much training data you have
- How many epochs you set
- Your GPU speed and batch size

When finished, you will see a message telling you where the model was saved.

How to Read the Training Loss Chart

The loss chart is the most important tool for understanding whether your training is working well. It shows two lines:

- **Green line (Train)** — how wrong the model is on the training data
- **Orange line (Validation)** — how wrong the model is on data it has never seen

Lower loss = better. Both lines should go down over time.

What Good Training Looks Like

```
Loss
 3 |
   |  \
 2 |   \  _____ orange (validation)
   |    \/
 1 |     \_____ green (training)
   |
 0 +------------>
   0    50   100   Epoch
```

Both lines drop together and level off at similar values. The model is learning patterns that work on new data too.

What Overfitting Looks Like

```
Loss
 3 |
   |  \
 2 |   \   ___/--- orange going UP (bad!)
   |    \ /
 1 |     X
   |      \_____ green keeps dropping
 0 +------------>
   0    50   100   Epoch
```

The green line keeps going down, but the orange line stops improving or goes back up. This means the model is **memorizing** the training data instead of learning general patterns. It will perform well on the

training data but poorly on new scans.

What to do when you see overfitting:

1. **Stop training** — the best model was saved at the epoch where validation loss was lowest
2. **Add more training data** — this is the most effective fix
3. **Reduce epochs** — next time, train for fewer epochs (stop around where validation loss was at its lowest)
4. **Check your data** — make sure you have validation data in `data/validation/` (see Step 7)

What a Noisy/Spiky Chart Means

If the orange (validation) line is very spiky and jumpy:

- Your **validation dataset is too small** — add more labeled data to `data/validation/`
- A smooth validation curve means enough validation data for reliable measurement

Summary Table

What You See	What It Means	What to Do
Both lines drop together	Training is working well	Keep going, stop when they level off
Green drops, orange stays flat	Starting to overfit	Stop soon, add more data
Green drops, orange goes up	Overfitting	Stop now, use the model from earlier epochs
Orange is very spiky	Validation set too small	Add more data to `data/validation/`
Neither line drops	Model is not learning	Check labels for errors, try higher learning rate
Both lines drop but are far apart	Mild overfitting	Add more data, try fewer epochs

---

Step 11: Use Your New Model

1. Go to the **Process** tab
2. Under **Model**, click **Import Model** and select your new `.pth` file
3. The model dropdown now shows your new model
4. Run the pipeline again on your data
5. Compare the results — the segmentation should be more accurate for your forest type

---

The Training Cycle

Training works best when you repeat the process:

```
Scan your forest
      |
      v
Run pipeline (with current model)
      |
      v
Check results in Label Editor
      |
      v
Fix the biggest mistakes
      |
      v
Save corrected labels
      |
      v
Split into training and validation sets
      |
      v
Train a new model (watch the loss chart!)
      |
      v
Run pipeline again (with new model)
      |
      v
Results are better! Repeat if needed.
```

Each time you go through this cycle, the model gets better at recognizing your specific forest. After 2-3 rounds, the model usually needs very few corrections.

Tips for Best Results

How Much Data Do You Need?

- **Minimum**: 1 corrected scan, split into training and validation pieces (the model will improve but may not generalize well)
- **Good**: 3-5 corrected scans from different parts of your forest
- **Best**: 5-10 scans covering the variety of conditions in your area

Where Does Each File Go?

What	Where	Purpose
Labeled .las files for learning	`data/train/`	The model studies these to learn
Labeled .las files for testing	`data/validation/`	The model is tested on these but never learns from them
Your trained model	`model/`	Saved automatically after each epoch
Training history	`model/training_history.csv`	Epoch-by-epoch loss and accuracy numbers

What to Focus On When Correcting

Focus your correction time on these common mistakes:

1. **Tree trunks labeled as CWD** — This is the most common error
2. **Low vegetation labeled as terrain** — Happens with dense understory
3. **Fallen logs labeled as stems** — CWD on the ground vs standing stems
4. **Large branches labeled as vegetation** — Should be stems

Different Forest Types Need Different Models

A model trained on Amazon rainforest will not work well on mangroves, and vice versa. If you work in very different forest types, consider training separate models:

- `amazon_model.pth` — for tall rainforest
- `mangrove_model.pth` — for coastal mangroves
- `cerrado_model.pth` — for savanna woodland

Save Your Work Often

- Save corrected label files with clear names that tell you what they are
- Keep notes about which scans you used for training
- Save your project before and after each pipeline run

Troubleshooting

"CUDA out of memory" error during training

- Lower the **Training batch size** to 2 or 3
- Close other programs that use the GPU

Training seems stuck (loss not going down)

- Try a higher **Learning rate** (0.0001)
- Make sure your training data has correct labels — wrong labels confuse the model
- Add more training data

Results are worse after training

- The model may be overfitting — check the loss chart (see above). The best model is from the epoch where validation loss was lowest, not the last epoch
- You may have too few corrected scans — add more training data
- The corrections may have introduced errors — double-check your labels

Validation loss is going up while training loss goes down

- This is overfitting. See the "How to Read the Training Loss Chart" section above

- Stop training and use the model from an earlier epoch
- Next time, train for fewer epochs and add more data

The model is good for one area but bad for another

- This is normal. Different forests need different training data
- Add corrected scans from the problem area and retrain

No validation line on the loss chart

- You need labeled data in `data/validation/` — see Step 7
- Make sure "Run validation during training" is checked in Step 4

Quick Reference

Task	Where to Find It
Run pipeline	Process tab > Run Pipeline
Open Label Editor	Training tab > Step 3 > Open Label Editor
See confidence colors	Label Editor > press C
Select uncertain points	Label Editor > Select Low Confidence
Fix a label	Select points > press 1/2/3/4
Hide a class	Label Editor > uncheck the class checkbox
Set camera focus	Press F, then right-click a point
Save corrections	Label Editor > Save Labels
Import for training	Training tab > Step 1 > Import Files
Pause training	Training tab > Step 5 > Pause
Stop training	Training tab > Step 5 > Stop
Start training	Training tab > Step 5 > Start Training

Use new model	Process tab > Import Model