Alma Mater Studiorum · University of Bologna

School of Science
Department of Physics and Astronomy
Master Degree in Physics

TITLE

Supervisor:
Prof. Mirko Degli Esposti

Submitted by:
Matteo Falcioni

Academic Year 2023/2024

Running over the same old ground,
what have we found?

Wish You Were Here - Pink Floyd

Abstract

This thesis explores the integration of deep learning techniques within Bologna's Urban Digital Twin project, focusing on the semantic classification of LiDAR point cloud data. By leveraging the high-resolution LiDAR dataset acquired for Bologna, we developed and evaluated a Multi-Scale Convolutional Neural Network (MCNN) tailored for classifying urban environments. Inspired by prior research, the MCNN architecture was adapted to process multi-channel feature images, enabling robust analysis of spatial patterns at varying scales.

Extensive experimentation on feature selection and window sizes resulted in an optimal configuration that achieved a classification accuracy of 94% across six urban classes. However, challenges such as computational inefficiencies and class imbalances were identified, alongside limitations in distinguishing visually similar categories like roads and railways.

Comparative analysis with a Random Forest benchmark highlighted the strengths and trade-offs of deep learning approaches in LiDAR data classification, underscoring the potential for hybrid methodologies. This work not only advances the application of AI in urban planning and digital twin projects but also sets the stage for future exploration of segmentation-based preprocessing, data augmentation, and even generative AI techniques to enhance LiDAR data utility.

The findings pave the way for innovative applications of artificial intelligence in urban digital twin initiatives, contributing to the broader vision of sustainable and intelligent city development.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Bologna's Digital Twin

The rapid pace of technological innovation is blurring the line between the physical and digital worlds. This process is driven by a growing need for optimized production, enhanced operational efficiency, and real-time decision-making capabilities across various domains. Central to this transformation is the concept of the Digital Twin (DT) [1].

A Digital Twin, broadly defined, is a digitally created virtual model of a physical object or system that leverages data to emulate real-world behavior [2]. By facilitating bidirectional communication between physical and digital entities, DTs enable iterative decision-making, advanced data analysis, and optimized control. This pervasive influence underscores the role of DTs as a cornerstone of the Industry 4.0 era [3].

Digital Twin technology has recently started to leverage more and more the combined power of Artificial Intelligence (AI) and Machine Learning (ML) [4]. While AI enables advanced analytical capabilities and autonomous control, ML facilitates the continuous learning process by detecting patterns, making predictions, and adapting based on new data [2].

As Digital Twin technology continues to evolve, its applications have expanded beyond individual industrial systems to encompass entire urban environments. The rise of the smart city concept in the 2010s introduced the idea of leveraging digital technologies to create more efficient, sustainable, and intelligent cities [5]. However, despite significant advancements, many smart city implementations have struggled to achieve dynamic, real-time interaction between digital models and physical urban environments.

Urban Digital Twins (UDTs) have emerged as a transformative solution to this challenge. By creating comprehensive, real-time digital replicas of cities, UDTs enable continuous monitoring, simulation, and optimization of urban systems. Unlike traditional smart city models, which often lack direct feedback mechanisms, UDTs maintain an up-to-date digital representation of the city by automatically integrating data from a wide

range of sources, including sensors, IoT devices, and satellite imagery [6].

In this context, the development of Urban Digital Twins represents a significant leap forward in the pursuit of sustainable, livable, and resilient cities. By combining the power of real-time data, AI-driven analytics, and digital modeling, UDTs offer an unparalleled tool for addressing the complex challenges of modern urbanization. This thesis builds on these advancements by contributing to Bologna's Urban Digital Twin project [7], focusing on AI-driven methodologies that enhance data analysis and predictive capabilities.

In the context of this work, a special acknowledgment goes to CINECA, whose support was instrumental. Their collaboration in Bologna's Digital Twin project, together with the Comune di Bologna and the University of Bologna, has been pivotal for its development. Regarding the scope of this thesis, they provided access to the Leonardo high-performance computing (HPC) infrastructure, enabling efficient training of the classifier model for the Bologna's LiDAR dataset developed during the course of this research.

1.2 LIDAR Technology

Light Detection and Ranging (LiDAR) is a remote sensing technology that measures distances by illuminating a target with laser light and analyzing the reflected pulses. This method allows for the generation of precise, three-dimensional data about objects and their surroundings, providing detailed information on their shape and surface characteristics [8].

LiDAR systems operate by emitting mono-frequency laser pulses and measuring the time it takes for the light to return after hitting an object. Time, intensity and waveform of the returned signal are used to calculate the interval or distance between objects, together with optical properties such as reflection and absorption [9].

In terms of the working principle, LiDAR is similar to a traditional microwave radar. The difference is that the former uses laser (e.g., $532 - nm$ or $1064 - nm$ wavelengths) as the carrier to measure the distance and orientation and to identify the target through the position, radial velocity, target scattering, and other characteristics [10].

Depending on the geometry of illuminated surfaces, several backscattered echoes can be recorded for a single pulse emission. This is particularly interesting in forested areas, since lidar systems can measure both the canopy height and the terrain elevation underneath at once, contrary to photogrammetric techniques [11] [12]. Fig. 1.1 shows the principle of multi-return LiDAR.

LiDAR systems can be broadly categorized into satellite, airborne, and terrestrial types, each designed for specific applications and environments.
Satellite-based LiDAR systems, such as NASA's ICESat-2 [13], are used for large-scale monitoring of Earth's topography, vegetation, and atmospheric properties. They are particularly effective for global studies due to their extensive coverage and ability to operate across different terrains [14].

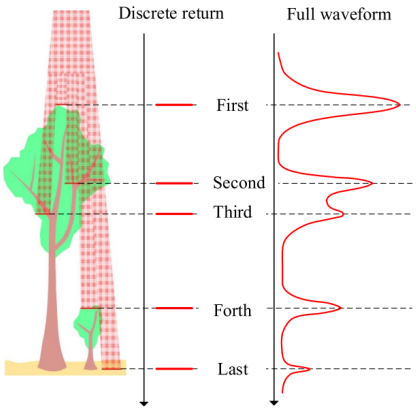


Figure 1.1: Depiction of the interaction between a laser beam from a multi-return system and a tree canopy [12].

Airborne LiDAR Systems (ALS) [15], mounted on aircraft or drones, are employed in applications like topographic mapping [16], forest monitoring [17], archaeological site documentation [18] and urban planning [19]. They offer high resolution and accuracy, making them suitable for regional-scale projects where detail is critical.
Terrestrial LiDAR systems, either static or mobile, operate from ground-based platforms and are used for a variety of applications: some of them unique to this category - due to their ability of capturing highly precise 3D models of localized areas - such as autonomous driving [20]; others overlapping and integrating with ALS frameworks, such as forest mapping [21].

As depicted in Fig. 1.2, an airborne LiDAR system integrates multiple components to capture detailed geospatial data [22].
The system is mounted on an airborne platform, such as an aircraft or helicopter, which carries a LiDAR sensor over the area of interest. The sensor emits short laser pulses and records the return signals. It then measures the travel time of the pulses, and associates each return with the corresponding Global Navigation Satellite System (GNSS) time and the scan angle of the transmitted pulse.
A GNSS receiver, working in tandem with a ground-based GNSS base station, determines the position of the aircraft at each epoch, typically at 1-2 Hz. Complementing this [23], an Inertial Measurement Unit (IMU) records the aircraft's accelerations and orientations at a much higher frequency (e.g., 400 Hz). An onboard computer synchronizes and timestamps the data from these sensors using GNSS time. In many cases, a medium-format digital camera ($60-100$ MP) is also included to provide additional color information about the terrain.
The geolocation process combines LiDAR sensor data (laser ranges and scan angles) with the aircraft's trajectory (determined using GNSS and IMU measurements) to compute the precise coordinates of ground points in a global reference system, the ECEF WGS-84

[24]. The resulting coordinates can be transformed into a desired horizontal or vertical datum for analysis.

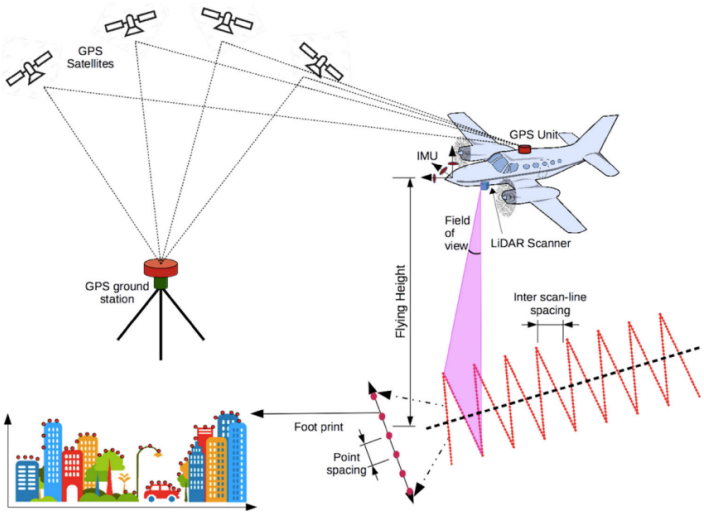


Figure 1.2: Airborne LiDAR data capture. Red dots show LiDAR data on terrain objects [22].

The data collected by LiDAR systems is commonly processed into point clouds (PC), dense collections of points in a three-dimensional coordinate system that represent the scanned environment.
Each point in a PC represents the x, y and z-coordinates of the sensed object, along with other attributes (e.g., timestamp, laser intensity, etc.) that are dependent on the LiDAR sensor and the presence or absence of integrated global positioning and/or camera instrumentation [25].
See Fig. 1.3 for a visualization of LiDAR point cloud data from the city of Chicago (USA).

Digital Surface Models (DSM) and Digital Terrain Models (DTMs) are essential geospatial products derived from such point cloud data.
A DSM captures the elevation of the Earth's surface including all above-ground features, such as vegetation, buildings, and other structures, excluding multiple returns. It represents the first reflective surface encountered by the sensor [27].
In contrast, a DTM focuses solely on the bare-earth surface, excluding all above-ground features like buildings and vegetation. A DTM is often created by processing DSM data through classification and filtering techniques to remove non-ground returns [28].
See Fig. 1.4 for a visual depiction of the difference between DTM and DSM.

LiDAR offers several advantages over traditional remote sensing methods. It provides centimeter-level accuracy, essential for high-precision applications like urban planning

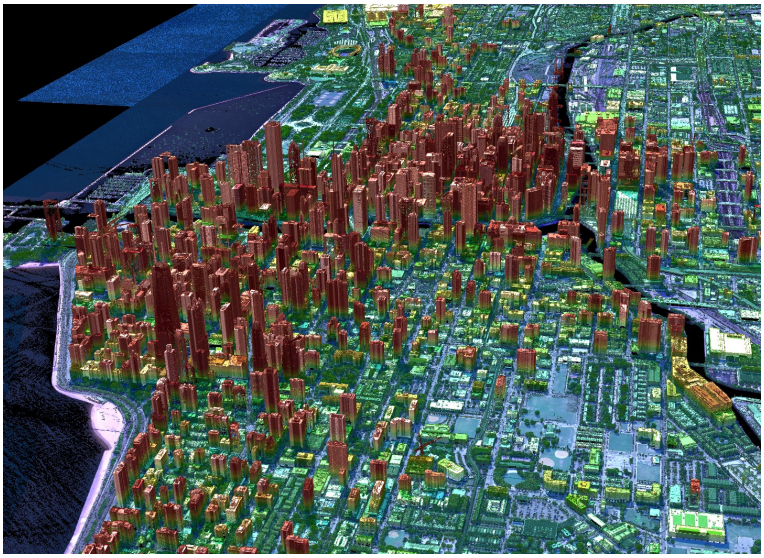


Figure 1.3: Lidar point cloud over Chicago (USA) [26].

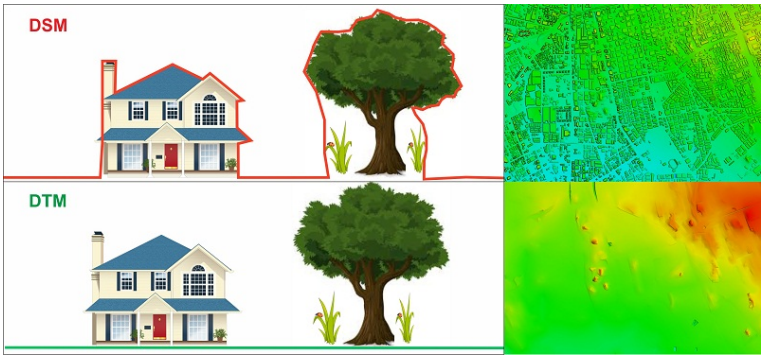


Figure 1.4: Visual comparison between a DSM and a DTM [29].

and powerline monitoring [30]. The technology's ability to penetrate dense vegetation using multi-return laser pulses makes it ideal for forestry and terrain mapping. Furthermore, its capacity to survey large areas rapidly significantly reduces time and resource demands compared to manual methods [31].

However, despite its revolutionary potential, LiDAR technology faces notable challenges. The high cost of LiDAR systems and the substantial data processing requirements often limit its accessibility, particularly for smaller organizations or projects with constrained budgets [31]. Additionally, its performance can be compromised by adverse weather conditions, such as heavy rain or fog, which scatter or absorb laser pulses, affecting data accuracy and coverage [32].

1.3 Bologna's LiDAR Dataset

The Comune di Bologna commissioned airborne LiDAR data acquisition from Compagnia Generale Ripreseaeree (CGR) [33], which used the CityMapper-2 sensor manufactured by Leica Geosystems [34].
This cutting-edge system integrates LiDAR with optical imaging, featuring six digital cameras: two nadiral - one RGB and one near-infrared - and the remaining four oblique at 45°. Such apparatus provides a pixel size of 3.76μm and a resolution of $14,192 \times 10,640$ pixels (150MP). The LiDAR unit operates at a wavelength of 1064nm, with a maximum acquisition frequency of 2MHz and the capability to capture up to 15 echoes per laser beam. These features make the dataset exceptionally suited for urban and terrain analysis.

The flight campaign ensured an average ground sample distance of 5cm and a LiDAR point density exceeding 20 points/m^2, a level of precision suitable for detailed mapping of ground and off-ground features. Data were pre-processed by CGR to include RGB and NIR indices for each point, and to classify points into ground and off-ground categories exploiting Axelsson's progressive densification algorithm [35]. Such classification allowed for the creation of both Digital Terrain Models (DTMs) and Digital Surface Models (DSMs).

The dataset is organized into LAS files [36] containing classified point clouds and associated metadata, while the additional DTMs and DSMs are stored as binary files.
The municipality is divided into 654 tiles, each covering an area of 500m × 500m. The tiles enable efficient processing and analysis of specific regions within Bologna.
In Fig. 1.5 you can find the subdivision of Bologna's municipality in tiles, and in Fig. 1.6 a tile visualized using CloudCompare software [37].

In addition to the original dataset features provided by CGR, additional features were computed to enhance the dataset's utility for more advanced analyses [38]. These computed features were derived from the provided LiDAR attributes and the RGB-NIR indices included in the data, following common procedures in literature [39–43]. For example, vegetation indices such as the Normalized Difference Vegetation Index (NDVI) and Normalized Difference Water Index (NDWI) were calculated using the near-infrared and red channels, highlighting vegetation and water areas, respectively. Similarly, geometric attributes like elevation above ground (N_h) were derived by subtracting Digital Terrain Model (DTM) elevations from the original point heights. More advanced features, such as planarity and sphericity, were extracted from the covariance matrix of neighboring points to quantify local geometric patterns in the point cloud. For such local features requiring a neighborhood of points to be computed, a spherical neighborhood of radius 1m was selected.
All features contained in the dataset are listed in the comprehensive Tab.1.1, and their computation is explained in detail in appendix A.

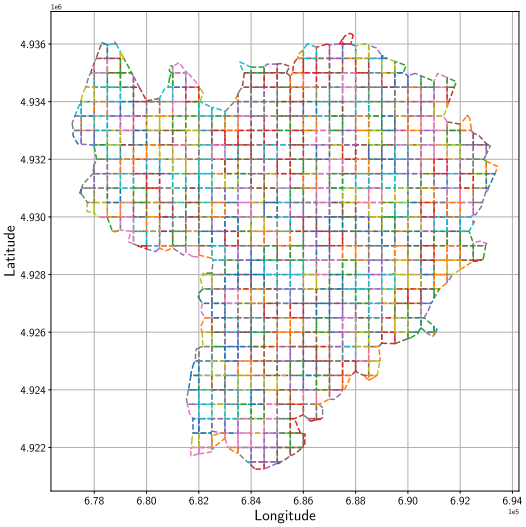


Figure 1.5: Depiction of Bologna's municipality subdivision into tiles. There is no data outside the municipality boundaries.

These additional features significantly expand the dataset's analytical potential, providing detailed insights into both the physical and spectral properties of the scanned environment.

1.4 Deep Learning Applications On LiDAR Data

The aim of this work was to explore the use of deep learning methods for the processing and classification of Bologna's LiDAR dataset. Machine learning models, such as Random Forests [44], had already been used for point labeling [38]. However, Neural Networks, with their ability to learn hierarchical representations directly from raw data, offered a promising alternative to to enhance the analysis of LiDAR data. Despite their growing prominence in computer vision and remote sensing, deep learning techniques had not been systematically applied to Bologna's LiDAR dataset prior to this study.

The application of neural networks and deep learning to LiDAR data has evolved in parallel with the development of novel architectures, encompassing both airborne and non-airborne datasets. Early efforts, such as Prokhorov et al. [45], utilized recurrent neural networks (RNNs) to classify vehicles and non-vehicles by modeling the temporal sequence of laser returns—one of the first examples of deep learning in this domain.

Subsequent developments addressed the challenge of processing unordered 3D point clouds directly. Qi et al. [46] introduced PointNet, a neural network architecture that bypassed the need for voxelization or image projections by leveraging global feature aggregation. This approach laid the groundwork for more complex models, such as

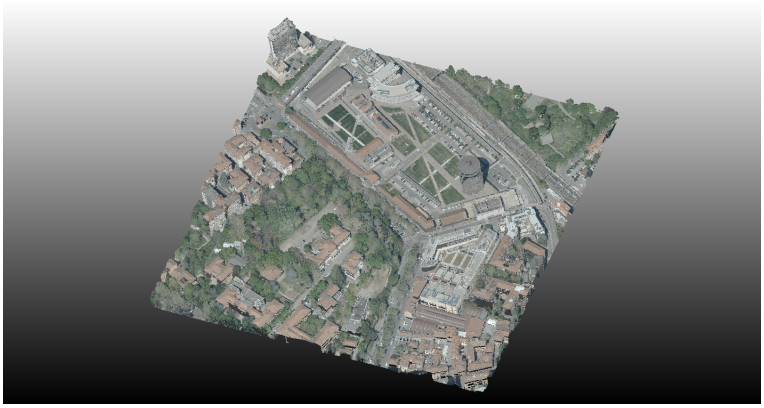


Figure 1.6: Visualization of one of Bologna's tiles, using CloudCompare.

PointNet++ [47], which extended PointNet with a hierarchical structure to capture fine-grained geometric features and adapt to varying point densities.

As neural network architectures advanced, convolutional neural networks (CNNs) emerged as a dominant approach. Yang et al. [48] proposed a multi-scale CNN (MCNN) for classifying airborne LiDAR data, transforming the inherently 3D data into 2D feature images, leveraging segmentation to reduce computational time. Similarly, Hang et al. [49] introduced a coupled CNN framework that effectively fused hyperspectral and LiDAR data, demonstrating how complementary data sources can enhance land-cover classification through both feature- and decision-level integration.

More recently, attention-based architectures have opened new possibilities for processing LiDAR data. For instance, Zhao et al. [50] developed Point Transformer, a novel architecture tailored for 3D point cloud analysis. By employing self-attention, the model excels in tasks like semantic segmentation and shape classification, achieving state-of-the-art results. Just recently, He et al. [51] introduced SegPoint, a framework that integrates Large Language Models (LLMs) with Transformer-based architectures for 3D segmentation. The model employs multi-head self-attention mechanisms to capture both local geometric details and global contextual relationships within point clouds, and combines these capabilities with the reasoning power of LLMs. This way, SegPoint can interpret natural language instructions to perform tasks such as semantic segmentation, object referencing, and open-vocabulary segmentation.

Parallel to these advancements, generative models have gained traction in LiDAR data synthesis and enhancement, particularly in applications related to autonomous driving. Caccia et al. [52] leveraged Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) to synthesize LiDAR point clouds, in order to recover data from noise and missing regions. This was made possible by mapping 3D LiDAR terrestrial scans into 2D spherical representations, thus enabling the use of GAN and VAE's existing structures on 2D data. Meanwhile, Nakashima et al. [53] introduced a denoising

diffusion probabilistic model (DDPM) to upsample sparse LiDAR data, demonstrating improved semantic and structural consistency. These generative approaches not only enhance data quality but also expand the utility of LiDAR datasets for robust downstream applications.

Table 1.1: Summary of Provided and Computed Features in the Dataset.

Category	**Feature**	**Description**
Provided Features	Intensity	Fraction of energy returned to the detector.
	Return Number	Index of the return signal for the emitted pulse.
	Number of Returns	Total returns from a single emitted pulse.
	Classification	Ground or off-ground classification.
RGB-NIR Features	NDVI	Highlights vegetation using near-infrared and red bands.
	NDWI	Detects water and vegetation using near-infrared and green bands.
	SSI	Differentiates dark shadows and vegetation.
Height Features	N_h	Elevation above the ground (difference from DTM).
	ΔZ	Vertical difference between the analyzed point and the lowest in its neighborhood.
	MAD	Median of absolute height differences within the neighborhood.
	ΔZ_{fl}	Vertical gap between the first and last returns of the same beam.
Covariance Features	Planarity	Measures flatness of the local point distribution.
	Sphericity	Indicates isotropic distribution of points.
	Linearity	Detects alignment of points along a line.
	Entropy	Quantifies randomness in point distribution.
Plane Features	θ	Normal direction of the local plane.
	σ_θ^2	Variance of plane inclination.

Chapter 2

MCNN Classifier

Convolutional neural networks (CNNs) are a specialized type of deep learning model designed to extract features from data with convolution structures [54]. The main benefit of CNN compared to its predecessors is that it automatically identifies the relevant features without any human supervision [55]. For a thorough description of the architecture of CNNs and their convolutional operations, see appendix B.

Given the complexity and multi-dimensionality of Bologna's LiDAR dataset, CNNs offered a promising framework for its analysis. Their ability to process spatially structured data makes them particularly well-suited for tasks such as feature extraction and segmentation, where understanding geometric relationships within the data is critical.

Several studies had already been conducted on the applications of CNNs to LIDAR data classification [49, 56–58]. In particular, we drew inspiration from the pioneering works of Yang et al. [59], which laid the groundwork for applying CNNs to this task and demonstrated the potential of deep learning models in lidar-based classification.

2.1 Feature Image Generation

In [59], the authors introduced an innovative approach to leverage CNNs for lidar data classification. For each point in the point cloud requiring classification, a "feature image" was generated. This consists of a grid centered around the target point with a specified spatial resolution - in this case, a resolution of 128, as detailed in the study. Each cell in this grid was populated with the features of the nearest point to the cell center.

If we label the coordinates of each point P_k in the point cloud as the triplet $(X_{p_k}, Y_{p_k}, Z_{p_k})$, then the coordinates of the cell centers will be defined as:

$$\begin{cases} X_{ij} = X_{p_k} - (64.5 - j) * w \\ Y_{ij} = Y_{p_k} - (64.5 - j) * w \\ Z_{ij} = Z_{p_k} \end{cases} \tag{2.1}$$

where i denotes the row number, j denotes the column number and w is the width of the cell. We can then look for the nearest neighbor point of each cell center, and assign its features to the cell.

The authors utilized the Vaihingen dataset [60] and computed a specific set of features from the data. They calculated the eigenvalue-based features planarity and sphericity P_λ and S_λ, the variance of the normal vector angle of the local plane from the vertical direction σ_z^2, and the normalized height above DTM, H_{above}. All features available to their study are listed in 2.1.

Table 2.1: Available features in [59].

Feature name	Notation
Intensity	I
Height	H_{above}
Planarity	P_λ
Sphericity	S_λ
Variance	σ_z^2

They then chose to encode such features into three channels, allowing the resulting image to be represented as an RGB-like image, to be fed to the Convolutional Neural Network.
Specifically, they encoded the feature values as

$$\begin{aligned}\text{RED} &= [255 * S_\lambda * \sigma_z^2] \\ \text{GREEN} &= [I * P_\lambda] \\ \text{BLUE} &= [255 * H_{\text{above}}]\end{aligned}$$

In this way, we get a 128×128 RGB feature image. The steps of feature images generation in [59] are synthetized in Fig. 2.1.

In our work, we decided to draw inspiration from the above procedure, modifying it slightly. First and foremost, we discarded the choice of encoding feature values in just 3 channels, for several reasons. First of all, it allowed for wider experimentation of feature combinations for feature image generation, eliminating the need to combine similar features together in a single channel. Secondly, since we had at our disposal a far more rich and complex dataset compared to the Vaihingen one, we avoided adding more complexity to the raw data by combining features together, in hope that the CNN model could benefit from this freedom, given its renowned capability to extract relevant information from raw data. Finally, this approach allowed our feature images to represent visually the section of point cloud they were encoding, since they where not actual 3-channel RGB-like images, but multi-channel images, where each dimension encoded

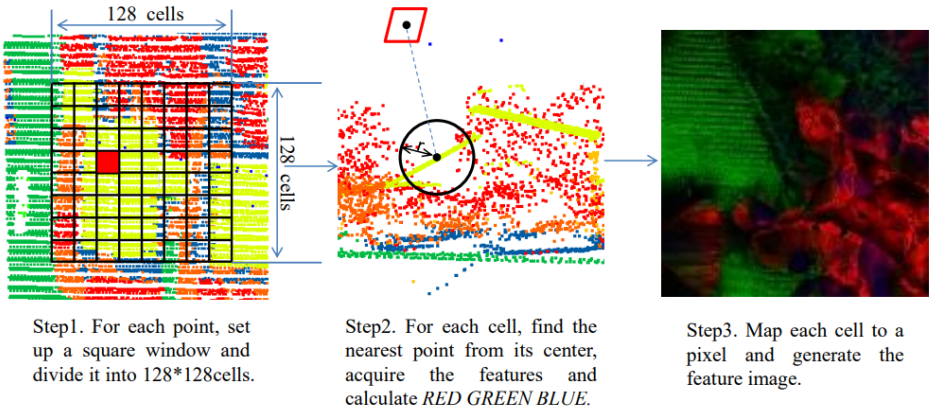


Figure 2.1: Steps of the feature image generation in [59].

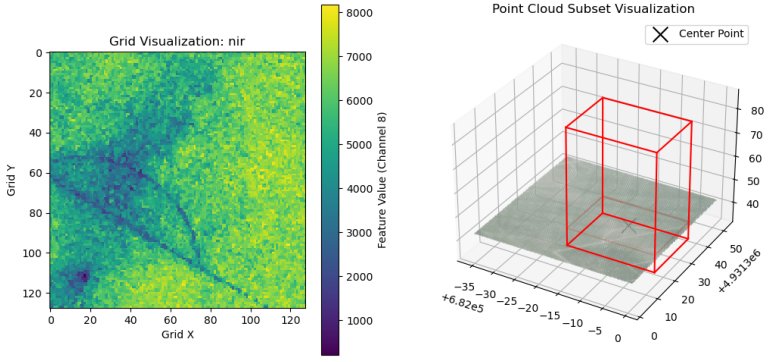


Figure 2.2: NIR channel visualization (left) of a portion of point cloud (right).

values from a selected feature from the data. This is shown in Figs. 2.2 and 2.3. All other specifics of feature images generation were left unchanged: again, for each point in the point cloud, we produced a grid of dimension 128×128, associating to each cell center of the grid the selected features of its nearest neighbor point.

Of course, this process is inherently time-consuming, since we need to search, for each point cloud point, the nearest neighbor between all the other points in the point cloud. A task that would be unfeasable without the use of efficient search algorithms. For this reason, scikit-learn's KDTree search algorithm [61] was employed, which is a commonly used procedure in LiDAR data processing [62].

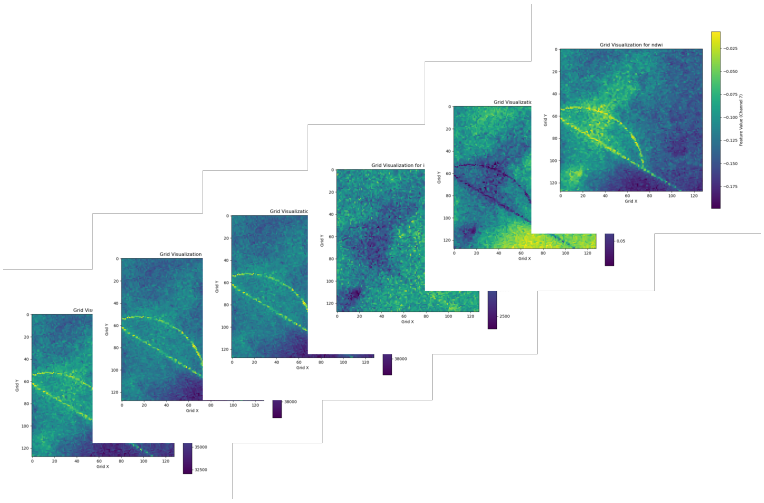


Figure 2.3: Different channels of the same feature image.

2.2 Network Architecture

As already described above, these feature images were designed specifically to be the input to our Convolutional Neural Network. We will now briefly go over the architecture of the model.
In [59], the authors introduced the structure depicted in Fig. 2.4: a CNN composed of 5 sequential convolutional layers, where each block - except for the fourth - consists of:

(i) a 3×3 Convolutional layer with padding of 1;

(ii) a Batch Normalization layer;

(iii) a ReLU activation function;

(iv) a max pooling layer to downsample the spatial dimensions by half,

with the fourth block not being equipped with the max pooling layer. Subsequently we have 2 fully connected layers, each followed by Batch Normalization and a ReLu activation function, in order to get the class scores as output.

This is the basic structure of what we will refer to, from now on, as a "single scale" CNN. As a matter of fact, Yang et al. [59] upgraded their model structure in their following work on the matter [48], implementing what they called a "Multi-Scale" Convolutional Neural Network (MCNN).

The idea is actually simple, but very effective: in order to capture the complexity of the LIDAR data at different scales, three single scale CNNs where "fused" together, each of them working with a different spatial dimension of the input feature images. The resulting structure is depicted in Fig. 2.5

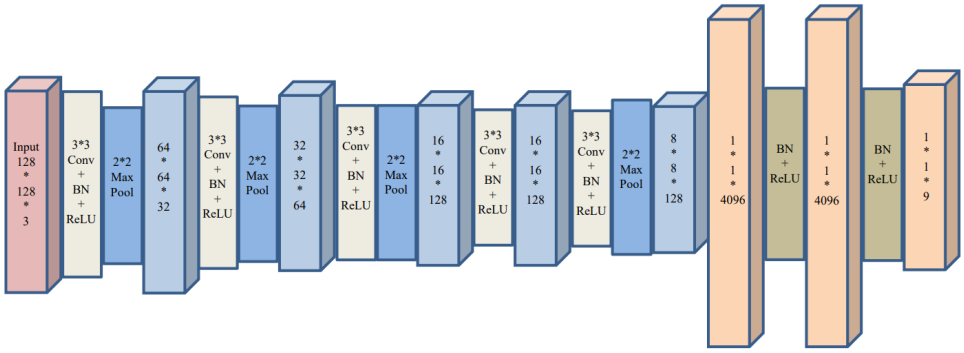


Figure 2.4: The architecture of the CNN described in [59].

Again, we drew inspiration from the work described in [48], but we slightly modified the architecture in order to be able to work with N-channel input feature images, and also with a different, variable number of output classes for our classification purposes, which we will denote as M. A depiction of the resulting architecture of our work is shown in Fig. 2.6.

In [48, 59], the authors implemented their model using Caffe [63], a deep learning framework developed by the Berkeley Vision and Learning Center in 2014. At the time, Caffe offered significant advantages in speed and ease of use for computer vision tasks. However, its reliance on static model definitions and its reduced community support in recent years limit its applicability compared to modern frameworks like PyTorch and TensorFlow. These limitations prompted us to reimplement the model architecture described in these works using PyTorch's dynamic computation graph, specifically the `torch.nn` module [64]. This approach not only provided greater flexibility in model design but also leveraged PyTorch's active development and extensive ecosystem. All code is available in GitHub[1] .

Another difference that needs to be addressed is that, in their approach, Yang et al. [48] implemented a three-step region-growing segmentation method as a preprocessing step before generating feature images. This segmentation process grouped points in the point cloud into distinct segments based on criteria such as curvature, surface normal vectors, and intensity values; the method was effective in reducing the computational burden and improving classification accuracy by leveraging segment-level features. However, since accurate segmentation methods were still under active experimentation at the time of our study, we chose to skip the segmentation step and directly process individual points to evaluate the impact on the model's performance. This decision allowed us to focus on the efficacy of the neural network architecture without introducing potential errors from imperfect segmentation.

[1]link: https://github.com/MatteoFalcioni/Multiscale_Convolutional_Neural_Network

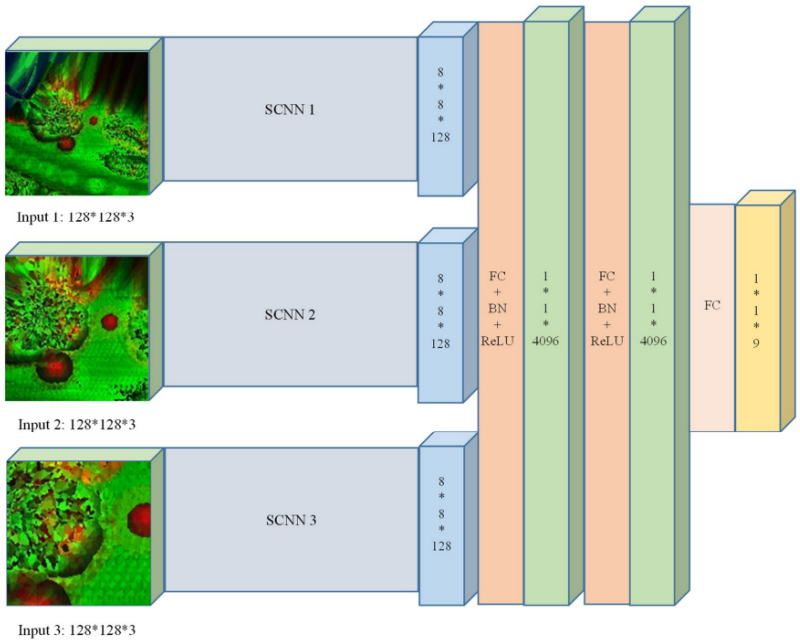


Figure 2.5: The architecture of the MCNN model described in [48].

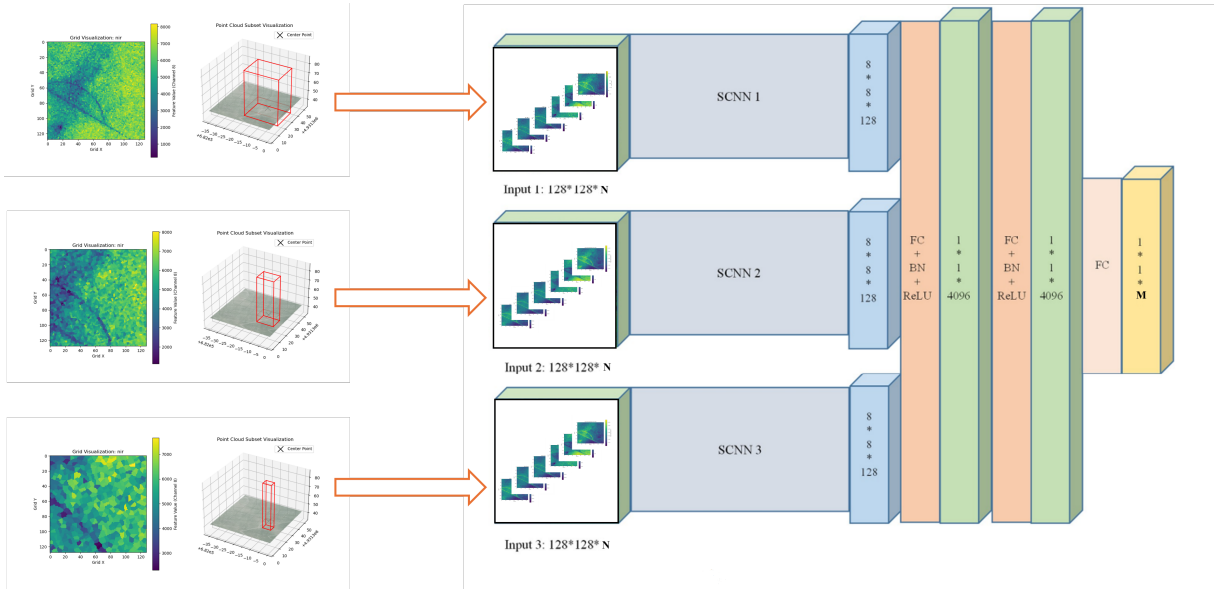


Figure 2.6: The architecture of the MCNN employed in this work. N represents the number of input channels, while M represents the number of output classes.

Chapter 3

Experimental Setup

The experiments were conducted to evaluate the effectiveness of the proposed Multiscale CNN architecture in classifying LiDAR data into 6 predefined categories. We evaluated our model on a subset of the Bologna Dataset, described in §1.3. The training and evaluation set consisted of a subsample of around 2.5 million point cloud points. The performance of the model was assessed using metrics such as accuracy, precision, recall, F1-score and a Confusion Matrix. The comparison with the ground truth labels for points was made possible thanks to the manual labeling of the tiles performed using CloudCompare. The proposed method was compared against a benchmark Random Forest model previously used for this task.

3.1 General Setup

The training and evaluation dataset was created using a subsample of approximately 2.5 million labeled points from the Bologna point cloud dataset. Ten tiles were carefully selected to represent a variety of areas within the city, ensuring a diverse and representative sample of different environments. Points were then extracted from these tiles and shuffled to ensure a randomized distribution of the data. Subsequently, 80% of the shuffled points were assigned to the training set, while the remaining 20% were allocated to the evaluation set. Finally, an additional shuffle was performed within each of the resulting subsets to further randomize their internal order. This approach ensured that both the training and evaluation datasets were representative of the overall data distribution, while capturing the variability present across different regions of Bologna.

The points in the dataset were manually labeled into six distinct classes, representing common urban features:

0: “Grass”;

1: “High Vegetation”;

2: “Building”;

3: “Railway”;

4: “Road”;

5: “Car”.

To address class imbalances inherent in the dataset, a very simple class rebalancing strategy was applied: overrepresented classes were downsampled to a fixed number of points. This ensured that the training process was not dominated by more frequent classes, allowing the model to learn to classify less common categories effectively. Tab. 3.1 shows the number of points in the training and validation datasets after rebalancing.

Table 3.1: Training and Validation Data Distribution.

Class	Training Samples	Validation Samples
Grass	500,000	124,999
High Vegetation	500,000	124,999
Building	500,000	124,999
Railway	420,322	105,081
Road	500,000	124,999
Car	260,032	65,008

During training, we aimed to replicate the hyperparameter choices from [48, 59] where applicable. For cases where the suggested values were unsuitable for our setup, we conducted a hyperparameter search to determine optimal values. The final hyperparameters used in training were as follows:

- **Epochs:** Training was conducted for a maximum of 10 epochs, with an early stopping criterion defined by a patience parameter of 2 epochs. This means training would halt if the validation loss failed to improve for two consecutive epochs, thereby mitigating the risk of overfitting.
- **Learning Rate:** An initial learning rate of 0.01 was used, coupled with a learning rate decay mechanism. The learning rate was reduced by a factor of 0.5 every 5 epochs to ensure stable convergence as training progressed.
- **Momentum:** A momentum value of 0.9 was employed to enhance optimization stability and accelerate convergence by dampening oscillations in the gradient descent updates.
- **Batch size:** A batch size of 32 was selected to leverage GPU parallelism effectively, balancing computational efficiency and memory constraints.

The choice of **window sizes** and **selected features** for feature image generation represented the most critical hyperparameters in this study. They were extensively tuned to achieve optimal results for both evaluation metrics and prediction performance.

Optimization of the network weights through error back-propagation was performed using Stochastic Gradient Descent, and the employed loss function was Pytorch's Cross Entropy Loss [65].

3.2 Exploration of Window Sizes

The selection of appropriate window sizes was a crucial step in our methodology, as it significantly impacted the quality of the generated feature images and, consequently, the performance of the model.

We began our experiments with small window sizes of 2.5, 5.0, and 10 meters, using a minimal set of features: intensity, NIR, Δz, planarity, and sphericity. However, the results were suboptimal, indicating that the chosen configuration was insufficient. To address this, we expanded the feature set to include additional attributes such as classification, return number, number of returns, NDVI, NDWI, SSI, linearity, MAD, and others, while maintaining the same window sizes. Despite this increase in feature diversity, the results showed negligible improvement.

Recognizing that window size might be a limiting factor, we shifted our focus to larger dimensions, testing windows of 5, 10, and 20 meters while returning to the original feature set of intensity, NIR, Δz, planarity, and sphericity. This approach yielded more satisfactory results, suggesting that larger window sizes provided a better representation of the data's contextual information. Encouraged by these findings, we further increased the window sizes to 10, 20, and 30 meters with the same feature set, which led to even greater improvements in model performance. However, when testing even larger window sizes of 20, 30, and 40 meters, the results deteriorated significantly, likely due to the inclusion of excessive and irrelevant spatial context, which overwhelmed the model and reduced classification accuracy. These experiments confirmed that window sizes need to be carefully balanced to capture sufficient spatial context without introducing unnecessary noise.

Yes, the paragraph is mostly well-written and correct, but it can be slightly refined to improve clarity and ensure the explanations are precise. Here's the revised version:

We evaluated the performance of the model using standard classification metrics provided by Python's *scikit-learn* package [66]: accuracy, precision, recall, F1-score, and support. Accuracy measures the overall correctness of the model, calculated as the ratio of correctly classified points (true positives and true negatives) to the total number of points in the dataset. While a high accuracy indicates that the model performs well overall, it does not reveal how well it performs on individual classes, particularly if the classes are imbalanced. Precision reflects the reliability of positive predictions, measuring

the proportion of true positives among all points predicted to belong to a class. High precision implies that the model makes few false positive errors. Recall, on the other hand, quantifies the model's ability to identify all actual members of a class, measuring the proportion of true positives captured out of all actual points in that class. Recall is particularly important when missing positive cases (false negatives) is costly. Finally, the F1-score provides a single metric that balances precision and recall, offering a more comprehensive assessment of performance, especially in scenarios with class imbalances or when precision and recall differ significantly.

Table 3.2 presents the detailed performance of the model across different window sizes values. Fig. 3.1 presents the confusion matrices for the respective models.

Yang et al. [48] employed window sizes of 6.4, 12.8, and 25.6 meters. In contrast, our optimal window sizes were significantly larger: 10, 20, and 30 meters. This might seem paradoxical at first, given that our point cloud data was far more densely distributed compared to the Vaihingen dataset. However, this can be explained by several factors. First, in such a densely distributed point cloud, capturing broader spatial relationships - such as identifying structures or patterns that extended beyond the immediate local neighborhood - is essential for classification tasks. Larger windows were needed to incorporate features from a wider area, enabling the model to effectively capture both local details and the necessary contextual information. Furthermore, smaller windows seemed to overemphasize local noise or overly specific details within the dense dataset, which negatively impacted classification accuracy. By increasing the window size, we were able to achieve a better balance between local detail and global context, resulting in more robust and reliable feature representations.

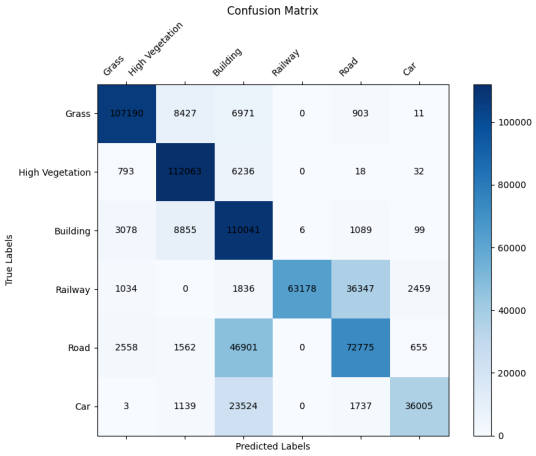
3.3 Exploration of Selected Features

Our investigation into feature selection began with a minimalist approach, using only intensity data. As anticipated, the classification results were suboptimal and far from satisfactory. Recognizing the need for richer input, we next incorporated the red, green, and blue (RGB) channels. This combination already led to significant improvements in classification accuracy, particularly highlighting the benefits of the so called *fusion* approach. This is a term used to refer to the combination of LiDAR raw data and RGB ortpohotos, which allow for the extraction of red, green and blue values and their integration in the dataset. It is a widely used method for improving tasks like 3D object detection and classification, and it's been shown to constitute a decisive enhancement in various deep learning applications involving LiDAR data [67–70]. Notably, Bologna's Lidar Dataset is inherently fusion, given that it already contains RGB values for each point.

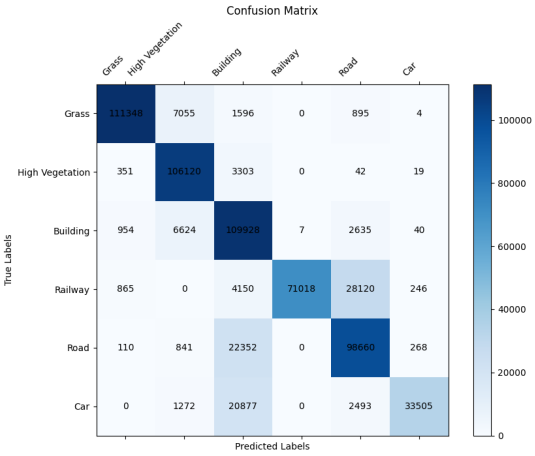
Building on this progress, we explored the potential of vegetation indices, i.e., NIR, NDVI and SSI. We started by adding to our previuos list of features the NIR index.

Table 3.2: Classification scores for window sizes exploration.

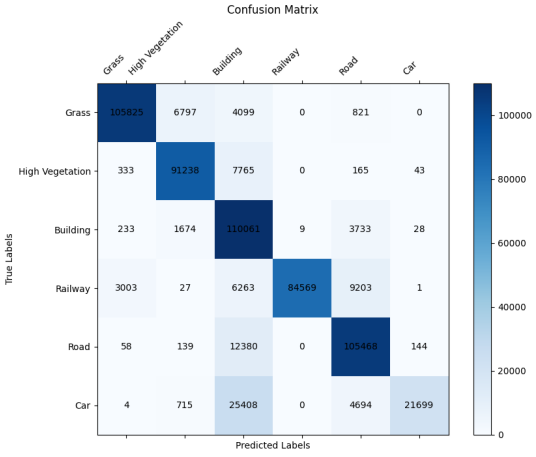
Window Sizes (m)	Class	Precision	Recall	F1-Score
2.5, 5.0, 10.0	0	0.93	0.87	0.90
	1	0.85	0.94	0.89
	2	0.56	0.89	0.69
	3	1.00	0.60	0.75
	4	0.64	0.58	0.61
	5	0.92	0.58	0.71
	Averages	**0.78**	**0.74**	**0.75**
5.0, 10.0, 20.0	0	0.98	0.92	0.95
	1	0.87	0.97	0.92
	2	0.68	0.91	0.78
	3	1.00	0.68	0.81
	4	0.74	0.81	0.77
	5	0.98	0.58	0.73
	Averages	**0.86**	**0.81**	**0.82**
10.0, 20.0, 30.0	0	0.97	0.90	0.93
	1	0.91	0.92	0.91
	2	0.66	0.95	0.78
	3	1.00	0.82	0.90
	4	0.85	0.89	0.87
	5	0.99	0.41	0.58
	Averages	**0.88**	**0.81**	**0.82**
20.0, 30.0, 40.0	0	0.79	0.94	0.86
	1	0.93	0.81	0.86
	2	0.35	0.69	0.46
	3	0.50	0.00	0.00
	4	0.66	0.86	0.74
	5	0.92	0.04	0.08
	Averages	**0.71**	**0.61**	**0.57**



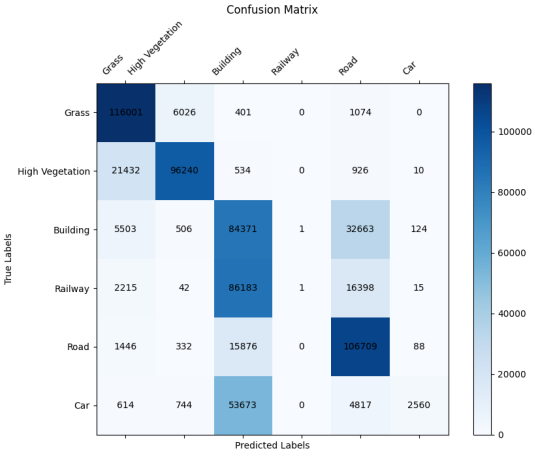
(a) Window Sizes: 2.5, 5.0, 10.0



(b) Window Sizes: 5.0, 10.0, 20.0



(c) Window Sizes: 10.0, 20.0, 30.0



(d) Window Sizes: 20.0, 30.0, 40.0

Figure 3.1: Confusion matrices for Different Models. Configuration **(c)**, with window sizes 10.0, 20.0 and 30.0, gave the best results.

This change alone yielded noticeable improvements, particularly in the classification of the railway class, which seemed to benefit from the additional spectral information. However, when NDVI and SSI were included alongside NIR, no further enhancements were observed. This outcome aligns with the known ability of convolutional neural networks to extract relevant information from simpler, less structured input data. Given that NIR and RGB already encapsulate much (if not all) of the information contained in NDVI and SSI, the CNN efficiently derived the necessary insights without requiring explicit redundancy in the features.

Next, we examined features from the "height features" group by adding the vertical difference Δz. Despite this combination seeming promising, the results did not show significant improvements, suggesting that height-based features alone were insufficient to meaningfully enhance classification performance.

To further refine our feature set, we turned to geometric descriptors derived from the covariance matrix, specifically the eigenvalues $\lambda_1, \lambda_2, \lambda_3$. Incorporating these features yielded notable improvements, particularly in the classification of the railway, road, and car classes. This demonstrated the value of local geometric patterns in capturing essential spatial relationships and enhancing classification scores.

Finally, we tested features related to the local plane orientation, specifically the normal direction angle θ and its variance σ_θ^2. Despite their theoretical potential, these features did not contribute to further improvements, suggesting that the model had reached its classification performance limit within the existing set of features.
Table 3.3 presents the comprehensive results of feature experimentation. In appendix C we present a more thorough description of the results, showing all scores for different features subsets and all of the respective confusion matrices.

Table 3.3: Feature subsets performance comparison. Additional features that led to performance improvements are shown in Green, while features that didn't optimize classification scores - and therefore were not included in the final subset - are shown in red.

Feature Subset	Accuracy	Precision	Recall	F1-Score
Intensity	0.59	0.67	0.77	0.68
+ RGB	0.88	0.92	0.88	0.90
+ NIR	0.91	0.94	0.91	0.92
+ NDVI, SSI	0.90	0.93	0.90	0.91
+ ΔZ	0.86	0.92	0.87	0.88
+ $\boldsymbol{\lambda_1, \lambda_2, \lambda_3}$	**0.94**	**0.92**	**0.89**	**0.90**
+ θ, σ_θ^2	0.89	0.92	0.89	0.90

Ultimately, our best-performing model utilized a relatively compact feature set: intensity, red, green, blue, NIR, and the eigenvalues $\lambda_1, \lambda_2, \lambda_3$. This finding underscores the efficiency of CNNs in extracting meaningful representations from raw data and high-

lights the inherent strengths of deep learning architectures in handling high-dimensional data. Rather than overwhelming the network with an extensive array of features, we found that a carefully selected subset was sufficient to achieve optimal results.

Chapter 4

Results

This chapter presents an in-depth evaluation of the proposed model's performance, highlighting its strengths and limitations in classifying LiDAR data.

We begin by discussing the predictions of the best-performing model on a set of ten validation tiles, examining both its successes and shortcomings. Particular attention is given to misclassifications, as they offer valuable insights into potential areas for improvement.

Next, we address the challenge of overfitting, which, despite various precautions, persists to some degree. This discussion explores potential underlying causes, such as feature representation and dataset characteristics.

Subsequently, we analyze the computational efficiency of the model, identifying bottlenecks and highlighting the trade-offs between accuracy and inference time.

4.1 Predictions

To evaluate the model's performance in real-life scenarios, ten tiles were carefully selected from Bologna's LiDAR Dataset. These tiles were chosen to represent the diverse environments of the city of Bologna, encompassing both urban and rural areas to ensure a comprehensive assessment. In the following, we present the best model predictions on some of these tiles, retained the most explicative of the model's strength and weaknesses.

Before visual inspection of the tiles, we can extract some interesting insights from the confusion matrix and the classification metrics of our best model. Let's analyze first the classification scores, which we report below in Tab. 4.1.

From said classification scores, we can notice first of all a low precision for class 4 - Roads, indicating that many points were incorrectly classified as Road when they actually belonged to other classes. In other words, the model tends to over-predict class 4, resulting in a high number of false positives. Additionally, the low recall for class 3 - Railway suggests that many Railway points were misclassified into other categories,

Table 4.1: Classification metrics for the **best model**, with features: Intensity, RGB, NIR, $\lambda_1, \lambda_2, \lambda_3$.

Class	Accuracy	Precision	Recall	F1-Score
Grass	0.97	0.99	0.99	0.99
High Vegetation	0.99	0.98	0.99	0.99
Buildings	0.99	0.85	0.99	0.92
Railways	0.84	1.00	0.59	0.74
Roads	0.96	0.77	0.96	0.85
Cars	0.86	0.95	0.82	0.88
Average	**0.94**	**0.92**	**0.89**	**0.90**

leading to a high number of false negatives. This is likely because Railway points share similar features with other classes, particularly Roads, causing confusion. The low F1-Score for Railways, which balances precision and recall, highlights that the model's performance for class 3 remains problematic.

Let's now shift our attention to the confusion matrix for the best model, shown below in Fig. 4.1. This tells us that the model tends to misclassify other classes, like Railway, Road or Car, as Buildings. As a matter of facts, the Buildings' class precision is not exceptionally high (0.85%). Additionally, we can again notice that the model occasionally confuses the Railway and Road classes, likely due to their similar visual and contextual features. Finally, we see that Cars are often misclassified as parts of Buildings, leading to lower accuracy and recall for this class. Although the number of points falling into incorrect classes is relatively small for both Railway and Cars, these mistakes make the model performance for this classes drop significantly, given that they are the less represented ones.

Let us now begin our visual inspection of the predictions, with a quite complex tile that, given its structured nature, has often been used during the course of this work as a first visual evaluation of the models' predictive performance. Its RGB image representation and the predicted labels are shown in Fig. 4.2. We can already get some meaningful insights on the model's behavior from this very first example of predictions.

First, as anticipated, the model exhibits a tendency to misclassify points from the Building class (orange) as belonging to the Road (grey) or Railway (brown) classes, particularly when dealing with roof structures. Additionally, the frequent confusion between the Road and Railway classes indicates that merging these two categories into a single one could improve classification performance, especially if distinguishing between them is not a critical requirement. Another notable observation is the misclassification of certain roof points as Cars (in white), which may be due to overlapping geometric or spectral features. Finally, large shadows appear to affect the model's performance, often leading to the erroneous classification of shaded areas as Buildings.

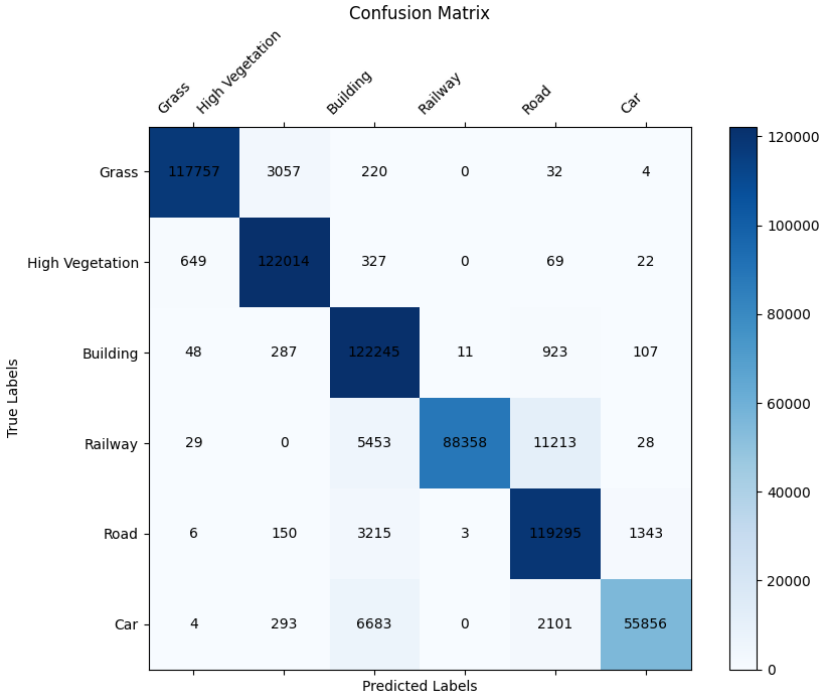


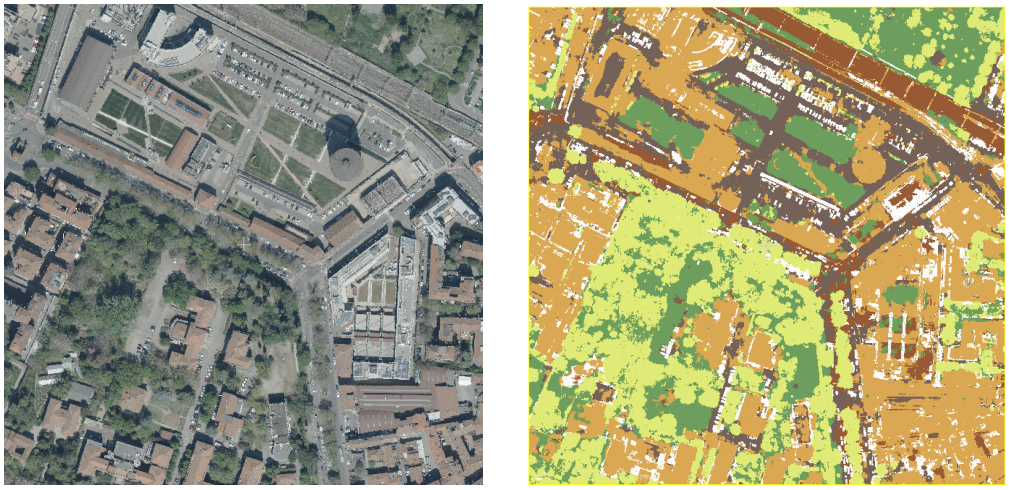
Figure 4.1: Confusion matrix for the best model.

The model demonstrates a notable strength in accurately classifying High Vegetation (depicted in yellow), even within dense urban areas. Grass (represented in green) is also effectively identified, particularly when it is located near High Vegetation. However, the model's performance diminishes when Grass is surrounded by buildings or roads. Remarkably, both the Grass and the High Vegetation classes are almost uneffected by the presence of shadows. Finally, we can notice that the railway points in the top portion of the tile are well classified.

Let us now investigate other interesting tiles, where vegetation is predominant over urban scenery. They are shown in Fig. 4.3.

Again, we can see that the model excels in classifying High Vegetation and Grass points, even though some errors are present. For instance, in 4.3d one of the two side-by-side football fields was classified as road, likely due to its more synthetic surface and darker color. Additionally, the model struggled to segment the thin trails surrounded by grass; this is most likely because these trails lack the distinct linearity, width, and consistent texture typically associated with roads. Furthermore, in Fig. 4.3b a portion of road in the top right corner was classified as Grass.

Once again, we observe that the model occasionally confuses Road and Railway classes. However, if these two classes were considered as a single category (brown + grey), the segmentation in Fig. 4.3b would appear accurate overall.



(a) RGB image **(b)** Predicted Labels

Figure 4.2: On the left, the image shows the point cloud in RGB values. On the right, the predicted labels. Buildings are orange, Grass is green, High Vegetation is yellow, Railway is brown, Roads are grey and Cars are white.

Notably, the model tends to misclassify points near bodies of water as buildings or roads. This behavior can likely be attributed to the scarcity of water-related scenarios in the training dataset, leaving the model ill-equipped to handle such situations during inference.

Finally, the model struggles to distinguish buildings in densely packed areas. Specifically, points from roofs are sometimes misclassified as cars, particularly in regions where multiple buildings are closely clustered together.

The tiles shown in Fig. 4.4 highlight the model's greatest strengths, while those in Fig. 4.5 underscore its key weaknesses.

In Fig. 4.4, which primarily features vegetation, the model achieves optimal label predictions. It effectively distinguishes Grass from High Vegetation, with only minor misclassification errors. These errors occur mainly in densely populated High Vegetation areas, where the model struggles to differentiate trees from the underlying ground. Additionally, the model demonstrates a reasonable ability to identify artificial structures surrounded by vegetation, though it still struggles to recognize smaller roads and trails.

Conversely, Fig. 4.5 illustrates the challenges faced by the model. In Fig. 4.5a, 4.5b, depicting a densely packed urban area with numerous closely spaced buildings and significant shadowing, roads are often misclassified, with most points incorrectly labeled as Buildings.

Fig. 4.5c, 4.5d, presents an almost inverse scenario: many rooftop points are misclassified as Road or Railway. Furthermore, certain building roofs and segments of roads are labeled as Grass, likely due to their planar shape, which may resemble the features

of grass-covered areas.

4.2 Comparative Analysis with Random Forest

In this section, we compare the performance of our proposed Multi-Class Convolutional Neural Network (MCNN) with the benchmark Random Forest (RF) model [38]. The classification metrics for both models, broken down by class, are presented in Table 4.2.

Table 4.2: Comparison of classification performance between the Random Forest and Neural Network models. Better scores are highlighted in bold.

Class	**Random Forest**				**Neural Network**			
	Accuracy	**Precision**	**Recall**	**F1**	**Accuracy**	**Precision**	**Recall**	**F1**
Grass	**0.99**	0.97	0.99	0.98	0.97	**0.99**	0.99	**0.99**
High Vegetation	0.99	**0.99**	0.99	0.99	0.99	0.98	0.99	0.99
Buildings	0.94	**0.97**	0.94	**0.96**	**0.99**	0.85	**0.99**	0.92
Railways	**0.96**	0.88	**0.96**	**0.92**	0.84	**1.00**	0.59	0.74
Roads	0.88	**0.97**	0.88	**0.92**	**0.96**	0.77	**0.96**	0.85
Cars	**0.95**	0.89	**0.95**	**0.92**	0.86	**0.95**	0.82	0.88
Total	**0.95**	**0.95**	**0.95**	**0.95**	0.94	0.92	0.89	0.90

The overall results indicate that the RF model outperforms the MCNN. Specifically, the RF model achieves higher recall and F1 scores across several classes, resulting in a better overall classification performance. Both models demonstrate exceptional performances on the Grass and High Vegetation classes. However, RF shows greater stability across categories, with more consistent scores. If we look, for instance, at the Building class, we can see that the RF maintains all scores above 94%, while the MCNN's precision drops to 85%. In contrast, the MCNN occasionally achieves outstanding results, such as 100% precision for the Railways class or 96% accuracy and recall for Roads, but this is accompanied by lower values in the other metrics, suggesting an imbalance in its predictions. In the following we try to investigate why these problems arose in our work.

A first culprit for the partial performance differences could be overfitting. This is an essential problem that Convolutional Neural Networks face [71]. As already described in §3.1, we chose to employ a patience of 2 epochs to mitigate this issue, meaning that after two epochs of non-increasing validation loss the training would automatically stop. The training and validation loss for our best trained model are shown in Fig. 4.6. We can notice how the model has indeed gone through a slight overfit towards the end of training, which was appropriately halted by the early stopping mechanism. This minimal overfitting may have played a role in the worse perfomance of our model compared to the RF, but its small magnitude suggests that deeper issues might underlie the MCNN's results.

One of these could be the MCNN model's excessive complexity. While deep neural networks are powerful tools, their capacity may sometimes exceed the requirements of the task. An excessively structured architecture can lead the model to over-adaptation to training data, rather than focusing on more generalizable features [72, 73]. In contrast, the Random Forest model, with its simpler decision boundaries and smaller feature space, may be better suited for the classification of Bologna's Dataset. A strenght of Random Forest models is specifically that they are not prone to overfitting, and can generalize well [44]. Furthermore, Random forest are known to possess an incredible robustness to noise. CNNs, on the other hand, often face issues with noisy data inputs [74].

The dataset itself may also have played a significant role. Subtle biases or a lack of diversity in the training data could limit the neural network's generalization capabilities [75]. Although the Random Forest model relies on simpler structures, it may handle these limitations more effectively [76]. Therefore, the MCNN's performance issues could partially stem from the training data's representativeness.

Finally, a critical limitation may lie in the restricted exploration of the feature and hyperparameter space. While efforts were made to align the model's training parameters with those proposed in [48, 59] where applicable, it is possible that further experimentation with hyperparameters - such as learning rates, learning rate decay values, or momentum - could have led to improved performance. However, due to time constraints, a more comprehensive exploration was not feasible. Building the model "from scratch" required significant effort, leaving limited time for extensive hyperparameter tuning.

4.3 Computational Efficiency

A limitation of this MCNN classifier is surely its expensive computational time. Segmentation, a critical preprocessing step to feature image generation in order to reduce computational overhead, was not performed due to the absence of reliable segmentation methods for the dataset, as previously described. While this omission may have slightly impacted classification performance [48], the primary issue lies in the resulting computational burden.

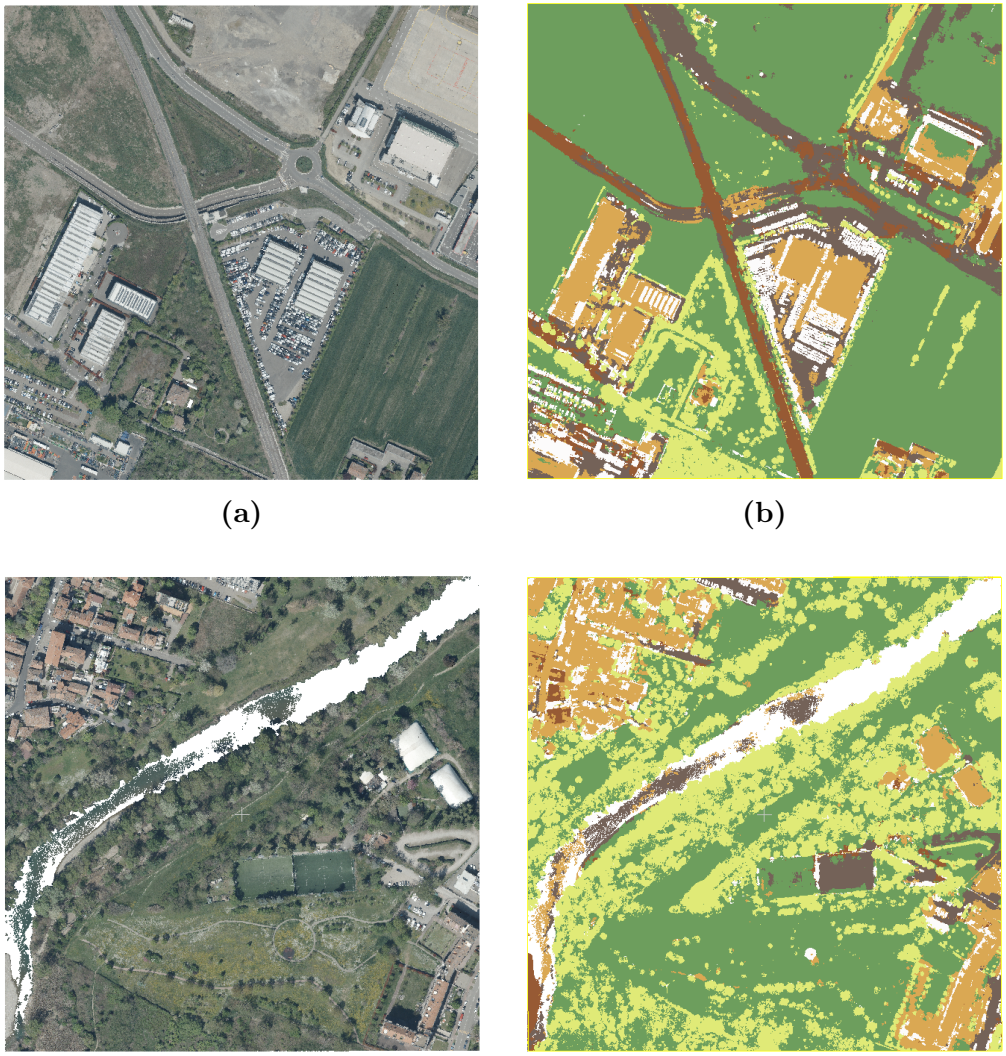
The MCNN classifier generates three high-resolution feature images for each point in the point cloud. This process involves searching for nearest neighbors and assigning their values to these images. However, the computational bottleneck is not the nearest neighbor search but rather the feature image generation itself. The massive scale of the dataset further exacerbates this issue, as pre-generating and saving the feature images would require an impractical amount of disk space due to the sheer number of points.

To mitigate this challenge, parallel processing was employed using multiple workers (`num_workers` > 0) in the PyTorch DataLoader [77]. Feature image generation was streamed dynamically, i.e., generated on-the-fly when points were retrieved from the dataset by the workers through the `get_item` method [78]. This approach significantly

reduced memory usage and allowed efficient data handling despite the dataset's size.

An additional attempt was made to transpose the nearest neighbor search and feature image generation to the GPU, utilizing PyTorch and a GPU-accelerated KDTree implementation [79]. However, this required dropping the streaming generation with multiple workers, as this process does not integrate swiftly with GPU usage [80]. Consequently, the GPU-based approach resulted in worse performance, further confirming that the nearest neighbor search is not the primary bottleneck in this workflow.

In summary, the high computational demand of the MCNN classifier stems from the 3-scales feature image generation, which remains a significant limitation despite attempts to optimize the process with both parallelization and GPU acceleration. This issue would be highly mitigated by adding a segmentation preprocessing step, as described in [48].



(a) **(b)**

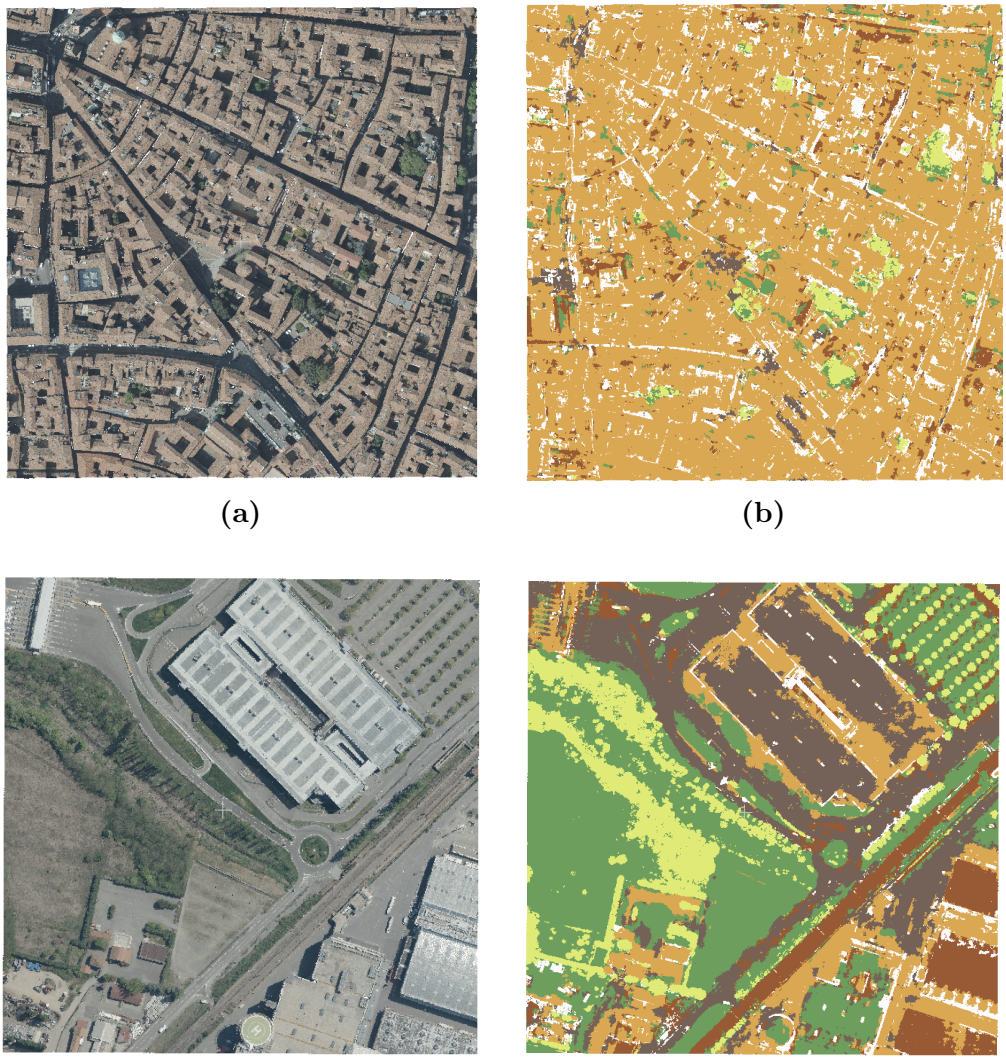
(c) **(d)**

Figure 4.3: On the left, the images show the point clouds in RGB values. On the right, the predicted labels. Buildings are orange, Grass is green, High Vegetation is yellow, Railway is brown, Roads are grey and Cars are white. Notice that the white color of the body of water present in Figs. 4.3c and 4.3d is not to be confused as points classified as Cars, but as missing points, since water was not detected in Bologna's LiDAR campaign.

(a) **(b)**

(c) **(d)**

Figure 4.4: On the left, the images show the point clouds in RGB values. On the right, the predicted labels. Buildings are orange, Grass is green, High Vegetation is yellow, Railway is brown, Roads are grey and Cars are white.



(a) **(b)**

(c) **(d)**

Figure 4.5: On the left, the images show the point clouds in RGB values. On the right, the predicted labels. Buildings are orange, Grass is green, High Vegetation is yellow, Railway is brown, Roads are grey and Cars are white.

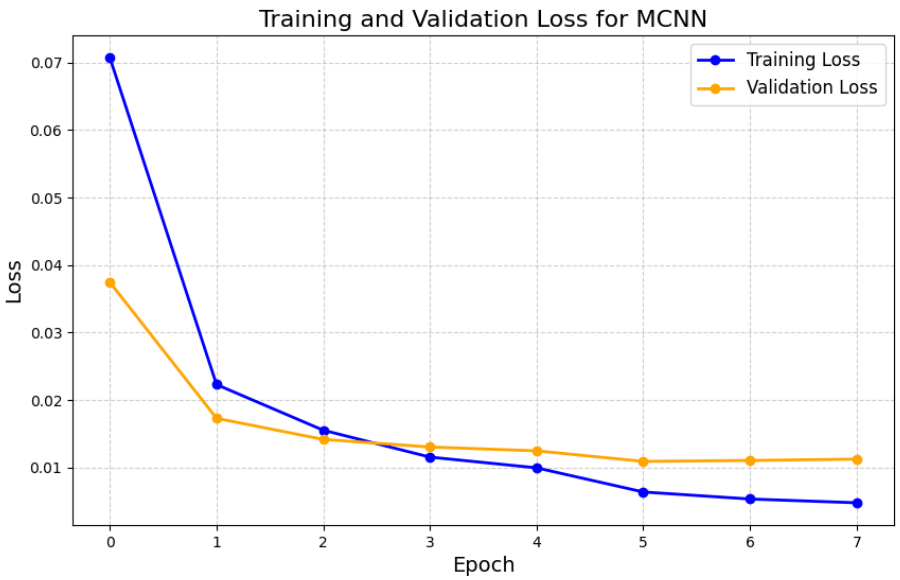


Figure 4.6: Training loss (in blue) and validation loss (in orange) of our best trained model.

Chapter 5

Conclusions

5.1 Summary

This thesis explored the application of deep learning techniques for the semantic classification of LiDAR data, contributing to the development of Bologna's Urban Digital Twin project. The primary objective was to harness the power of Convolutional Neural Networks (CNNs) for semantic segmentation of LiDAR point clouds, introducing the use of Artificial Intelligence for the classification task.

Digital Twin technology, which has recently emerged as a transformative paradigm, enables real-time interaction between digital models and physical urban environments, particularly within the scope of Urban Digital Twins [6]. As part of Bologna's Urban Digital Twin initiative, this work specifically addressed the classification of LiDAR data. The project commenced with an overview of LiDAR technology, followed by an in-depth analysis of Bologna's LiDAR dataset, comprising high-resolution airborne scans of the city enriched with RGB, NIR indices, and additional computed features [38].

A central aspect of this research was the creation of multi-channel feature images that encapsulate both geometric and spectral information to serve as inputs for the neural network. The proposed multi-scale CNN (MCNN) architecture, inspired by the foundational works of Yang et al. [48, 59], was adapted to process these feature images effectively, capturing spatial patterns at three distinct scales. The implementation and accompanying resources are publicly available on GitHub at this link[1].

Extensive experimentation led to the determination of optimal window sizes and feature combinations that enhanced the classification performance. The model achieved satisfactory results across six predefined urban classes, as presented in Table 4.1. Specifically, it attained a total accuracy of 94%, with the lowest class accuracy recorded at 84%. However, some limitations emerged, including misclassification of road and railway points, computational inefficiencies, and slight overfitting tendencies.

[1] https://github.com/MatteoFalcioni/Multiscale_Convolutional_Neural_Network

A comparative analysis was performed to evaluate the proposed model's performance against a benchmark Random Forest (RF) model [38]. The findings indicated that while the RF model outperformed the MCNN in several metrics, the results underscored the potential of CNN-based approaches, leaving room for further refinement of the architecture and methodology.

The results demonstrate the potential of CNN-based approaches for urban LiDAR data classification, while highlighting areas for improvement, such as segmentation-based preprocessing, further hyperparameter optimization, and other deep learning architecture explorations.

Overall, the results demonstrate the feasibility of leveraging CNNs for urban LiDAR data classification while identifying areas for future work, including segmentation-based preprocessing, hyperparameter optimization, and the exploration of alternative deep learning architectures. This research represents a significant step forward in methodologies for LiDAR data processing and serves as the first application of AI within Bologna's Urban Digital Twin project. Furthermore, it lays the groundwork for extending AI applications to other facets of Bologna's Urban Digital Twin initiative, paving the way for broader integration of artificial intelligence technologies in urban planning and management.

5.2 Future Directions

This thesis has established a strong starting point for the application of deep learning techniques to LiDAR data classification within Bologna's Urban Digital Twin project. However, there remain numerous opportunities to expand and refine the methodology.

One promising direction is the development of hybrid models that combine the strengths of Convolutional Neural Networks (CNNs) and Random Forests (RFs). These models, which have been explored in several different fields [81–84], could very well be applied to LiDAR data in order to capitalize on the feature extraction capabilities of CNNs while benefiting from the interpretability and stability of RFs. Such approaches may yield significant improvements, particularly in the case of complex datasets like Bologna's LiDAR data.

Improving computational efficiency and generalization is another key area for exploration. Segmentation-based preprocessing could be implemented to reduce computational overhead and improve classification performances, as demonstrated by Yang et al. [48]. Similarly, incorporating advanced normalization techniques, such as dropout [85], could help mitigate overfitting and enhance model robustness.

Data augmentation represents another promising avenue for improvement. Given the complexity and diversity of Bologna's LiDAR dataset, applying augmentation techniques such as rotation, scaling, flipping, and synthetic noise could help diversify the training data. This would allow the model to better adapt to real-world variations, ultimately

improving its accuracy and reliability on unseen data [86].

Beyond classification, an intriguing avenue for future work lies in the application of generative AI to LiDAR point clouds. Generative models, such as diffusion-based approaches, GAN and VAEs, could be explored to create synthetic LiDAR data that replicates the structure and variability of real-world datasets. This capability could enable simulations, where hypothetical scenarios are tested using artificially generated point clouds [87], or the enhancement of existing datasets [88] to improve their utility for urban planning or other analytical purposes. These possibilities extend the scope of AI applications within Bologna's Urban Digital Twin project, offering innovative tools for research and development.

By pursuing these directions, this work can evolve to address current limitations and unlock new opportunities for AI-driven advancements in urban LiDAR data processing, contributing to the broader goals of Bologna's Urban Digital Twin initiative.

Appendix A

Computed Features

In §1.3, we described Bologna's LiDAR Dataset, and enumerated the features included in the dataset. In the following, we list the computed features [38], which were not present in the original raw LiDAR data.

- **NDVI (Normalized Difference Vegetation Index)**:

$$\text{NDVI} = \frac{\text{NIR} - \text{Red}}{\text{NIR} + \text{Red}}$$

  Highlights vegetation by leveraging the difference between near-infrared (NIR) and red reflectance.

- **NDWI (Normalized Difference Water Index)**:

$$\text{NDWI} = \frac{\text{Green} - \text{NIR}}{\text{Green} + \text{NIR}}$$

  Useful for detecting vegetation and water-related features.

- **SSI (Spectral Shape Index)**:

$$\text{SSI} = |R + B - 2G|$$

  Combines red, green, and blue values to distinguish shadows and vegetation.

- N_h: Elevation above the ground, calculated as:

$$N_h = z - \text{DTM}$$

  Useful for separating ground from elevated points like trees and buildings.

- ΔZ: The gap between the analyzed point's height and the lowest point in its neighborhood:

$$\Delta Z = z_{\text{analyzed}} - z_{\text{min}}$$

- **MAD (Median Absolute Deviation)**: Median of the absolute deviations in height within the local neighborhood:

$$\text{MAD} = \text{median}_i \left[\left| N_{h_i} - \text{median}_j(N_{h_j}) \right| \right]$$

- $\Delta Z_{\mathbf{fl}}$: Height difference between the first and last return of the same beam.

- θ **and** σ_θ^2: Describe the inclination of the local plane and its variance.

To compute the local plane for a point's neighborhood, the points within a defined radius around the central point were selected. A plane was then fit to these points by minimizing the sum of the distances between the points and the plane. The $L_{1.2}$ norm was used for this minimization, as it provided a balance between the L_1 norm, which ignores outliers, and the L_2 norm, which can overly penalize them. Once the plane was defined, the vector perpendicular to it (the normal vector) was calculated, and its angle θ with the vertical axis was used as a feature. Then, its variance σ_θ^2 was computed [38].

Since local geometrical features were calculated from the covariance matrix in a neighborhood of the analised point, let us first describe the computing process, and then list them. The local point cloud can be viewed as a distribution in three-dimensional space (x, y, z), allowing the computation of a 3×3 covariance matrix C [38]. Each element C^{ij} of the matrix is calculated as:

$$C^{ij} = \sum_k \left(p_k^i - \bar{p}^i \right) \left(p_k^j - \bar{p}^j \right), \tag{A.1}$$

where k indexes the points within the neighborhood, p_k^i and p_k^j are the coordinates of the points, and $\bar{p}$ is the mean position of the neighborhood points. Since the covariance matrix is symmetric, it can be diagonalized with real eigenvalues, which describe the variance of the point cloud along its principal directions - the eigenvectors directions.

The covariance matrix is positive semi-definite because the variance along any linear combination of the coordinates is always non-negative:

$$\mathbf{u}^T C \mathbf{u} = V \geq 0, \tag{A.2}$$

where V is the variance and $\mathbf{u}$ is a non-zero vector. Physically, this means the eigenvalues of the matrix correspond to the variance of the points' scattering along the eigenvectors, representing the principal axes of the distribution. These eigenvalues ($\lambda_1, \lambda_2, \lambda_3$, with $\lambda_1 \geq \lambda_2 \geq \lambda_3$) help describe the geometry of the local point cloud as an ellipsoid centered on the analyzed point.

- λ_1: The largest eigenvalue corresponds to the direction of maximum variance, where the points are most scattered.
- λ_2: The second eigenvalue reflects variance along the intermediate axis.
- λ_3: The smallest eigenvalue represents the least scattering and the shortest axis.

The eigenvalues can indicate the shape of the point distribution:

a) If λ_1, λ_2, and λ_3 are nearly equal, the points form a spherical distribution.

b) If λ_1 is much larger than λ_2 and λ_3, the points align along a line.

c) If λ_1 and λ_2 are large compared to λ_3, the points form a planar structure.

We are now ready to list the covariance matrix derived features:

- **Planarity**:
$$\text{Planarity} = \frac{\lambda_2 - \lambda_3}{\lambda_1}$$
Describes how planar the local point cloud is, based on eigenvalues $\lambda_1, \lambda_2, \lambda_3$ of the covariance matrix.
- **Sphericity**:
$$\text{Sphericity} = \frac{\lambda_3}{\lambda_1}$$
Measures isotropy of point distribution.
- **Linearity**:
$$\text{Linearity} = \frac{\lambda_1 - \lambda_2}{\lambda_1}$$
Identifies points distributed along a line.
- **Entropy**: Describes the randomness in the distribution of neighboring points. It's defined as
$$\text{entropy} = plan \log\left(\frac{1}{plan}\right) + spher \log\left(\frac{1}{spher}\right) + lin \log\left(\frac{1}{lin}\right)$$

Appendix B

Convolutional Neural Networks

Convolutional Neural Networks are a widely popular category of Neural Network architecture, mainly used for applications involving structured data, such as images and videos. Their main strengths reside in their ability to automatically learn hierarchical features from raw data [54] and their relatively low number of parameters, due to the hierarchical structure of their layers [89].

The structure of a Convolutional Neural Network (CNN) resembles that of a funnel, where the architecture is designed to progressively distill the input data into its most relevant features. The initial layers operate on high-dimensional input data, such as images, extracting low-level features like edges and textures. As the data flows through deeper layers, these features are hierarchically combined to capture more complex patterns, ultimately reducing the spatial dimensions. This process culminates in a compact, high-dimensional vector representation that encodes the most significant information about the input.
This condensed vector is then passed to fully connected layers, which perform the final classification or regression tasks. Such a design enables CNNs to efficiently "squeeze out" information from the input data, ensuring that the network focuses on the most discriminative features for the given task. See Fig. B.1 for a visual depiction of this process.

More specifically, basic CNNs are usually comprised of four types of layers: convolutional layers, pooling layers, activation layers and fully-connected layers [89]. When these layers are stacked, a CNN architecture has been formed. Let's quickly go through their description [55].

1) **Convolutional Layer:** this is the most significant component of the architecture. It consists of a collection of convolutional filters (so-called kernels). The input image, a multi-dimensional tensor (e.g., height $\times$ width $\times$ depth for images), is convolved with these filters to generate the output feature map. The kernels consist

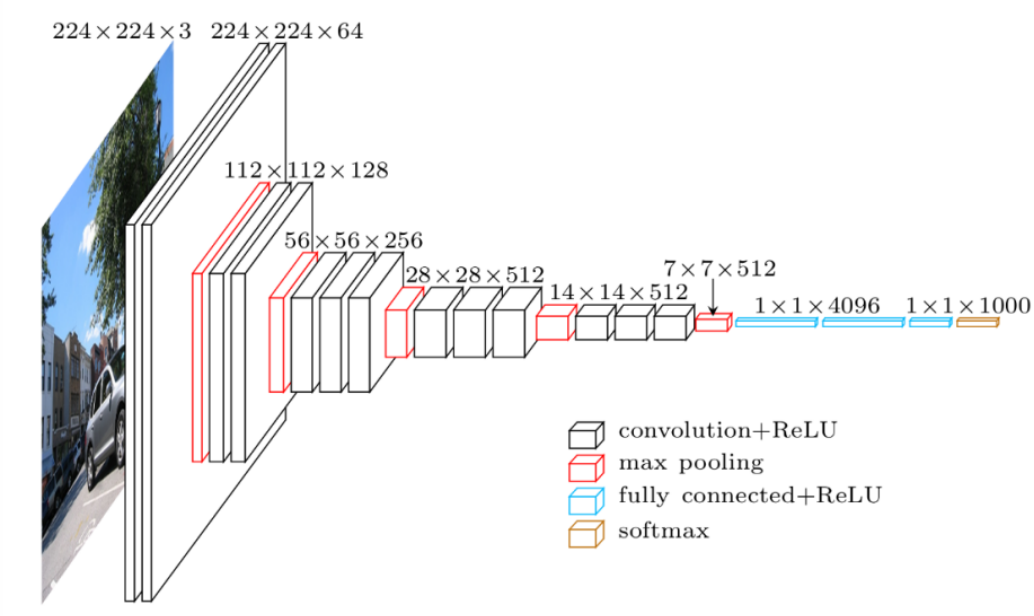


Figure B.1: Visual depiction of the functioning of a Convolutiona Neuran Network, from [90].

of grids of discrete number, whose values are randomly assigned[1] at the beginning of the training process. The network will then learn the weights and adjust their values.
The convolution operation involves sliding the kernel across the input image both horizontally and vertically. At each position, the dot product between the kernel and the corresponding region of the input image is computed by multiplying their values element-wise and summing the results. This produces a single scalar value for that position. The process is repeated over the entire image, resulting in a feature map that encodes the detected features at each location. This process is depicted in Fig. B.2

In the schematized example, the kernel is moved to the right with a step size of 1, which is referred to as the stride. This parameter can be set to values greater than 1 to reduce the size of the output feature map. Conversely, smaller strides lead to higher-resolution feature maps. Additionally, in this example, no padding is applied. Padding refers to the addition of extra rows and columns around the border of the input image, typically filled with zeros. It is used to control the spatial dimensions of the output feature map; without padding, the feature map size is reduced after each convolution, as the kernel cannot fully cover the edges of the input image. By adding padding, we can instead preserve the original spatial dimensions or control the extent of this reduction.

[1] In general, but there are several initialization methods for the weights [55].

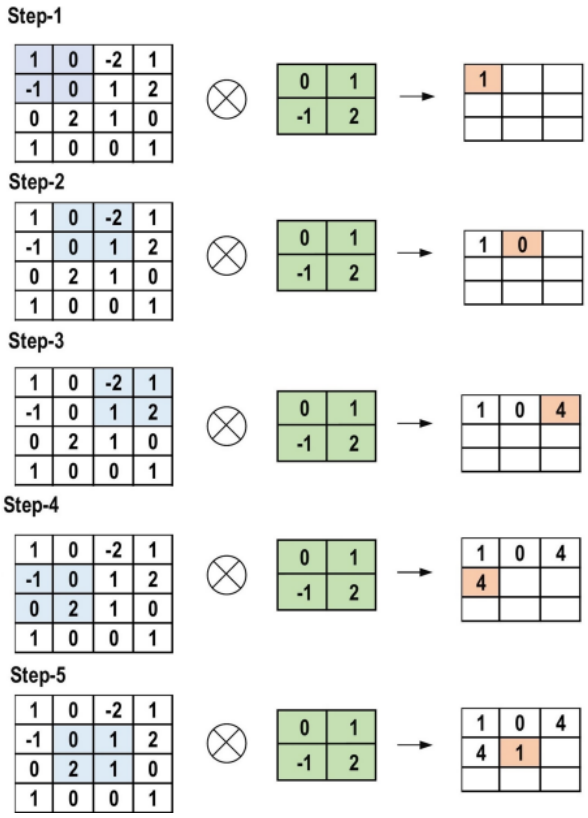


Figure B.2: Example of the calculations executed at each step of convolutional layer [55].

Convolutional layers have two primary benefits:

(i) **Sparse Connectivity**: Each neuron of a layer in FC neural networks links with all neurons in the following layer. By contrast, in CNNs, only a few weights are available between two adjacent layers. Thus, the number of required weights or connections is small, while the memory required to store these weights is also small.

(ii) **Weight Sharing**: In CNNs, the same set of weights (referred to as a kernel or filter) is applied across all regions of the input matrix within a layer. This means that the weights are "shared" across different parts of the input rather than being unique for each connection between neurons. By learning a single group of weights that is reused across the entire input, the network drastically reduces the number of parameters it needs to train. This not only decreases the training time but also helps the model converge faster and reduces the risk of overfitting.

2) Pooling Layer: The main task of the pooling layer is to reduce the spatial di-

mensions of feature maps while retaining the most dominant information. This dimensionality reduction helps to decrease computational complexity, prevent overfitting, and enhance feature invariance to small translations in the input. Several types of pooling methods are available, with the most common being max pooling, min pooling, and global average pooling (GAP). Max pooling captures the most significant feature in each region, while min pooling captures the smallest value, and GAP calculates the average value of the entire feature map, often used in the final stages of the network. Fig. B.3 illustrates these three pooling operations.

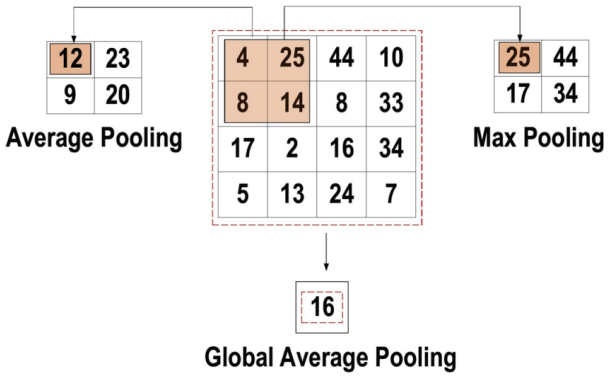


Figure B.3: The three most common types of pooling operations [55].

3) **Activation Layer:** As usual, activation functions are at the core structure of the network. In CNN architecture, non-linear activation layers are employed after all layers with weights (so-called learnable layers, such as FC layers and convolutional layers). Several commonly used activation functions include the sigmoid $f(x) = 1/1 + e^{-x}$, the hyperbolic tangent $f(x) = \tanh(x)$, the ReLU $f(x) = \max(0, x)$, and so on. To enable error back-propagation during training, the chosen activation function must be differentiable.

4) **Fully Connected Layer:** Typically located at the end of a CNN architecture, this layer connects each neuron to every neuron in the preceding layer, following the structure of a traditional multi-layer perceptron. Fully connected layers aggregate features extracted by earlier layers and produce the final output, often in the form of classification scores or regression values.

5) **Batch Normalization:** Additionally, Batch Normalization layers could be present in more refined architectures. Batch normalization is a crucial technique in modern CNNs, designed to improve training efficiency and stability: it normalizes the input of each layer to have a mean of 0 and a variance of 1 across a mini-batch, reducing internal covariate shift - the change in the distribution of layer inputs during

training. By standardizing inputs, batch normalization enables faster convergence, allows for higher learning rates, and mitigates issues like vanishing or exploding gradients. Additionally, it has a slight regularization effect, helping to reduce overfitting in some cases. This layer is typically placed after the linear transformations in convolutional or fully connected layers and before the activation functions.

Structuring these layers together constructs a basic Convolutional Neural Network.

The architecture can then be refined by adding additional layers with different purposes; for example, a Dropout Layer, whose goal is to reduce overfitting by randomly "dropping out" (i.e., setting to zero) a fraction of neurons during training.

The choice of the loss function is another crucial step in the definition of the network. While a detailed discussion of this topic is beyond the scope of this work, we will briefly outline commonly used loss functions:

(i) **Cross-Entropy or Softmax Loss Function:** Often used in classification tasks, this loss function leverages softmax activations in the output layer to produce a probability distribution over the output classes. The softmax function is mathematically defined as:

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a} \tag{B.1}$$

where a_i is the non-normalized output (logits) from the previous layer, and N is the total number of output neurons. The resulting p_i represents the probability of the input belonging to class i. The cross-entropy loss measures the difference between the predicted probability distribution p and the true class labels y. It is given by:

$$H(p, y) = -\sum_{i=1}^{N} y_i \log(p_i), \tag{B.2}$$

where y_i is the one-hot encoded ground truth label for class i, and p_i is the predicted probability for the same class. Cross-entropy loss penalizes incorrect predictions, driving the network to adjust its weights to maximize the probability assigned to the correct class.

(ii) **Mean Squared Error (MSE):** MSE is a commonly used loss function for regression tasks, as it measures the average squared difference between the predicted and true values. The MSE penalizes larger errors more significantly than smaller ones, making it sensitive to outliers. The mathematical formula for MSE is:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}_i\right)^2, \tag{B.3}$$

where N is the number of data points, y_i is the true value, and $\hat{y}_i$ is the predicted value for the i-th sample. MSE encourages the network to minimize large deviations, resulting in predictions that closely approximate the ground truth.

(iii) **Hinge Loss:** Hinge loss is widely used in classification tasks involving Support Vector Machines (SVMs) and is particularly effective when aiming to maximize the margin between the decision boundary and data points. It is designed for binary classification problems, where the labels are typically encoded as -1 or $+1$. The hinge loss function is given by:

$$\text{Hinge Loss} = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i \cdot \hat{y}_i), \tag{B.4}$$

where y_i is the true label (-1 or $+1$) and $\hat{y}_i$ is the predicted score. The hinge loss penalizes predictions that fall on the wrong side of the decision boundary or are within the margin, driving the network to improve classification accuracy by pushing the predictions away from the boundary and maximizing the margin.

As described earlier in §3.1, we chose to employ the Cross Entropy Loss in our work, following [48].

Another pivotal point is the optimizer choice. Optimizers play a crucial role in training neural networks by updating the model's weights to minimize the loss function. They adjust the parameters based on the gradients computed during back-propagation, guiding the network toward optimal performance. Again, a thorough description of all optimizers is fairly out of the scope of our work, so let us quickly go over some common possible choices. Stochastic Gradient Descent (SGD) [91] is a widely used optimizer, with its possible variants like Momentum and Nesterov Accelerated Gradient [92], which improve convergence speed by incorporating velocity terms. Adaptive optimizers, such as Adam [93], dynamically adjust learning rates for each parameter, making them effective for complex and non-stationary tasks.
In our model, we chose SGD optimization, again following [48, 59].

Appendix C

Feature Exploration Results

In §3.3, we summarized the outcomes of the exploration of the selected features hyper-parameter of our model. We will thoroughly present in this appendix the results of the different models. For simplicity, we will use numbers instead of full text labels in the tables, so let us recall that the classes were the following: 0 - Grass; 1 - High Vegetation; 2 - Building; 3 - Railway; 4 - Road; 5 - Car.

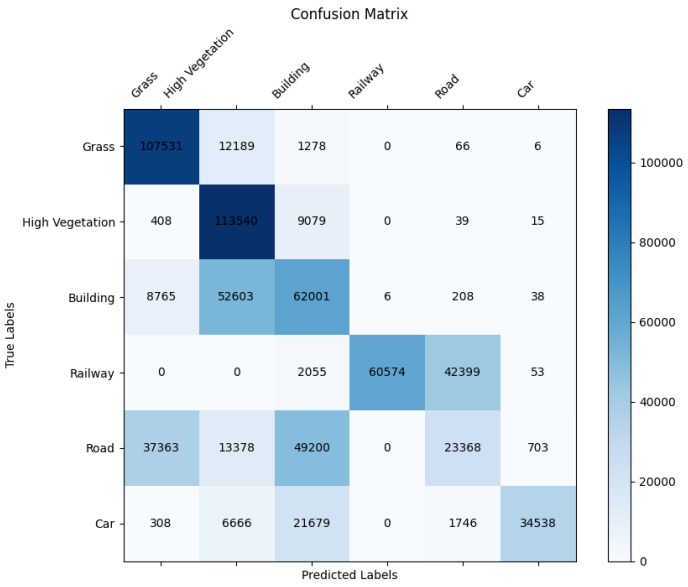


Figure C.1: Confusion Matrix for model with: Intensity

Class	Accuracy	Precision	Recall	F1-Score
0	0.89	0.70	0.89	0.78
1	0.92	0.57	0.92	0.71
2	0.50	0.43	0.50	0.46
3	0.57	1.00	0.58	0.73
4	0.19	0.34	0.19	0.24
5	0.53	0.98	0.53	0.69
Average	**0.59**	**0.67**	**0.77**	**0.68**

Table C.1: Classification metrics for model with: Intensity

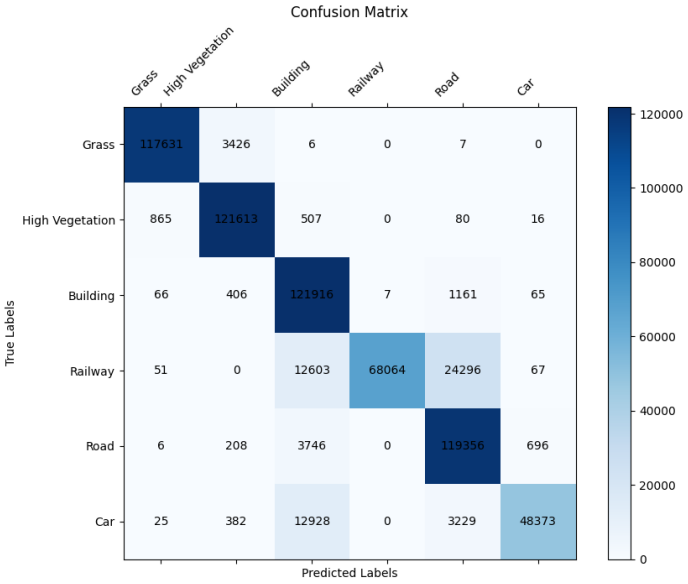


Figure C.2: Confusion Matrix for model with: Intensity, RGB.

Class	Accuracy	Precision	Recall	F1-Score
0	0.97	0.99	0.97	0.98
1	0.99	0.96	0.99	0.98
2	0.97	0.80	0.99	0.89
3	0.66	1.00	0.65	0.79
4	0.96	0.81	0.96	0.88
5	0.75	0.98	0.74	0.85
Average	**0.88**	**0.92**	**0.88**	**0.90**

Table C.2: Classification metrics for model with: Intensity, RGB.

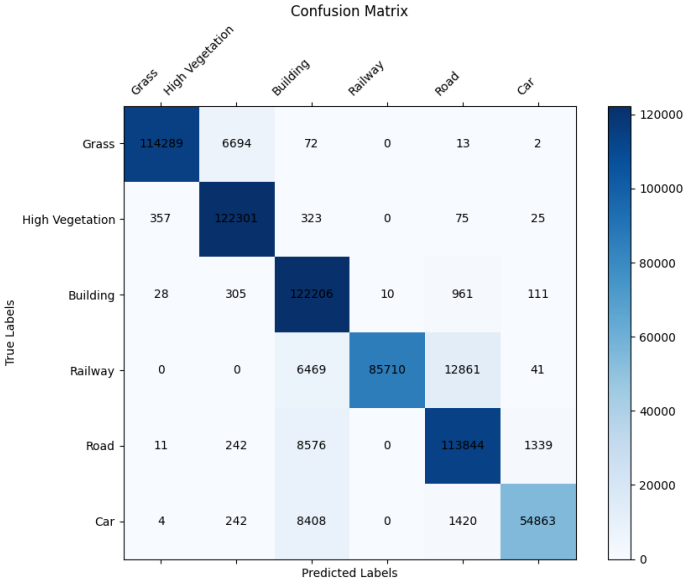


Figure C.3: Confusion Matrix for model with: Intensity, RGB, NIR.

Class	Accuracy	Precision	Recall	F1-Score
0	0.94	1.00	0.94	0.97
1	0.99	0.94	0.99	0.97
2	0.93	0.84	0.99	0.91
3	0.82	1.00	0.82	0.90
4	0.91	0.88	0.92	0.90
5	0.85	0.97	0.84	0.90
Average	**0.91**	**0.94**	**0.91**	**0.92**

Table C.3: Classification metrics for model with: Intensity, RGB, NIR.

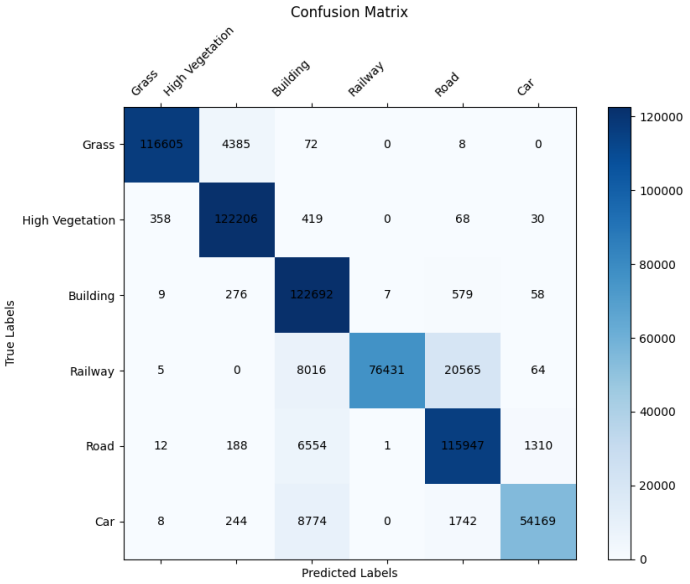


Figure C.4: Confusion Matrix for model with: Intensity, RGB, NIR, NDVI, SSI.

Class	Accuracy	Precision	Recall	F1-Score
0	0.96	1.00	0.96	0.98
1	0.99	0.96	0.99	0.98
2	0.93	0.84	0.99	0.91
3	0.73	1.00	0.73	0.84
4	0.92	0.83	0.93	0.88
5	0.84	0.97	0.83	0.90
Average	**0.90**	**0.93**	**0.90**	**0.91**

Table C.4: Classification metrics for model with: Intensity, RGB, NIR, NDVI, SSI.

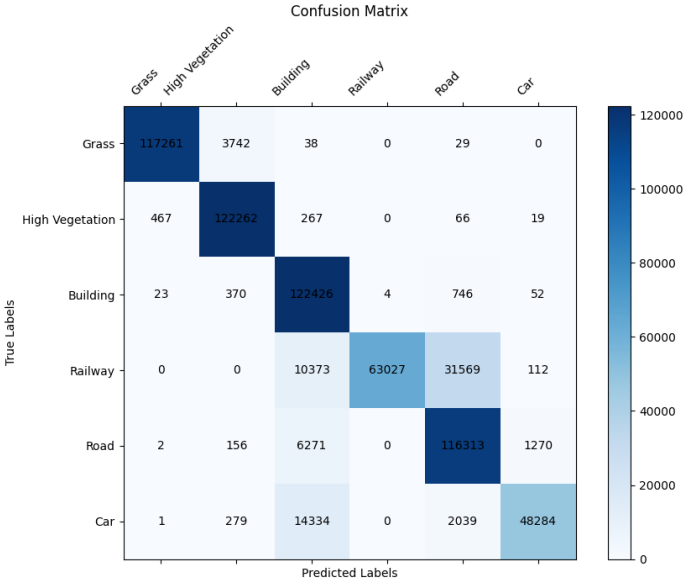


Figure C.5: Confusion Matrix for model with: Intensity, RGB, NIR, Δz.

Class	Accuracy	Precision	Recall	F1-Score
0	0.97	1.00	0.97	0.98
1	0.99	0.96	0.99	0.98
2	0.93	0.80	0.99	0.88
3	0.60	1.00	0.60	0.75
4	0.92	0.77	0.94	0.85
5	0.73	0.97	0.74	0.84
Average	**0.86**	**0.92**	**0.87**	**0.88**

Table C.5: Classification metrics for model with: Intensity, RGB, NIR, Δz.

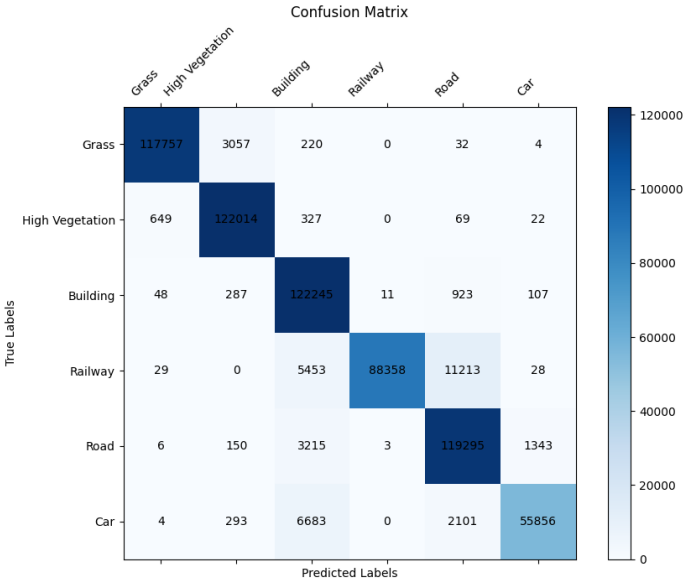


Figure C.6: Confusion Matrix for the **best model**, with features: Intensity, RGB, NIR, $\lambda_1, \lambda_2, \lambda_3$.

Class	Accuracy	Precision	Recall	F1-Score
0	0.97	0.99	0.99	0.99
1	0.99	0.98	0.99	0.99
2	0.99	0.85	0.99	0.92
3	0.84	1.00	0.59	0.74
4	0.96	0.77	0.96	0.85
5	0.86	0.95	0.82	0.88
Average	**0.94**	**0.92**	**0.89**	**0.90**

Table C.6: Classification metrics for the **best model**, with features: Intensity, RGB, NIR, $\lambda_1, \lambda_2, \lambda_3$.

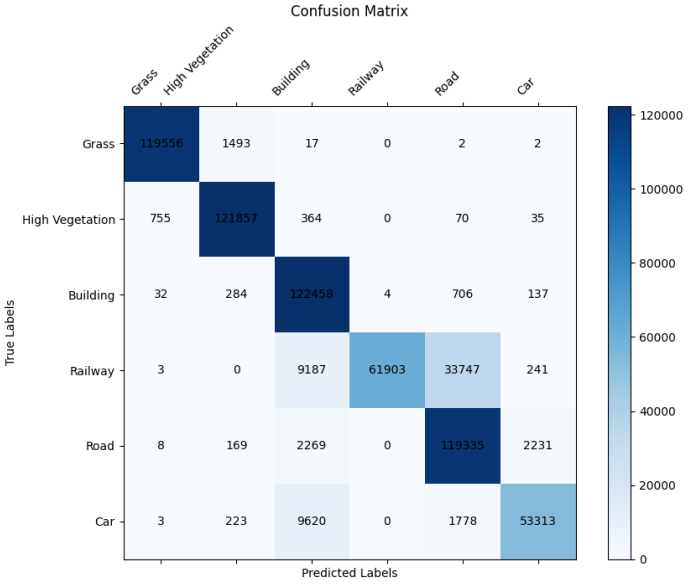


Figure C.7: Confusion Matrix for model with: Intensity, RGB, NIR, $\lambda_1, \lambda_2, \lambda_3$, θ and σ_θ^2.

Class	Accuracy	Precision	Recall	F1-Score
0	0.99	0.99	0.99	0.99
1	0.99	0.98	0.99	0.99
2	0.99	0.85	0.99	0.92
3	0.59	1.00	0.59	0.74
4	0.96	0.77	0.96	0.85
5	0.82	0.95	0.82	0.88
Average	**0.89**	**0.92**	**0.89**	**0.90**

Table C.7: Classification metrics for model with: Intensity, RGB, NIR, $\lambda_1, \lambda_2, \lambda_3$, θ and σ_θ^2.

Acknowledgments

I would like to thank my supervisor for the opportunity to work on this incredibly interesting project, and all the knowledge I gained from this experience. I hope this work was as useful and instructive to him as it was to me.
To all the members of CINECA and of the University of Bologna I collaborated with, thank you for allowing me to work with you, for your help and for your valuable technical and theoretical expertise.

I would like to thank my parents for two main reasons: first, for supporting me and allowing me to pursue my academic career; and second, for having given me the opportunity to live such important experiences and meet such wonderful people over the past five years. I deeply admire you for understanding that, even though you love me to death, you need to let me follow my own path to grow.
I would like to thank my sister, who inspired me to try and be a good example for her. I hope I was sometimes; you surely were for me.

I would like to thank my friends, for demonstrating me that there are so many amazing and interesting people in the world - like you guys - and thus giving me the desire to continue meeting new people and living new experiences. You all hold a special place in my heart, no matter how far apart we are from each other.

Bibliography

[1] Fei Tao et al. "Digital twin modeling". In: *Journal of Manufacturing Systems* 64 (2022), pp. 372–389. ISSN: 0278-6125. DOI: https://doi.org/10.1016/j.jmsy.2022.06.015. URL: https://www.sciencedirect.com/science/article/pii/S0278612522001108.

[2] Rasha F. El-Agamy et al. "Comprehensive analysis of digital twins in smart cities: a 4200-paper bibliometric study". In: *Artificial Intelligence Review* 57.6 (2024), p. 154. ISSN: 1573-7462. DOI: 10.1007/s10462-024-10781-8. URL: https://doi.org/10.1007/s10462-024-10781-8.

[3] Mohd Javaid, Abid Haleem, and Rajiv Suman. "Digital Twin applications toward Industry 4.0: A Review". In: *Cognitive Robotics* 3 (2023), pp. 71–92. ISSN: 2667-2413. DOI: https://doi.org/10.1016/j.cogr.2023.04.003. URL: https://www.sciencedirect.com/science/article/pii/S2667241323000137.

[4] M. Mazhar Rathore et al. "The Role of AI, Machine Learning, and Big Data in Digital Twinning: A Systematic Literature Review, Challenges, and Opportunities". In: *IEEE Access* 9 (2021), pp. 32030–32052. DOI: 10.1109/ACCESS.2021.3060863.

[5] Ayoub Arroub et al. "A literature review on Smart Cities: Paradigms, opportunities and open problems". In: *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. 2016, pp. 180–186. DOI: 10.1109/WINCOM.2016.7777211.

[6] Jaume Ferré-Bigorra, Miquel Casals, and Marta Gangolells. "The adoption of urban digital twins". In: *Cities* 131 (2022), p. 103905. ISSN: 0264-2751. DOI: https://doi.org/10.1016/j.cities.2022.103905. URL: https://www.sciencedirect.com/science/article/pii/S0264275122003444.

[7] Comune di Bologna. *Bologna's Digital Twin Project*. Accessed: 2025-01-15. 2024. URL: https://www.comune.bologna.it/notizie/gemello-digitale.

[8] S. Dittmann, E. Thiessen, and E. Hartung. "Applicability of different non-invasive methods for tree mass estimation: A review". In: *Forest Ecology and Management* 398 (2017), pp. 208–215. ISSN: 0378-1127. DOI: https://doi.org/10.1016/

j.foreco.2017.05.013. URL: https://www.sciencedirect.com/science/article/pii/S0378112717304838.

[9] Aloysius Wehr and Uwe Lohr. "Airborne laser scanning—an introduction and overview". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 54.2 (1999), pp. 68–82. ISSN: 0924-2716. DOI: https://doi.org/10.1016/S0924-2716(99)00011-8. URL: https://www.sciencedirect.com/science/article/pii/S0924271699000118.

[10] Cheng Wang et al. *Introduction to LiDAR remote sensing*. CRC Press, 2024.

[11] Clément Mallet and Frédéric Bretar. "Full-waveform topographic lidar: State-of-the-art". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 64 (Jan. 2009), pp. 1–16. DOI: 10.1016/j.isprsjprs.2008.09.007.

[12] Xiaolu Li et al. "Airborne LiDAR: state-of-the-art of system design, technology and application". In: *Measurement Science and Technology* 32.3 (Dec. 2020), p. 032002. DOI: 10.1088/1361-6501/abc867. URL: https://dx.doi.org/10.1088/1361-6501/abc867.

[13] Waleed Abdalati et al. "The ICESat-2 Laser Altimetry Mission". In: *Proceedings of the IEEE* 98.5 (2010), pp. 735–751. DOI: 10.1109/JPROC.2009.2034765.

[14] Sorin C. Popescu et al. "Satellite lidar vs. small footprint airborne lidar: Comparing the accuracy of aboveground biomass estimates and forest structure metrics at footprint level". In: *Remote Sensing of Environment* 115.11 (2011). DESDynI VEG-3D Special Issue, pp. 2786–2797. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2011.01.026. URL: https://www.sciencedirect.com/science/article/pii/S0034425711001325.

[15] Michal Gallay. "Direct acquisition of data: airborne laser scanning". In: *Geomorphological techniques* 2.1.4 (2013), pp. 1–17.

[16] M. Bisson et al. "The use of historical cartography and ALS technology to map the geomorphological changes of volcanic areas: A case study from Gran Cono of Somma-Vesuvius volcano". In: *Geomorphology* 380 (2021), p. 107624. ISSN: 0169-555X. DOI: 10.1016/j.geomorph.2021.107624. URL: https://www.sciencedirect.com/science/article/pii/S0169555X21000325.

[17] Chris Hopkinson et al. "Moving Toward Consistent ALS Monitoring of Forest Attributes across Canada". In: *Photogrammetric Engineering and Remote Sensing* 79.2 (2013), pp. 159–173. ISSN: 0099-1112. DOI: 10.14358/PERS.79.2.159. URL: https://www.ingentaconnect.com/content/asprs/pers/2013/00000079/00000002/art00003.

[18] Giacomo Vinci et al. “LiDAR Applications in Archaeology: A Systematic Review”. In: *Archaeological Prospection* n/a.n/a (2024). DOI: https://doi.org/10.1002/arp.1931. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/arp.1931. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/arp.1931.

[19] C. Yao et al. “Research Note: Multi-Algorithm-Based urban tree information extraction and Its applications in urban planning”. In: *Landscape and Urban Planning* 253 (2024), p. 105226. ISSN: 0169-2046. DOI: 10.1016/j.landurbplan.2024.105226. URL: https://www.sciencedirect.com/science/article/pii/S0169204624002251.

[20] You Li and Javier Ibanez-Guzman. “Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems”. In: *IEEE Signal Processing Magazine* 37.4 (2020), pp. 50–61. DOI: 10.1109/MSP.2020.2973615.

[21] Jan Trochta et al. “3D Forest: An application for descriptions of three-dimensional forest structures using terrestrial LiDAR”. In: *PLOS ONE* 12 (May 2017), pp. 1–17. DOI: 10.1371/journal.pone.0176871. URL: https://doi.org/10.1371/journal.pone.0176871.

[22] Bharat Lohani and Suddhasheel Ghosh. “Airborne LiDAR Technology: A Review of Data Collection and Processing Systems”. In: *Proceedings of the National Academy of Sciences, India Section A: Physical Sciences* 87.4 (2017), pp. 567–579. ISSN: 2250-1762. DOI: 10.1007/s40010-017-0435-9. URL: https://doi.org/10.1007/s40010-017-0435-9.

[23] Dorota A. Grejner-Brzezinska et al. “Multisensor Navigation Systems: A Remedy for GNSS Vulnerabilities?” In: *Proceedings of the IEEE* 104.6 (2016), pp. 1339–1353. DOI: 10.1109/JPROC.2016.2528538.

[24] Gabriel Popescu. “Pixel geolocation algorithm for satellite scanner data”. In: *Scientific Papers. Series E. Land Reclamation, Earth Observation & Surveying, Environmental Engineering* 3 (2014), pp. 127–136.

[25] Chamin Nalinda Lokugam Hewage et al. “Scalability and Performance of LiDAR Point Cloud Data Management Systems: A State-of-the-Art Review”. In: *Remote Sensing* 14.20 (2022). ISSN: 2072-4292. DOI: 10.3390/rs14205277. URL: https://www.mdpi.com/2072-4292/14/20/5277.

[26] U.S. Geological Survey. *Geiger Mode LiDAR over Chicago, IL*. Accessed: 2024-12-08. Aug. 2020. URL: https://www.usgs.gov/media/images/geiger-mode-lidar-over-chicago-il.

[27] G. Priestnall, J. Jaafar, and A. Duncan. "Extracting urban features from LiDAR digital surface models". In: *Computers, Environment and Urban Systems* 24.2 (2000), pp. 65–78. ISSN: 0198-9715. DOI: https://doi.org/10.1016/S0198-9715(99)00047-2. URL: https://www.sciencedirect.com/science/article/pii/S0198971599000472.

[28] Mahmoud El Nokrashy O. Ali et al. "Generation of digital terrain model from multispectral LiDar using different ground filtering techniques". In: *The Egyptian Journal of Remote Sensing and Space Science* 24.2 (2021), pp. 181–189. ISSN: 1110-9823. DOI: https://doi.org/10.1016/j.ejrs.2020.12.004. URL: https://www.sciencedirect.com/science/article/pii/S1110982320303525.

[29] L3Harris Technologies. *Understanding Flood Models: A Guide*. Accessed: 2024-12-08. Oct. 2023. URL: https://www.l3harris.com/newsroom/editorial/2023/10/understanding-flood-models-guide.

[30] R. Valerie Ussyshkin et al. "Advantages of Airborne Lidar Technology in Power Line Asset Management". In: *2011 International Workshop on Multi-Platform/Multi-Sensor Remote Sensing and Mapping*. 2011, pp. 1–5. DOI: 10.1109/M2RSM.2011.5697427.

[31] Waheed Uddin. "Airborne Laser Terrain Mapping for Expediting Highway Projects: Evaluation of Accuracy and Cost". In: *Journal of Construction Engineering and Management* 134.6 (2008), pp. 411–420. DOI: 10.1061/(ASCE)0733-9364(2008)134:6(411). eprint: https://ascelibrary.org/doi/pdf/10.1061/%28ASCE%290733-9364%282008%29134%3A6%28411%29. URL: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%290733-9364%282008%29134%3A6%28411%29.

[32] Mathieu Dassot, Thiéry Constant, and Meriem Fournier. "The use of terrestrial LiDAR technology in forest science: application fields, benefits and challenges". In: *Annals of Forest Science* 68.5 (2011), pp. 959–974. ISSN: 1297-966X. DOI: 10.1007/s13595-011-0102-2. URL: https://doi.org/10.1007/s13595-011-0102-2.

[33] Compagnia Generale Ripreseaeree (CGR). *Compagnia Generale Ripreseaeree*. https://www.cgrspa.com/. Accessed: December 10, 2024.

[34] Leica Geosystems. *Leica CityMapper-2 High-Performance Urban Mapping Sensor*. https://leica-geosystems.com/it-it/products/airborne-systems/hybrid-sensors/leica-citymapper-2. Accessed: December 10, 2024.

[35] P. Axelsson. "DEM Generation from Laser Scanner Data Using Adaptive TIN Models". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 33 (2000), pp. 110–117. URL: https://api.semanticscholar.org/CorpusID:59795495.

[36] ASPRS. *LAS File Format Exchange Activities*. https://www.asprs.org/divisions-committees/lidar-division/laser-las-file-format-exchange-activities. Accessed: December 10, 2024.

[37] CloudCompare. *CloudCompare (3D point cloud and mesh processing software)*. https://www.danielgm.net/cc/. Accessed: December 10, 2024.

[38] Tommaso Rondini. "Towards a digital twin of Bologna: features extraction and semantic classification using LiDAR". PhD thesis. Alma Mater Studiorum - University of Bologna, 2024. URL: https://amslaurea.unibo.it/id/eprint/33368/.

[39] Yunhao Chen et al. "Hierarchical object oriented classification using very high resolution imagery and LIDAR data over urban areas". In: *Advances in Space Research* 43.7 (2009), pp. 1101–1110. ISSN: 0273-1177. DOI: https://doi.org/10.1016/j.asr.2008.11.008. URL: https://www.sciencedirect.com/science/article/pii/S0273117708005735.

[40] Hengfan Cai et al. "Systematic Comparison of Objects Classification Methods Based on ALS and Optical Remote Sensing Images in Urban Areas". In: *Electronics* 11.19 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11193041. URL: https://www.mdpi.com/2079-9292/11/19/3041.

[41] H. Gross, B. Jutzi, and U. Thoennessen. *Segmentation of tree regions using data of a full-waveform laser*. en. 2007. DOI: 10.24406/publica-fhg-357022. URL: https://publica.fraunhofer.de/handle/publica/357022.

[42] Martin Weinmann et al. "Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 105 (2015), pp. 286–304. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2015.01.016. URL: https://www.sciencedirect.com/science/article/pii/S0924271615000349.

[43] R. C. dos Santos et al. "BUILDING DETECTION FROM LIDAR DATA USING ENTROPY AND THE K-MEANS CONCEPT". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W13 (2019), pp. 969–974. DOI: 10.5194/isprs-archives-XLII-2-W13-969-2019. URL: https://isprs-archives.copernicus.org/articles/XLII-2-W13/969/2019/.

[44] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[45] Danil V. Prokhorov. "Object recognition in 3D lidar data with recurrent neural network". In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 2009, pp. 9–15. DOI: 10.1109/CVPRW.2009.5204114.

[46] Charles R. Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[47] Charles R. Qi et al. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. 2017. arXiv: 1706.02413 [cs.CV]. URL: https://arxiv.org/abs/1706.02413.

[48] Zhishuang Yang et al. "Segmentation and Multi-Scale Convolutional Neural Network-Based Classification of Airborne Laser Scanner Data". In: *Sensors* 18.10 (2018). ISSN: 1424-8220. DOI: 10.3390/s18103347. URL: https://www.mdpi.com/1424-8220/18/10/3347.

[49] Renlong Hang et al. "Classification of Hyperspectral and LiDAR Data Using Coupled CNNs". In: *CoRR* abs/2002.01144 (2020). arXiv: 2002.01144. URL: https://arxiv.org/abs/2002.01144.

[50] Hengshuang Zhao et al. "Point Transformer". In: *CoRR* abs/2012.09164 (2020). arXiv: 2012.09164. URL: https://arxiv.org/abs/2012.09164.

[51] Shuting He et al. *SegPoint: Segment Any Point Cloud via Large Language Model*. 2024. arXiv: 2407.13761 [cs.CV]. URL: https://arxiv.org/abs/2407.13761.

[52] Lucas Caccia et al. *Deep Generative Modeling of LiDAR Data*. 2019. arXiv: 1812.01180 [cs.CV]. URL: https://arxiv.org/abs/1812.01180.

[53] Kazuto Nakashima and Ryo Kurazume. *LiDAR Data Synthesis with Denoising Diffusion Probabilistic Models*. 2024. arXiv: 2309.09256 [cs.CV]. URL: https://arxiv.org/abs/2309.09256.

[54] Zewen Li et al. "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12 (2022), pp. 6999–7019. DOI: 10.1109/TNNLS.2021.3084827.

[55] Laith Alzubaidi et al. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". In: *Journal of Big Data* 8.1 (Mar. 2021), p. 53. ISSN: 2196-1115. DOI: 10.1186/s40537-021-00444-8. URL: https://doi.org/10.1186/s40537-021-00444-8.

[56] Xin He et al. "LiDAR Data Classification Using Spatial Transformation and CNN". In: *IEEE Geoscience and Remote Sensing Letters* 16.1 (2019), pp. 125–129. DOI: 10.1109/LGRS.2018.2868378.

[57] Hongbo Gao et al. "Object Classification Using CNN-Based Fusion of Vision and LIDAR in Autonomous Vehicle Environment". In: *IEEE Transactions on Industrial Informatics* 14.9 (2018), pp. 4224–4231. DOI: 10.1109/TII.2018.2822828.

[58] Hui Li et al. "CNN-Based Individual Tree Species Classification Using High-Resolution Satellite Imagery and Airborne LiDAR Data". In: *Forests* 12.12 (2021). ISSN: 1999-4907. DOI: 10.3390/f12121697. URL: https://www.mdpi.com/1999-4907/12/12/1697.

[59] Zhishuang Yang et al. "A Convolutional Neural Network-Based 3D Semantic Labeling Method for ALS Point Clouds". In: *Remote Sensing* 9.9 (2017). ISSN: 2072-4292. DOI: 10.3390/rs9090936. URL: https://www.mdpi.com/2072-4292/9/9/936.

[60] ISPRS. *ISPRS Vaihingen 3D Semantic Labeling Benchmark*. https://www.isprs.org/education/benchmarks/UrbanSemLab/Default.aspx. Accessed: December 11, 2024.

[61] F. Pedregosa et al. *scikit-learn: KDTree*. Accessed: December 15, 2024. 2024.

[62] V-H. Cao et al. "Toward a new approach for massive LiDAR data processing". In: *2015 2nd IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services (ICSDM)*. 2015, pp. 135–140. DOI: 10.1109/ICSDM.2015.7298040.

[63] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *CoRR* abs/1408.5093 (2014). arXiv: 1408.5093. URL: http://arxiv.org/abs/1408.5093.

[64] PyTorch Team. *PyTorch Documentation: torch.nn Module*. Accessed: December 26, 2024. 2024. URL: https://pytorch.org/docs/stable/nn.html.

[65] PyTorch Documentation. *torch.nn.CrossEntropyLoss*. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. Accessed: 2024-12-29. 2024.

[66] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12 (2011), pp. 2825–2830.

[67] Danni Wu, Zichen Liang, and Guang Chen. "Deep learning for LiDAR-only and LiDAR-fusion 3D perception: a survey". In: *Intelligence and Robotics* 2.2 (2022). ISSN: 2770-3541. DOI: 10.20517/ir.2021.20. URL: https://www.oaepublish.com/articles/ir.2021.20.

[68] S. Daneshtalab and H. Rastiveis. "DECISION LEVEL FUSION OF ORTHOPHOTO AND LIDAR DATA USING CONFUSION MATRIX INFORMATION FOR LNAD COVER CLASSIFICATION". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-4/W4 (2017), pp. 59–64. DOI: 10.5194/isprs-archives-XLII-4-W4-59-2017. URL: https://isprs-archives.copernicus.org/articles/XLII-4-W4/59/2017/.

[69] Nafiseh Ghasemian Sorboni, Jinfei Wang, and Mohammad Reza Najafi. "Fusion of Google Street View, LiDAR, and Orthophoto Classifications Using Ranking Classes Based on F1 Score for Building Land-Use Type Detection". In: *Remote Sensing* 16.11 (2024). ISSN: 2072-4292. DOI: 10.3390/rs16112011. URL: https://www.mdpi.com/2072-4292/16/11/2011.

[70] Faten Nahas et al. "Deep Learning Approach for Building Detection Using LiDAR-Orthophoto Fusion". In: *Journal of Sensors* 2018 (Aug. 2018). DOI: 10.1155/2018/7212307.

[71] Hiba Dishar and Lamia Muhammed. "A Review of the Overfitting Problem in Convolution Neural Network and Remedy Approaches". In: *Journal of Al-Qadisiyah for Computer Science and Mathematics* 15 (July 2023), pp. 155–165. DOI: 10.29304/jqcm.2023.15.2.1240.

[72] Jayadeva et al. *Learning Neural Network Classifiers with Low Model Complexity*. 2021. arXiv: 1707.09933 [cs.LG]. URL: https://arxiv.org/abs/1707.09933.

[73] Taiwo Oyedare et al. "Keep It Simple: CNN Model Complexity Studies for Interference Classification Tasks". In: *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, May 2023, pp. 1–6. DOI: 10.1109/infocomwkshps57453.2023.10226045. URL: http://dx.doi.org/10.1109/INFOCOMWKSHPS57453.2023.10226045.

[74] Ben Dickson. *Understanding the limits of CNNs, one of AI's greatest achievements*. Accessed: 2025-01-05. 2020. URL: https://bdtechtalks.com/2020/03/02/geoffrey-hinton-convnets-cnn-limits/?utm_source=chatgpt.com.

[75] Damien Dablain et al. *Understanding CNN Fragility When Learning With Imbalanced Data*. 2022. arXiv: 2210.09465 [cs.CV]. URL: https://arxiv.org/abs/2210.09465.

[76] Vrushali Kulkarni and Pradeep Sinha. "Random forest classifiers: A survey and future research directions". In: *International Journal of Advanced Computing* 36 (Jan. 2013), pp. 1144–1153.

[77] PyTorch Developers. *PyTorch DataLoader Documentation*. Accessed: 2025-01-05. 2025. URL: https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader.

[78] PyTorch Developers. *Working with Data in PyTorch*. Accessed: 2025-01-05. 2025. URL: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html#getitem.

[79] Thomas Grand. *torch_kdtree: GPU-accelerated KDTree for PyTorch*. Accessed: 2025-01-05. 2025. URL: https://github.com/thomgrand/torch_kdtree.

[80] PyTorch Developers. *Multiprocessing Best Practices*. Accessed: 2025-01-05. 2025. URL: https://pytorch.org/docs/stable/notes/multiprocessing.html#multiprocessing-cuda-note.

[81] Shuting Yang et al. "Crop Classification Method Based on Optimal Feature Selection and Hybrid CNN-RF Networks for Multi-Temporal Remote Sensing Imagery". In: *Remote Sensing* 12.19 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12193119. URL: https://www.mdpi.com/2072-4292/12/19/3119.

[82] Geun-Ho Kwak et al. "Potential of Hybrid CNN-RF Model for Early Crop Mapping with Limited Input Data". In: *Remote Sensing* 13.9 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13091629. URL: https://www.mdpi.com/2072-4292/13/9/1629.

[83] Ying Wang Aili Wang and Yushi Chen. "Hyperspectral image classification based on convolutional neural network and random forest". In: *Remote Sensing Letters* 10.11 (2019), pp. 1086–1094. DOI: 10.1080/2150704X.2019.1649736. eprint: https://doi.org/10.1080/2150704X.2019.1649736. URL: https://doi.org/10.1080/2150704X.2019.1649736.

[84] Yuanyuan Liu et al. "Conditional convolution neural network enhanced random forest for facial expression recognition". In: *Pattern Recognition* 84 (2018), pp. 251–261. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2018.07.016. URL: https://www.sciencedirect.com/science/article/pii/S0031320318302516.

[85] Claudio Filipi Gonçalves Dos Santos and João Paulo Papa. "Avoiding Overfitting: A Survey on Regularization Methods for Convolutional Neural Networks". In: *ACM Comput. Surv.* 54.10s (Sept. 2022). ISSN: 0360-0300. DOI: 10.1145/3510413. URL: https://doi.org/10.1145/3510413.

[86] Aoran Xiao et al. *PolarMix: A General Data Augmentation Technique for LiDAR Point Clouds*. 2022. arXiv: 2208.00223 [cs.CV]. URL: https://arxiv.org/abs/2208.00223.

[87] Vlas Zyrianov et al. *LidarDM: Generative LiDAR Simulation in a Generated World*. 2024. arXiv: 2404.02903 [cs.CV]. URL: https://arxiv.org/abs/2404.02903.

[88] Zhengkang Xiang, Zexian Huang, and Kourosh Khoshelham. "Synthetic lidar point cloud generation using deep generative models for improved driving scene object recognition". In: *Image and Vision Computing* 150 (2024), p. 105207. ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2024.105207. URL: https://www.sciencedirect.com/science/article/pii/S0262885624003123.

[89] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE]. URL: https://arxiv.org/abs/1511.08458.

[90] Towards Data Science. *Convolutional Neural Network: A Deep Dive*. https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529. Accessed: 2024-12-29. 2024.

[91] Léon Bottou. "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *Proceedings of COMPSTAT'2010*. Ed. by Yves Lechevallier and Gilbert Saporta. Heidelberg: Physica-Verlag HD, 2010, pp. 177–186. ISBN: 978-3-7908-2604-3.

[92] Yurii Nesterov. *A method for solving the convex programming problem with convergence rate O(1/k²)*. Soviet Mathematics Doklady, 1983.

[93] Zijun Zhang. "Improved Adam Optimizer for Deep Neural Networks". In: *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. 2018, pp. 1–2. DOI: 10.1109/IWQoS.2018.8624183.