

Alexey Kozhakin¹, Tram T.N. Nguyen¹, Dylan Seychell², Janice Formosa Pace¹, Saviour Formosa¹

Affiliations: ¹Department of Criminology. ²Department of Artificial Intelligence, University of Malta

Criminology In Action Occasional Papers

Series 2025:04

DOI: 10.5281/zenodo.22308255

September 2025

Cited as: Kozhakin, A., Nguyen, T.T.N., Seychell, D., Formosa Pace, J., Formosa, S., (2025). *Extracting those LiDAR pointclouds – a ML search for shapes*, Criminology In Action Occasional Papers: Department of Criminology, University of Malta, Msida, 15 September 2025, Series 2025:04. DOI: 10.5281/zenodo.22308255

1.INTRODUCTION

Emerging in the 1960s, LiDAR technology has experienced significant development due to its undeniable advantages and is now widely applied across various fields. Its most successful integrations have been seen in geodesy, cartography, archaeology, environmental monitoring, and the automotive industry. Today, LiDAR has become an integral part of modern systems such as unmanned aerial vehicles (UAVs), drones, and augmented reality platforms.

LiDAR has supplemented—and in some cases replaced—traditional methods of spatial data acquisition. Providing high measurement accuracy, these technologies enable the rapid collection of large volumes of data.

Alongside the development of LiDAR itself, there has been substantial progress in methods for interpreting the resulting data. Traditional approaches such as statistical analysis, geostatistics, and geospatial analysis are widely used. However, intelligent solutions based on machine learning (ML) have recently become more prominent for deriving valuable analytical insights from LiDAR data.

Machine learning methods have found wide application in the processing of LiDAR-derived 3D data, especially in geodesy and cartography. The data collected is used to generate digital elevation models (DEMs), which serve as a foundation for numerous mapping and surveying applications. ML techniques ensure both high processing speed and sufficient accuracy when interpreting spatial information.

For example, in [1], researchers proposed a new method combining adaptive iterative segmentation and hierarchical overlay analysis through data fusion to process LiDAR data. Their approach achieved a segmentation quality of 85.5%. In [2], the authors conducted six classification experiments using features from single and multiple data sources to obtain accurate 2D and 3D urban morphological parameters. Classification methods included K-Nearest Neighbors (KNN), Random Forest (RF), and Extreme Gradient Boosting (XGBoost). XGBoost demonstrated superior performance with an overall accuracy of 86.22%. In

another study [3], a Probabilistic Neural Network (PNN) classifier was developed for roadside LiDAR object classification, achieving an accuracy of 97.6% on the test dataset.

The domain analysis reveals a wide array of intelligent techniques applied in LiDAR data analysis. However, such methods often require significant computational resources. In settings where computing power is limited, processing more than several terabytes of data can be a serious challenge.

This research proposes the development of a custom method for semantic segmentation of LiDAR data. The scientific novelty lies in implementing a new intelligent algorithm based on machine learning techniques, specifically neural networks. Notably, the proposed approach does not require extensive computational power due to its architectural efficiency and well-structured preprocessing pipeline.

This study focuses on the processing of 3D surface data for the Republic of Malta, acquired via LiDAR. The research objectives are as follows:

- To develop a custom method for the semantic segmentation of 3D LiDAR data and train it using the open STPLS3D dataset;
- To evaluate the method's performance on a custom dataset;
- To generate a 2D map of Malta highlighting building footprints;

2. METHODS

The proposed workflow for 3D LiDAR data segmentation involves several sequential stages, beginning with data acquisition and preprocessing and concluding with the classification of each point in the LiDAR cloud and the visualization of results. The entire pipeline is modular and designed to operate efficiently even on limited computing resources.



Fig 1- Steps of pipeline

2.1 DATA PREPROCESSING

To ensure consistent quality and input for the machine learning model, a preprocessing phase is applied to the raw LiDAR data. This stage includes:

- **Outlier Removal:** Noise and erroneous measurements are filtered out using statistical and spatial filtering techniques. Outliers are removed to improve model accuracy and reduce the

likelihood of misclassification.

- **Point Cloud Sparsification:** To reduce computational load, point density is adjusted by selecting a representative subset of the data. This helps balance detail and efficiency during training.
- **Tiling:** The filtered and (if necessary) sparsified data is divided into square tiles (e.g., 500×500 points). This allows for batch processing and parallelization in further stages.

These steps are performed automatically before model inference begins.

2.2 PROJECTION TO 2D AND FEATURE ENCODING

Each 3D tile is projected onto a 2D grid using a custom projection method that retains key height and geometric features. A multi-channel tensor is formed, where each channel encodes a specific property of the original 3D data (e.g., height, intensity, local geometry features such as curvature or roughness).

This step ensures compatibility with convolutional neural networks (CNNs), which require 2D image-like input data. The feature tensor typically includes 9 to 12 channels, depending on the configuration.

A critical step in transforming LiDAR point clouds into a 2D format suitable for deep learning is the generation of informative features (also called channels) based on local spatial neighborhoods. Each cell of the output tensor corresponds to a small area of the original tile, and contains a vector of features computed from nearby 3D points.

GRID CONSTRUCTION

For each tile (typically 1×1 km in size), we generate a regular grid of size $M \times M$ (e.g., 512×512), covering the entire tile area. Each grid cell is associated with a central coordinate (x, y) and will serve as a location for feature computation.

NEAREST NEIGHBOR SEARCH

For each grid cell, we identify K nearest neighbors from the original point cloud using the XY coordinates. This is implemented efficiently using KD-tree search. The result is a 4D array of shape (M, M, K, D) , where D is the number of raw features available per point (typically 3 to 7: x, y, z, r, g, b, classification).

FEATURE COMPUTATION

From the K neighbors within each grid cell, we compute a set of statistical and geometric features, resulting in a tensor of shape (M, M, C) , where C is the number of feature channels. These features may include:

- **Mean elevation (z_mean):**

$$z_{mean_{i,j}} = \frac{1}{K} \sum_{k=1}^K z_{i,j,k}$$

- **Standard deviation of elevation (z_std):**

$$z_{std_{i,j}} = \sqrt{\frac{1}{K} \sum_{k=1}^K (z_{i,j,k} - z_{mean_{i,j}})^2}$$

- **Local surface normal components (approximated via PCA or covariance):**

- Vertical component:

$$n_z = \frac{\sigma_z}{\sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2}}$$

- Radial (horizontal) component:

$$n_r = \frac{\sqrt{\sigma_x^2 + \sigma_y^2}}{\sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2}}$$

Here, $\sigma_x, \sigma_y, \sigma_z$ are standard deviations of the K neighboring points along respective axes.

FINAL OUTPUT

The result is a set of multi-channel 2D tensors representing each tile, where each pixel contains locally aggregated information about the underlying 3D structure. These tensors are then used as input for convolutional neural networks (CNNs) or other computer vision models.

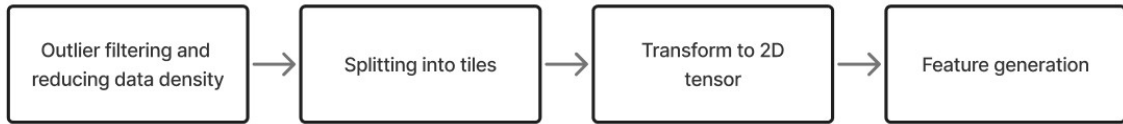


Fig 2 - Preprocessing

2.3 NEURAL NETWORK ARCHITECTURE

A modified U-Net architecture is employed for semantic segmentation. U-Net is known for its strong performance on pixel-wise classification tasks due to its encoder-decoder structure with skip connections. In this study, the U-Net model was adapted to process multi-channel inputs corresponding to the projected LiDAR tiles.

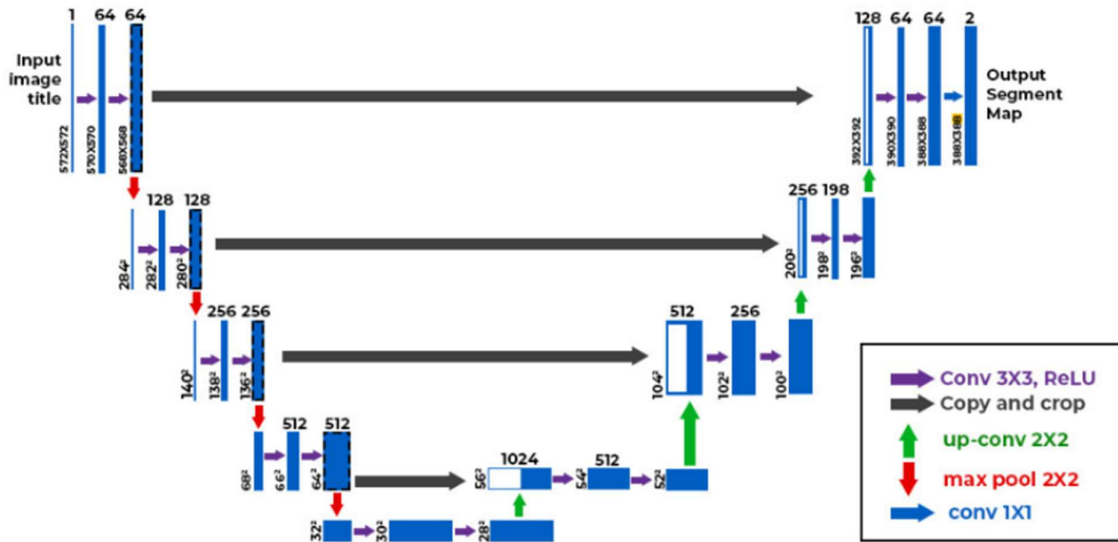


Fig 3 - U-Net architecture

The model is trained on the **STPLS3D** open dataset, which provides labeled point clouds across multiple urban scenes. Augmentation techniques such as rotation, flipping, and scaling are used to improve generalization.

2.4 PREDICTION AND REVERSE MAPPING

After inference, the model produces a semantic map for each tile, where each pixel corresponds to a predicted class (e.g., building, road, vegetation). This 2D result is then mapped back onto the original 3D points using coordinate indexing saved during the tiling and projection phase.

Each 3D point thus receives a predicted class label. In addition to class, color (RGB) values are assigned to each point based on the class-to-color mapping used in visualization.

2.5 POSTPROCESSING AND VECTORIZATION

Following classification, the segmented 2D masks can be optionally post-processed to refine object boundaries using morphological operations. From these masks, vector polygons are extracted and saved in standard GIS formats (e.g., shapefiles), enabling integration with mapping software.

Simultaneously, the newly labeled point cloud (with class and RGB data) is saved in LAS format, preserving both geometry and semantic information for further 3D analysis.

3. MODEL TRAINING

In this study, two primary segmentation tasks were addressed: **binary segmentation** and **multi-class segmentation** (Table 1). These tasks differ in terms of model output, application focus, and evaluation strategies.

TABLE 1 – SEGMENTATION TASKS OVERVIEW

Segmentation Task	Building Segmentation Model	Multi-class Segmentation Model
Framework	PyTorch with Segmentation Models PyTorch	PyTorch with Segmentation Models PyTorch
Architecture	U-Net with ResNet34 encoder	U-Net with ResNet34 encoder
Input Channels	3 (RGB or feature channels)	3 (RGB or feature channels)
Output Classes	2 (building / non-building)	20+ (buildings, roads, vegetation, water)
Activation Function	Softmax (binary classification)	Softmax (multi-class classification)
Purpose	Building footprint extraction	Comprehensive object classification

TRAINING PROCEDURE

Both segmentation models were trained using a similar training pipeline:

- **Loss Function:** Cross-entropy loss
- **Optimizer:** Adam optimizer
- **Data Augmentation:** Standard image-based transformations (e.g., flipping, rotation, scaling)
- **Validation Strategy:** Hold-out validation set
- **Model Checkpointing:** Automatic saving of best-performing models for reproducibility

4. PERFORMANCE AND LIMITATIONS

The building segmentation model achieved high accuracy for detecting building footprints. Inference was efficient, processing approximately 8 images per batch on CPU-based systems. The architecture was optimized for deployment both on GPU and CPU environments, with low memory overhead.

However, the dataset used for training was **imbalanced**, particularly in the multi-class segmentation setting. As a result, **training loss and accuracy graphs exhibited noticeable fluctuations**. To better assess model learning behavior, we visualized performance for the four most frequent classes:

- **Class 0** – Ground
- **Class 1** – Building
- **Class 2** – Low Vegetation
- **Class 5** – Vehicle

Figure 4 presents dynamics metrics performance throughout the training epochs.

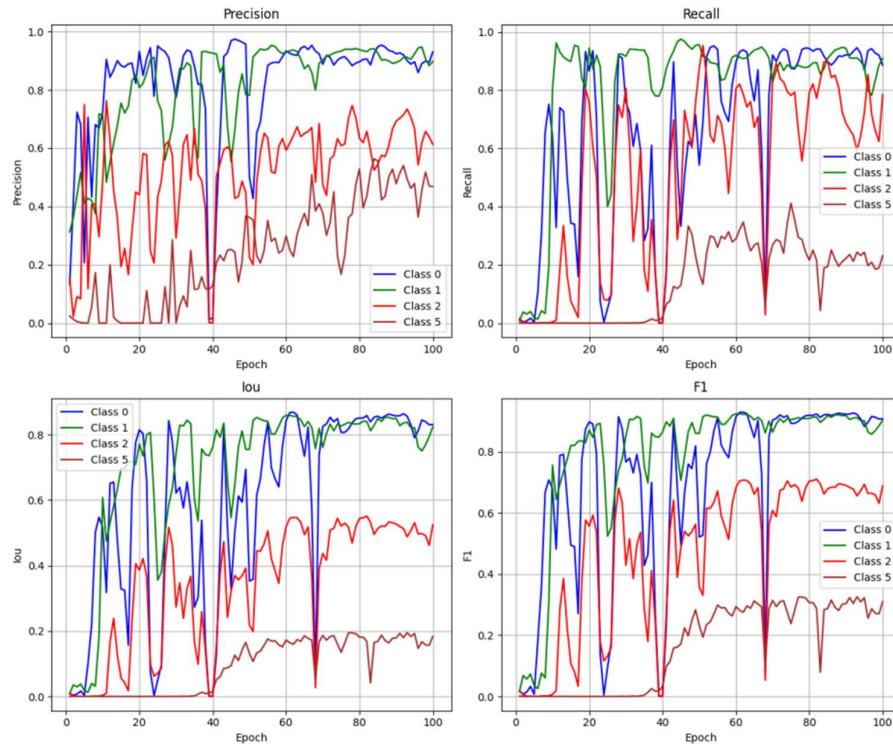


Fig 4 - Training and validation metrics curves

5. POST-PROCESSING AND SPECIALISED MODELS

Post-processing plays a critical role in restoring spatial coherence, enhancing output interpretability, and converting model predictions into formats suitable for downstream geospatial analysis. This section describes the key post-processing stages used in our pipeline.

5.1. TILE MERGING AND RECONSTRUCTION

During preprocessing and model inference, large LiDAR datasets are divided into smaller tiles (e.g., 250×250 m) to reduce memory usage and enable parallel processing. After classification or segmentation is performed on these individual tiles, it becomes necessary to **reconstruct the full spatial layout** of the original scene.

This reconstruction is performed via a **tile stitching algorithm**, which:

- Uses the embedded coordinates in tile filenames (e.g., 453000_3974000.png) to position them correctly on a grid.
- Aligns tiles row-wise and column-wise based on their coordinates.
- Groups and assembles tiles per data layer, handling multiple input channels or prediction types.
- Automatically fills in missing tiles with black (empty) placeholders to maintain spatial consistency.

The script supports two modes of operation:

1. **Grouped Merging** — assembles images layer-wise or by category.
2. **Full Mosaic Merging** — compiles all tiles into a single output mosaic.

This step is crucial for creating high-resolution visualizations and for integrating model outputs into geographic information systems (GIS).

5.2. OBJECT CONTOUR EXTRACTION AND VECTORIZATION

To make segmentation results suitable for GIS analysis and visual validation, we extracted **object boundaries** and converted them into both raster and vector formats.

PROCESSING STEPS:

1. **Image Loading and Binarization**
The model output (prediction mask) is converted into a binary image. Depending on the background and object color, thresholding or inversion may be applied to highlight the segmented objects.
2. **Contour Detection**
Using OpenCV's `cv2.findContours`, external contours of objects are identified. Contours with areas smaller than a predefined threshold (`min_area`) are discarded to suppress noise and irrelevant artifacts.

3. Visual Overlay Generation

Filtered contours are rendered on a white canvas:

- Filled with gray to show object area
 - Bordered in red for clear outline visualization
- This output is saved as a `.png` image for manual inspection.

4. Shapefile Creation (Vectorization)

Each valid contour is converted into a polygon and stored in a **Shapefile (.shp)** with a unique ID attribute.

These vector files are compatible with standard GIS software (QGIS, ArcGIS) and facilitate further spatial analysis.

5.3. 2D-TO-3D SEGMENTATION RESULT MAPPING

Following segmentation or classification in 2D (raster) space, the predicted labels must be **mapped back to the original 3D point cloud** (LAS format). This reverse mapping ensures the enriched point cloud retains both spatial and semantic information.

The implemented approach uses **Nearest Neighbor Interpolation**:

- The original LAS file is loaded, containing 3D points (x, y, z).
- A corresponding 2D prediction mask (RGB image) is loaded, where each pixel color encodes a predicted class.
- XY coordinates from the LAS file are normalized to align with the 2D image grid.
- For each 3D point, the closest pixel in the 2D mask is found.
- The pixel color is mapped to a class using a predefined `class_colors` dictionary.

Finally, each point in the LAS file is updated:

- The **classification** field is set based on the predicted class.
- The **RGB values** are assigned using 16-bit encoding, as required by the LAS specification.

The enhanced LAS file is saved with updated attributes, making it compatible with point cloud visualization and analysis tools.

6. EXPERIMENTS AND RESULTS

6.1 DATASET AND EXPERIMENTAL SETUP

To evaluate the effectiveness of the proposed method, we used the **STPLS3D** dataset, which contains diverse urban LiDAR scans with point-wise semantic labels. The dataset includes various urban scenes such as residential areas, streets, and industrial zones. Each point is annotated with one of 19 semantic classes.

The data was divided into tiles of size **500×500** points. For each tile, a multi-channel tensor was generated, encoding height-based and geometric features. The input tensor size for the model was fixed at **(512, 512, C)**, where **C** is the number of channels (typically 9–12 depending on configuration).

The model was trained using the Adam optimizer with a learning rate of $1e-4$ and a batch size of 8. Cross-entropy loss was used as the objective function. Training was performed on a single GPU with 12 GB of VRAM.

6.2 EVALUATION METRICS

To evaluate performance, we used the following standard metrics for semantic segmentation:

- **Overall Accuracy (OA):** The ratio of correctly classified points to the total number of points.
- **Mean Intersection over Union (mIoU):** Averaged IoU across all classes.
- **Per-Class Accuracy:** Accuracy calculated individually for each semantic class.

6.3 RESULTS

The model demonstrated high performance across most urban classes, including buildings, roads, vegetation, and parked vehicles. Table 1 summarizes the class-wise IoU scores on the validation set.

Figure 5 shows the output of the model performing binary segmentation.



RGB color representation of LiDAR point cloud data for St. Pauls Bay area



Building segmentation results for St. Pauls Bay area showing detected building footprints

Fig 5 - Binary classification

Figure 6 shows the output of the model performing multi-class segmentation.



Fig 6 - Multi-class segmentation results for St. Pauls Bay area with 20 classes

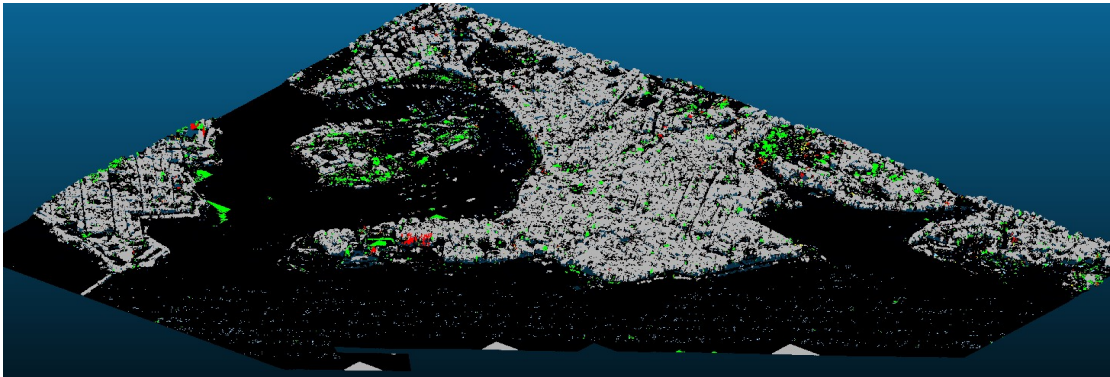


Fig 7 - 3D segmentation

Table 1. Per-Class IoU on STPLS3D Validation Tiles

Class	IoU (%)
Ground	84.3
Building	82.9
Low Vegetation	57.5
Car	21.2

6.4 QUALITATIVE RESULTS

Qualitative analysis confirms that the model generalizes well across different urban environments. In particular, boundaries between objects are well preserved due to the high-resolution projection and detailed geometric features. Figure 3 shows examples of predictions visualized in 3D, with points colored according to class.

Misclassifications were most common in ambiguous areas such as low vegetation vs. grass, and thin structures like wires and poles. These errors are partly due to limitations in resolution during projection and class imbalance in the dataset.

REFERENCES

1. Chen, S., Shi, W., Zhou, M., Zhang, M., and Chen, P., (2020). Automatic Building Extraction via Adaptive Iterative Segmentation With LiDAR Data and High Spatial Resolution Imagery Fusion, in *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 2081-2095, 2020, doi: 10.1109/JSTARS.2020.2992298.
2. Mo, Y., Guo, Z., Zhong, R., Song, W., & Cao, S. (2024). Urban Functional Zone Classification Using Light-Detection-and-Ranging Point Clouds, Aerial Images, and Point-of-Interest Data. *Remote Sensing*, 16, 386. <https://doi.org/10.3390/rs16020386>
3. Zhang, Jiancheng & Pi, Rendong & Ma, Xiaohong & Wu, Jianqing & Li, Hongtao & Yang, Ziliang. (2021). Object Classification with Roadside LiDAR Data Using a Probabilistic Neural Network. *Electronics*. 10. 803. 10.3390/electronics10070803.