Statistical Methods for Machine Learning - Skynet

Matteo Biglioli - 938199 - matteo.biglioli@studenti.unimi.it

Abstract

This paper discuss an experimental project which aims to explore the performance of different machine learning models in predicting the primary function (Residential, Commercial, Industrial and Public Services) of a building based on the LiDAR representation of its rooftop.
We will first present the dataset used in the project and explain the pre-processing applied in order to extract the features. We will then examine a features-oriented approach in which we will experiment with two well-known learning algorithms, kNN and Tree Predictors. Finally we will shift our perspective by exploiting different Convolutional Neural Networks to which we will provide the rooftops' LiDAR point clouds as if they were one-channel images.
The different approaches will be compared in the final chapter in which we will also discuss some next steps that could be taken in order to further improve the whole project.

1 Introduction

The integration of AI-powered processes in architecture and urban planning has been gaining significant attention in the past years as it offers the potential to enhance the accuracy and efficiency of building design and planning.

While at first glance the output of this study could be perceived as pointless, it should be noted that this classification step is already a central matter for both government agencies and private entities. A complete and always up-to-date dataset of this kind would enable multiple applications: from accurate estimates of the energy consumption of buildings, essential for both real-time balancing and planning processes of the different energy grids, all the way to advanced analytics needed to understand the population distribution.

LiDAR technology, which employs the use of laser beams, has established itself as a reliable method for obtaining high-resolution 3D models of objects. It is currently widely used in an incredible range of diverse applications such as autonomous driving, 3D design, speed guns and many others. In the architectural field it is the state of the art in building information modeling (BIM), urban mapping, and heritage conservation. Thanks to Figure 1 we can better understand the process employed when applying LiDAR technology to Land management use-cases.

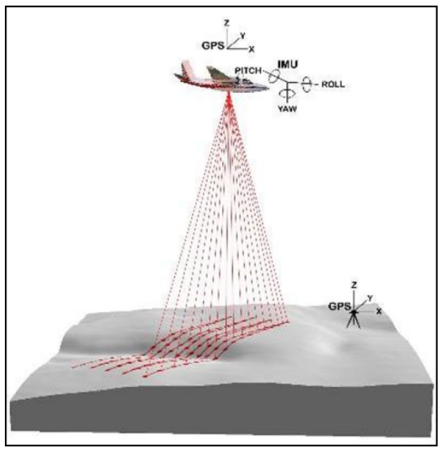

Figure 1: Aerial LiDAR mapping

In 2022, the Italian government has provided

funding for an high-precision remote sensing plan aimed at monitoring areas at high hydrogeological risk. The project aims to support the needs of different agencies[1] in managing soil defense, predicting and managing consequences of natural events in emergency cases and managing national geotopographic and security competencies.

Figure 2: Coverage of the LiDAR dataset

While this plan (executed between 2008 and 2015) did not aim to cover every residential area on the national soil, as shown in Figure 2, we naturally find an overlapping with many of them, which we exploited in this project. More specifically, mainly due to computational time/space reasons, we focused our attention on Milan, one of the biggest Italian cities.

As stated in the abstract, our goal is to train a ML model capable of predicting the primary function of a building based on the LiDAR representation of its rooftop.
We proceeded with two different approaches which will be further explained in the following sections:

- **Features-oriented Approach**: We will extract various features from the LiDAR point cloud of every rooftop, such as the highest and lowest point, which will be used to train different ML learning algorithms, mainly k-NNs and Tree Predictors.
- **Image-oriented Approach**: In this second approach we will instead look at the point clouds as one-channel images, and we will work with Convolutional Neural Networks in order to generate an hopefully enhanced classifier.

2 Dataset and Pre-Processing

The dataset on which this project is based on was provided by the Italian Ministry for the Environment. It consists in an enormous amount of raw data of the form *(longitude, latitude, elevation)* stored in thousands of *TSV* files, each one containing all the points of a single tile[2].
We can use multiple open-source Geographic Information System softwares, in our case QGIS[3], to gain a better understanding of the information encoded in these files. In Figure 3 we show the LiDAR point cloud over different buildings in which the colour represents the elevation of each single point.

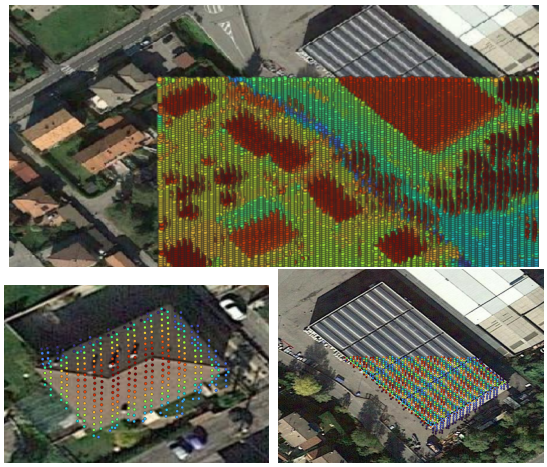

Figure 3: Examples of the LiDAR Point Cloud

[1] Mainly the Protezione Civile, an Italian organization responsible for the protection of citizens and their properties from natural and man-made emergencies and disasters.

[2] The different tiles can be seen in Figure 2.

[3] QGIS is an open-source cross-platform desktop geographic information system application that supports viewing, editing, printing, and analysis of geo-spatial data.

Before being able to actually work with the dataset, we had to retrieve among the 8TB of raw data just the files containing points overlapping the city of Milan. We did so by exploiting the metadata attached to our dataset which contained, among others, information about the bounding box of each file.

Due to the size of the input (the raw data restricted just to the city of Milan adds up to 20GB in size) we had to face multiple challenges in the pre-processing.

Firstly we identified a dataset containing the footprints of every building in the area; fortunately the Italian Government provides this kind of data in an open-source fashion directly on Github[4]. It should be noted that we also developed a small Python application which helps to retrieve footprints for a given area that can be defined with thanks to a Python-based GUI[5].

We then joined the two datasets described above, respectively the point cloud over Milan and the footprints of all the city's buildings, in order to obtain the input of the actual ML algorithms. Due to the size of our data, this step was performed exploiting PostGis[6], a spatial database extension for the PostgreSQL DBMS. In order to keep the whole process completely reproducible, we actually leveraged a small bash script[7] which, for each tile, spawns a Docker container with the *postgis/postgis* image and executes a pre-processing SQL script[8] on the points contained in the tile.

In this pre-processing step we performed the following operations:

1. We geographically joined the two datasets so that we could map each building to a subset of the point cloud that overlaps its footprint.

2. We selected just the points overlapping a building and discarded all the others, drastically reducing the size of the data we had to work with. It should be noted that we actually defined a small buffer surrounding each footprint in order to both correct for possible discrepancies in the projections and keep a ground reference for the building height estimations.

3. We computed all the features which we will present in the **Features-Oriented Approach**. This step was performed here in order to leverage the processing speed of PostGis.

4. We wrote the output of the pre-processing of each tile in a separate CSV file.

Before diving into the ML models, we performed an exploratory analysis on the data in order to check and correct possible issues. As expected we identified a strong unbalance in the dataset: in Figure 4 we show that the number of residential buildings is greater than the one of the other classes by different orders of magnitude. To solve this problem we decided to select a subset of all the buildings in such a way that each category will have the same number of buildings.

[4]There are actually 5 different repositories, each mapped to an area of the national territory. The one we used to retrieve data for Milan is https://github.com/pcm-dpc/DPC-Aggregati-Strutturali-ITC-NordOvest.

[5]This is part of a personal project, and can be found at the following link https://github.com/bigliolimatteo/skynet.

[6]https://postgis.net/.

[7]The script can be found under MSA/preprocessing/-postgis_processing.sh

[8]MSA/preprocessing/preprocessing.sql

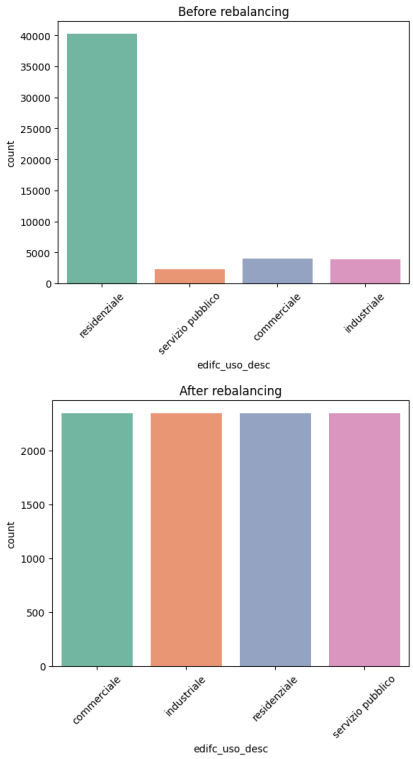

Figure 4: Classes distribution pre and post dataset re-balancing

We add just two considerations regarding this step:

- This is neither the only way to re-balance a dataset nor the best. We could have exploited different techniques, such as synthetic data generation, which could have improved our analysis overall. We choose this approach mainly to avoid adding unnecessary complexity to a project which focuses on the application of ML algorithms.
- It is critical to subset the main dataset in a random fashion, this is due to the fact that it is usual (at least in Italy) to see neighboring houses with a similar - if not identical - structure, roof included. Underestimating the nature of our dataset could always result in introducing bias in our models.

3 Features-Oriented Approach

In this first approach we are going to consider as input data different features extracted during the pre-processing step. In order to properly understand how we computed the different features we recall that for each building we kept both the LiDAR points falling directly in the footprint and the ones included in a small buffer area close to the building itself.
Here we present all the features with a brief description:

- **max_in_footprint**: elevation of the highest point inside the footprint.
- **percentile_20_in_footprint**: value of the 20° percentile of the elevation of points inside the footprint.
- **percentile_40_in_footprint**: value of the 40° percentile of the elevation of points inside the footprint.
- **percentile_60_in_footprint**: value of the 60° percentile of the elevation of points inside the footprint.
- **percentile_80_in_footprint**: value of the 80° percentile of the elevation of points inside the footprint.
- **min_overrall**: elevation of the lowest point, considering both the footprint and the buffer area.
- **building_height**: height of the building computed as *max_in_footprint* - *min_overrall*.
- **roof_height**: height of the roof estimated as *max_in_footprint* - *percentile_40_in_footprint*[9].
- **footprint_area**: area of the footprint in square meters.

While we will not comment on the basic features, we highlight that the reason why we decided to extract the different *percentile_x_in_footprint* is to encode a raw

[9]This estimation tries to correct for the fact that most of the times the footprint is not perfectly aligned with the LiDAR coverage, which results in multiple points technically overlapping the footprint which are actually at ground level.

representation of the rooftop structure. In this way we hope to help our models make a distinction between flat and uneven rooftops.

We then proceeded to project and plot the data-points in a 2-dimensional space in order to grasp the overall distribution. The plot shown in Figure 5 was obtained leveraging Principal Component Analysis over our data points, in which we used different colors to distinguish between the labels associated to each single building.
It is obvious that there is not a straightforward distinction across the four labels, and, in this specific space, the different classes seems to almost completely overlap each other. We highlight that, while this representation tells us that we cannot hope in an easy classification process based on the two components extracted from PCA, we should still proceed the application of different ML models which will work in multi-dimensional spaces defined by our various features.

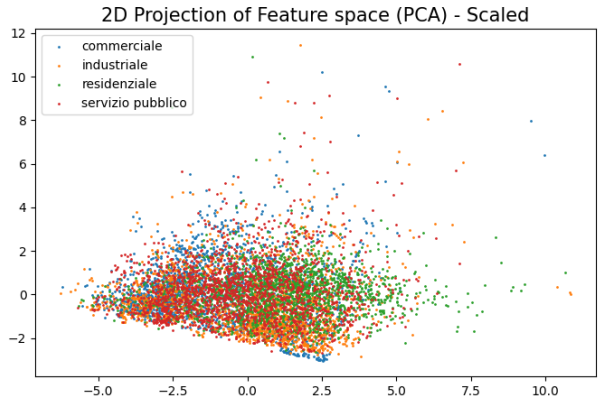

Figure 5: 2D representation leveraging Principal Component Analysis

It should be noted that before applying the learning algorithms to our data we properly re-scaled them leveraging the *StandardScaler* class from *scikit-learn*.

3.1 K Nearest Neighbors

The first learning algorithm that we will implement is the K Nearest Neighbors.
The way K-NN works is by finding the distance between a given data point and each example in the training set (which is completely stored in memory); it then chooses the K examples closest to the query point and, in the case of classification, returns the label with the highest frequency among the K selected train samples.

In order to select the best value for K, and therefore to choose the best learning algorithm, we leveraged the *validation_curve* function of *scikit-learn* which allows us to compute scores for any estimator using various values for a given parameter. In our case we let the *n_neighbors* parameter to vary in a *range(1,200,10)* in order to obtain the plot in Figure 6; we then performed an actual Grid Search (*GridSearchCV* function) over a *range(1, 75, 3)* through which we selected $K = 31$.

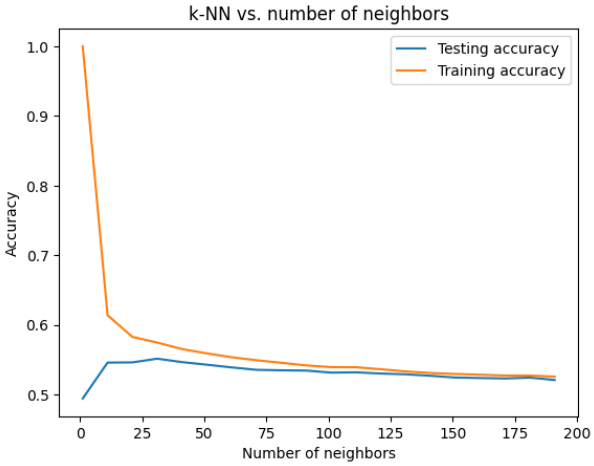

Figure 6: K-NN accuracy over different values for *n_neighbors*

Finally we evaluated the predictor in different ways:

- We computed the Cross-Validation Score (*cross_val_score* function) which gives us 0.5516.
- We plotted the confusion matrix over the test set shown in Figure 7.
- We plotted the decision boundary over the training set shown in Figure 8.

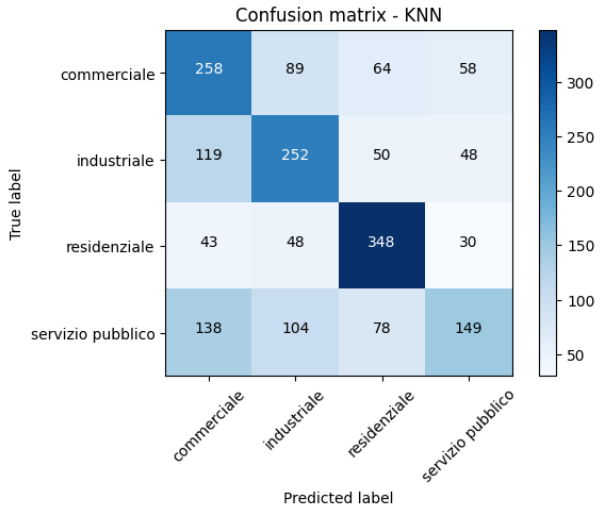

Figure 7: Confusion matrix of best 31-NN predictor

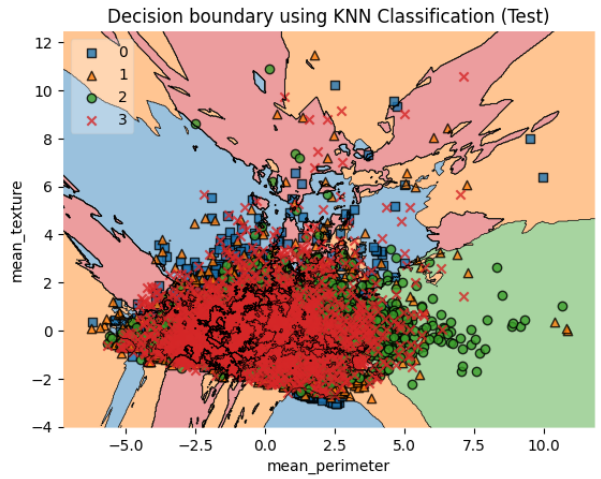

Figure 8: Decision boundary of best 31-NN predictor

Before proceeding with the next learning algorithm we will briefly discuss some remarks:

- While the CV Score is useful in order to give us an idea of how the model performs, it should not be used to compare k-NN against other learning algorithms. In order to carry out a proper comparison we should leverage the nested cross-validation technique.
- Looking at the Confusion Matrix at Figure 7 we highlight different findings:
  - The most accurate class is the *Residential* one. This is definitely expected as Italian households' rooftops tend to follow a very common pyramidal structure which differs considerably from the commercial/industrial flat rooftops.
  - While still being highly accurate, the predictor slightly confuses the commercial and industrial classes. This could be due to fact that, as stated before, their rooftop structure usually share some trademarks.
  - The less accurate class is the one related to public services; the cause of this resides definitely in the fact that the spectrum of buildings which are part of this class is exceptionally broad, spacing from schools to law enforcement offices.
- We plotted decision boundary for the sake of completeness but, as already stated before, the visualization over the two main Principal Components does not help us in any way to grasp the actual outcome of the algorithm.

3.2 Decision Tree

The second learning algorithm which we will implement is the Decision Tree.
This algorithm works by learning simple decision rules inferred from training data. In order to classify a new data point we begin at the tree's root: we compare the root attribute's value against the one of the record. We follow the branch associated with the outcome of the comparison and move on to the following node.

While we could let our tree grow indefinitely, it has been shown that extremely deep trees lead to overfitting. For this reason we decided to force the maximum depth of the tree by leveraging the *max_depth* parameter of the *DecisionTreeClassifier* function of *sckit-learn*. In order to select the best value for *max_depth* we leveraged again the *validation_curve* function

over a *range(1,30,1)* which resulted in the plot at Figure 9. As before we then performed an actual Grid Search which led us to select 8 as the best *max_depth* value for our classification problem.

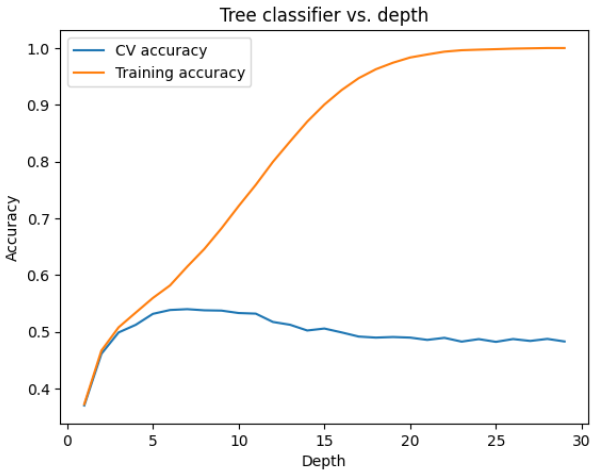

Figure 9: Decision Tree accuracy over different values for *max_depth*

Finally we evaluated the predictor in the same ways explained before, obtaining:

- A Cross-Validation Score of 0.552, which is slightly better than the k-NN one.
- A confusion matrix shown in Figure 10
- A graphic representation of the Decision Tree in Figure 11. It should be noted that the plotted tree is not actually the best Decision Tree due to size reasons[10].

[10]Because the actual best predictor has a *max_depth*= 8 its graphic representation is definitely too large to fit in a page of this paper.

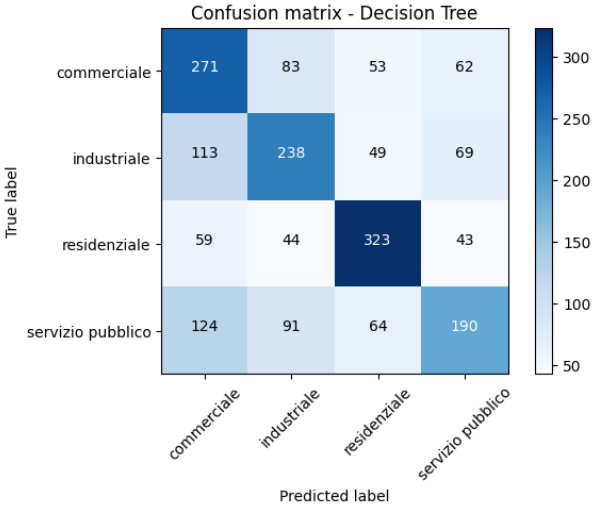

Figure 10: Confusion matrix of best Decision Tree predictor

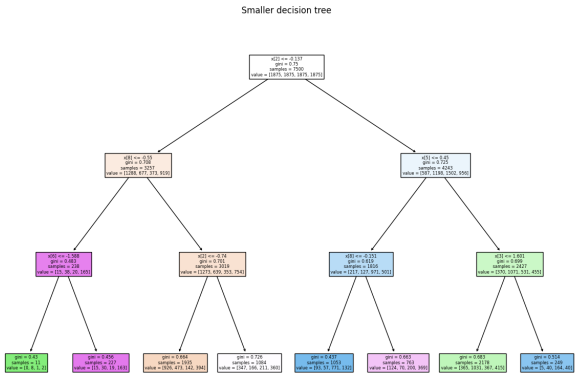

Figure 11: Best Decision Tree with *max_depth*= 8

As we did in the previous section, we will discuss some remarks before going forward with the next approach:

- It should be noted that, while we compared the Cross-Validation Score obtained here with the one from the best k-NN, this value should not be used in order to choose between the two learning algorithms as stated before.
- The confusion matrix presents the same characteristics highlighted above which we won't repeat here for obvious reasons.

Lastly we show in Figure 12 the importance of each feature, computed as the mean and standard deviation of accumulation of the impurity decrease within each tree[11]. From the plot we easily notice that the footprint area plays an important role in the classification of our data, followed by the min overall value which is strangely positioned at the second place.
While for the first feature we can obviously identify a correlation, it is harder to understand the reasons behind the importance of the min overall value. A possible explanation could be that class of buildings (e.g. industries) are located in a small neighborhood (e.g. industrial areas) which has a lower elevation with respect to the rest of the city.

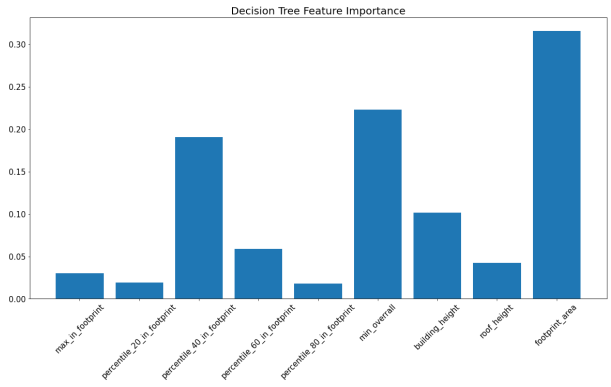

Figure 12: Decision Tree Feature Importance

4 Image-Oriented Approach

In this second approach we will shift our focus and consider the rooftops' LiDAR point clouds as if they were one-channel images. In this way our problem will turn into a image multi-class classification problem that we will try to address leveraging different Neural Network algorithms.

Before diving into the models we would need to perform a little pre-processing phase due to the fact that, because our data-points are not actual images, they do not share neither the same dimensions nor the same scale.

[11] More can be found at the following link: https://scikit-learn.org...

To solve this we performed the two following steps:

- We interpolated the elevation values leveraging the *LinearNDInterpolator* function from *Scipy* and extracted just the values on 100x100 *np.meshgrid* which led us to convert all of our LiDAR point clouds into matrices of 10000 values each[12].
- We normalized our matrices so that the elevation values of each point cloud range from 0 to 1.

It should be noted here that the two steps described above were performed on the whole dataset (without distinction of train-test). While this kind of approach would be wrong in most situations[13], because we both interpolated over a fixed grid and rescaled every image based on its own min and max value, we are not introducing any kind of dependency between train/test set. For this reason it makes sense to perform this step once on the whole dataset.

Finally we performed some basic steps in order to prepare the data to be fed into different Neural Networks, including:

1. Reshaping the whole dataset in a proper tensor shape (In our case that would be $(-1, 100, 100, 1)$).
2. Encoding the labels in a numeric format leveraging a simple *dict*. It should be noted that, because we will use the *sparse_categorical_crossentropy* as loss function, the actual values of the labels will not have any kind of geometrical meaning.
3. Splitting the dataset in Train, Validation and Test sets leveraging the *train_test_split* function from *Sckit-Learn* with a fixed random state.

[12] Note that, while this step is necessary to be sure that the input size of our network is fixed, we will lose any information related to the square area of the footprint.

[13] When we applied the *StandardScaler* during the features-based approach we fit the function just on the training set and not on the whole dataset.

We show a few examples of the images we will classify in Figure 13.

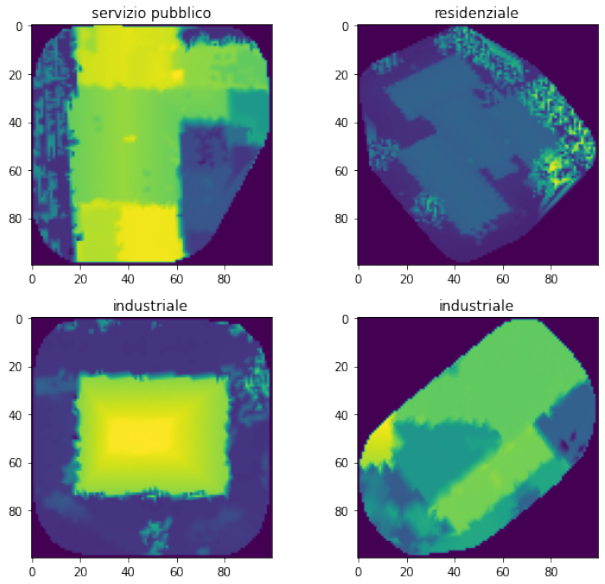

Figure 13: Samples of images of our dataset

Before diving into the actual models we need to address the topic of reproducibility: even if we tried to fix every random state leveraged during the computation in a first section called *Random Settings*, we still did not manage to get the same results over different runs. This issue seems to be caused by the way in which Google Colab deploys its notebook over various machines with different architectures. While the actual output values could vary across different runs, is should be noted that in a single run the results are consistent with the discussion we will present below.

4.1 Basic Neural Network

The first architecture that we experimented with consists in three dense hidden layers connected in a sequential fashion, as shown in the model's summary in Figure 14.

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_10 (Flatten)        (None, 10000)             0

 dense_36 (Dense)            (None, 128)               1280128

 dense_37 (Dense)            (None, 64)                8256

 dense_38 (Dense)            (None, 32)                2080

 dense_39 (Dense)            (None, 4)                 132

=================================================================
Total params: 1,290,596
Trainable params: 1,290,596
Non-trainable params: 0
_________________________________________________________________
```

Figure 14: Summary of basic NN model

As expected this model does not provide high values of accuracy as it clearly over-fits our data keeping the validation accuracy between 0.3-0.4 and providing a test accuracy of 0.35. We highlight that, for a multi-class classification problem with 4 classes, this means that the predictor performs slightly better than randomly assign a class[14].

For the sake of completeness we show in Figure 15 the training and validation metrics of this model and in Figure 16 the confusion matrix computed over the test values.
We will not discuss in depth neither the configurations of this model nor the results obtained in this section mainly because they are not that interesting as they resembles the insights that we already gained with the previous models, with a lower accuracy value. While it is obvious that we could further improve this kind of model in multiple ways[15], we will directly skip to the next section in which we will experiment with Convolutional Neural Networks which are the state-of-the-art models when dealing with images.

[14]Note that 0.25 would be the accuracy value of a model that randomly assign classes to our data points.

[15]Increasing the neurons/layers, experimenting with different activation functions/optimizers, introducing advanced techniques like Dropout layers, ...

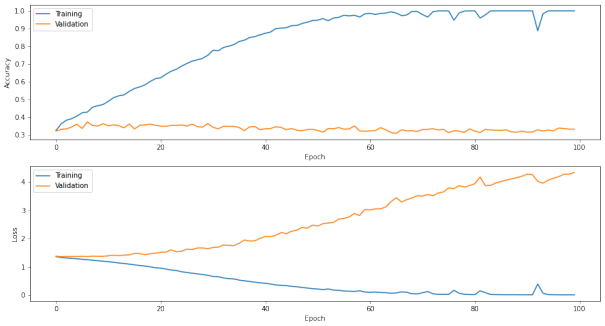

Figure 15: Training and Validation metrics of basic Neural Network predictor

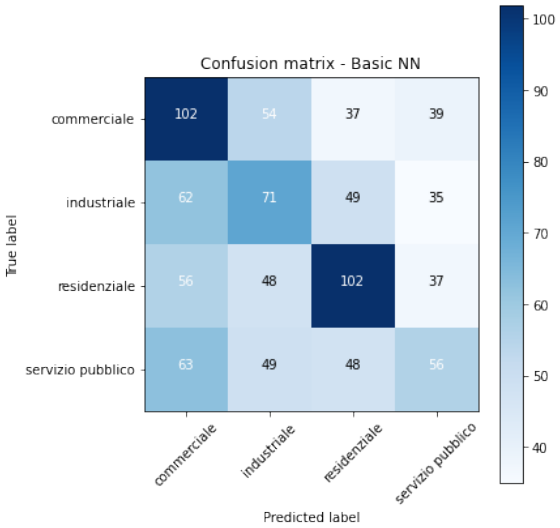

Figure 16: Confusion matrix of basic Neural Network predictor

4.2 Convolutional Neural Networks

In this section we will experiment with different CNN's architectures with the goal of maximizing the accuracy of our predictions. In order to avoid the confusion that can be generated when working with multiple learning algorithms, we decided to fix some of the hyper-parameters at the beginning of our discussion and pledge not to experiment with them in this analysis, including:

- The **loss function**: we decided to work with **Sparse Categorical Crossentropy** which is state-of-the-art when working with multiclass classification problems.
- The **optimizer**: we will use an **Adam** optimizer with a learning rate fixed at 0.0001 for all of our models.
- The **number of training epochs** will be 50 for all models, as we noticed that most of the models reach a stability in training metrics after less than 20 epochs.
- The **batch size** will be fixed at 8, as it seemed a pretty standard trade-off to maintain a reasonable training speed and still allow our model to properly generalize.

In order to properly set our expectations and goals in this last section we started by experimenting with a simple CNN which has a single convolutional hidden layer and a small dense layer, as shown in the model's summary in Figure 17.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 98, 98, 32)        320

 flatten_1 (Flatten)          (None, 307328)            0

 dense_4 (Dense)              (None, 32)                9834528

 dense_5 (Dense)              (None, 4)                 132

=================================================================
Total params: 9,834,980
Trainable params: 9,834,980
Non-trainable params: 0
_________________________________________________________________
```

Figure 17: Summary of simple CNN model

While this model still over-fits our data, it is slightly more accurate than the basic NN model discussed above, keeping the validation accuracy closer to 0.40 and providing a final test accuracy of 0.3844. We show the training metrics and the confusion matrix of this model in figure 18 and 19.

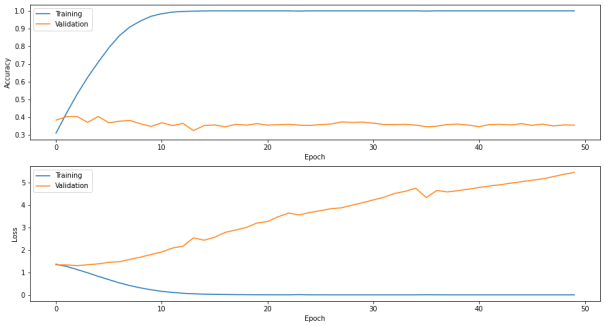

Figure 18: Training and Validation metrics of basic CNN predictor

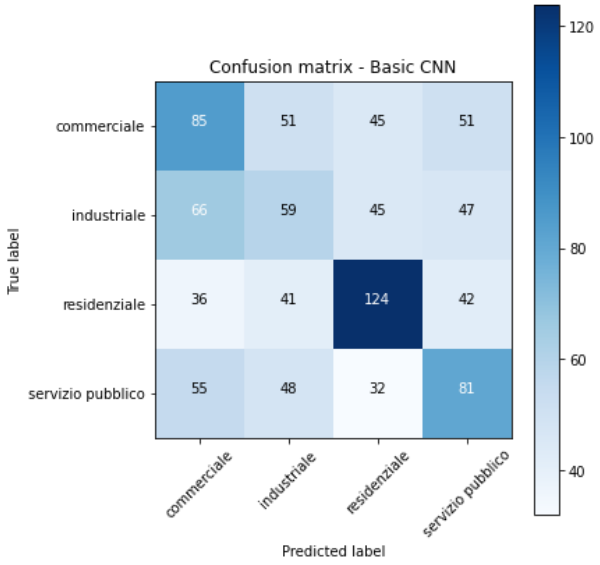

Figure 19: Confusion matrix of basic CNN predictor

As a next step in our analysis we decided to drastically enhance both the dimension and the complexity of our neural network. As shown in the summary (Figure 20), we introduced two new convolutional layers with more filters, respectively 32, 64 and 128, of the same size as the basic model[16] and a dense layer with 128 neurons.

Finally we added a *BatchNormalization* and a *MaxPooling2D*$((2,2))$ steps after each convolution for these reasons:

[16] In these initial models we kept the filter size fixed at $(3,3)$, we will experiment with increasing this value later on in the analysis.

- *BatchNormalization* is a function that normalizes each batch's mini-contributions to a layer, allowing for a stabilization of the learning process and a significant reduction in the number of epochs needed to train a CNN.

- *MaxPooling2D* is a process that filters out the dark pixels in each patch, significantly reducing the dimension of the image while keeping the pixels that we assume to contain the maximum information. In our case each patch has a 2x2 dimension, in this way the size of the image is reduced by a factor of 4 in each *MaxPooling2D* step.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)            (None, 98, 98, 32)        320

 batch_normalization_3 (Batc  (None, 98, 98, 32)        128
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 49, 49, 32)        0
 2D)

 conv2d_6 (Conv2D)            (None, 47, 47, 64)        18496

 batch_normalization_4 (Batc  (None, 47, 47, 64)        256
 hNormalization)

 max_pooling2d_4 (MaxPooling  (None, 23, 23, 64)        0
 2D)

 conv2d_7 (Conv2D)            (None, 21, 21, 128)       73856

 batch_normalization_5 (Batc  (None, 21, 21, 128)       512
 hNormalization)

 max_pooling2d_5 (MaxPooling  (None, 10, 10, 128)       0
 2D)

 flatten_4 (Flatten)          (None, 12800)             0

 dense_11 (Dense)             (None, 128)               1638528

 dense_12 (Dense)             (None, 32)                4128

 dense_13 (Dense)             (None, 4)                 132

=================================================================
Total params: 1,736,356
Trainable params: 1,735,908
Non-trainable params: 448
_________________________________________________________________
```

Figure 20: Summary of the medium-complexity CNN model

The outcomes of this model are quite disappointing if compared with the raise in complexity - and consequently training resources - that this model brings: as shown in Figures 21 the validation accuracy is now slightly higher than 0.40 while the test accuracy reaches 0.4405. While these numbers represent an improvement over the basic model, they hardly reach our expectations in terms of accuracy. If we then compare the confusion matrix at Figure 22 with the one related to the basic model we can quickly

understand that there was an actual improvement in the model's ability to properly classify our data-points; this is mainly shown in the increased density of the upper left quadrant of the matrix which, as discussed in the previous section, correspond to the most similar rooftops.

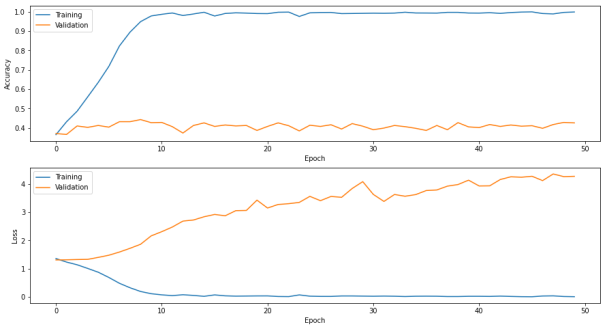

Figure 21: Training and Validation metrics of the medium-complexity CNN predictor

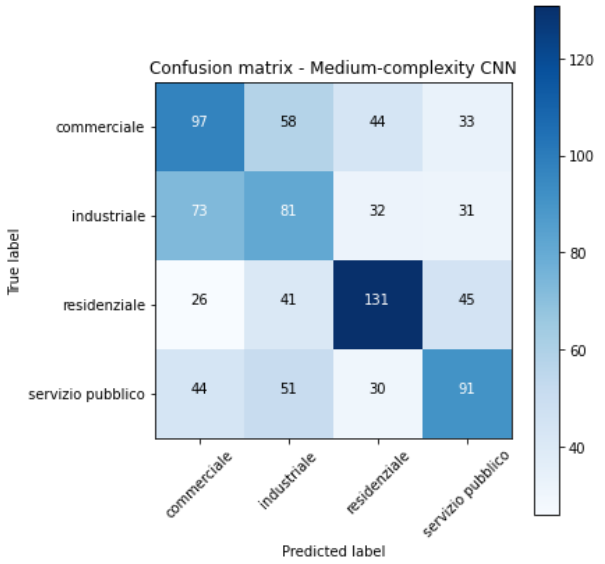

Figure 22: Confusion matrix of the medium-complexity CNN predictor

For the final step in our analysis we experimented with two advanced models based on the medium-complexity CNN described above. We decided not to increase the number of layers and neurons due to the fact that our models already over-fit the data, mainly because for both performance and balancing reasons we were not able to train on a large dataset in terms of number of data-points.
The intuitions behind these two final models are the following:

- The first technique is based merely on NN literature and focuses on the over-fitting we clearly see in all of our models. We introduced *Dropout* steps after each hidden layer that work by randomly select neurons which are ignored during training. This technique is widely used to prevent over-fitting when working with small datasets.

- The second techniques focuses more on the peculiar nature of our data. Because we had to interpolate the rooftop's point clouds, we definitely introduced some blur on the images, which could prevent our models from properly identify the roof edges when using a small $(3, 3)$-size filter. We thus decided to increase the filter size to $(10, 10)$ in all the convolutional layers while keeping the other hyper-paramers fixed.

We will not show the Keras summary of the model as we did before for space reasons but we will obviously present and discuss the resulting metrics.

In Figure 23 and 24 we show the outputs of the model which leverages a *Dropout* step with a 0.2 rate[17]. While we can clearly see the effect of the technique if we compare the slope of the training accuracy of this model with the ones above, the improvement on the validation accuracy seems negligible. This is also confirmed by the value of the test accuracy, 0.4438, which is infinitesimally better than the one of the medium-complexity CNN presented above.
If we look at the confusion matrix we see a clear improvement in the predictions based on the same intuition that we presented above related to the upper left quadrant of the matrix. Overall we can be quite satisfied with this technique due to the fact that it actually slightly improved our predictions and postponed the over-fitting of a few epochs.

[17]This value corresponds to the percentage of neurons that are dropped out by the layer.

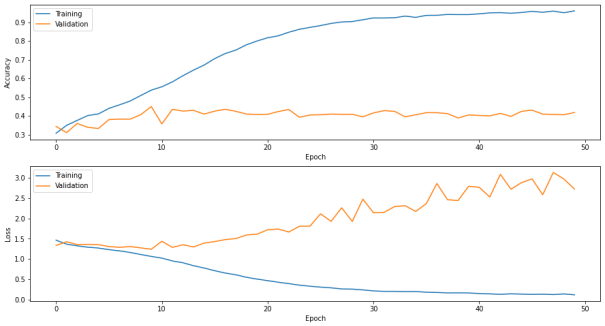

Figure 23: Training and Validation metrics of the advanced CNN predictor leveraging Dropout

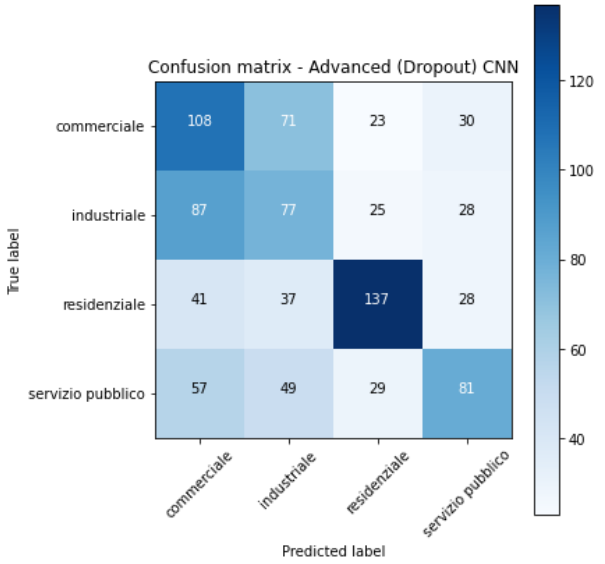

Figure 24: Confusion matrix of the advanced CNN predictor leveraging Dropout

In Figure 25 and 26 we show the outputs of the model which leverages (10, 10) filters' size. This approach definitely worsen the prediction accuracy on every side. While we cannot clearly identify a difference in the validation accuracy plot, the testing accuracy (0.3943) is not only worse than the model with *Dropout*, but it is even worse than the medium-complexity CNN. The confusion matrix confirms our first intuitions as it shows a clear worsening of the predictions.

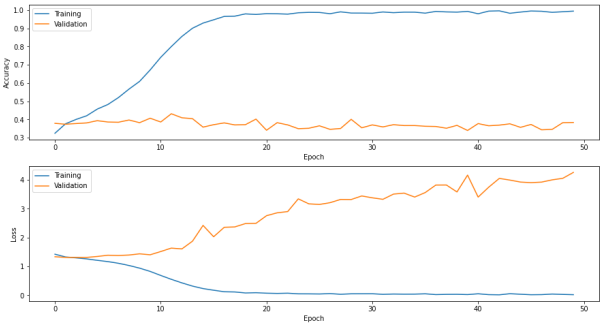

Figure 25: Training and Validation metrics of the advanced CNN predictor leveraging (10, 10) filters

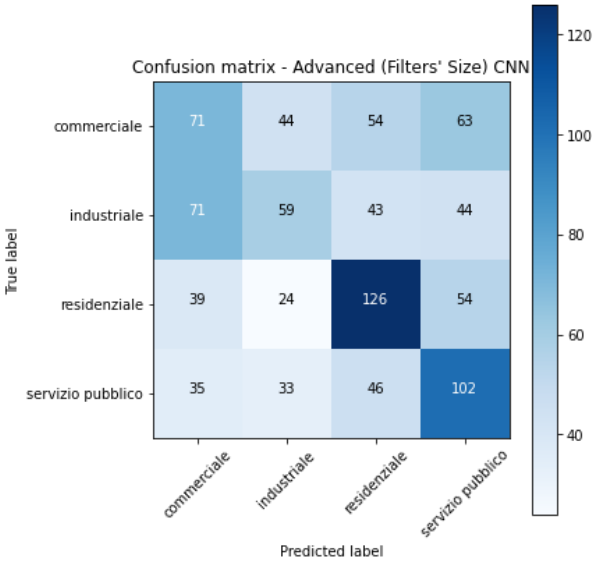

Figure 26: Confusion matrix of the advanced CNN predictor leveraging (10, 10) filters

5 Conclusion and Next Steps

In this last section we will briefly compare the models described above.

We start by highlighting that we do not have an accuracy value that we will cross-reference in the different approaches mainly for two reasons:

- In order to properly compare different learning algorithms we should leverage a more complex approach (e.g. nested cross validation) which would definitely increase the

complexity of our analysis aimed just at exploring different models.

- If we wanted to compare algorithms we should be able to provide them with the same "quantity of information" in their input data. In our case, as stated above, the NN models do not have access to the data regarding the square area of the footprint, which we know is the most important feature for the Decision Tree Classifier.

While we cannot directly compare the models, we clearly understand that the feature-based approaches perform definitely better than the neural networks and seem to provide with a similar level of accuracy. This output is evidently unanticipated as we structured our analysis to work with, what we expected to be, models with increasing accuracy.
The insights we can gain from the outcome of our analysis are:

1. The feature encoding the area of the footprint is extremely relevant in this classification process, to the point that more advanced models will perform worse if not provided with it.
2. Enhancing the CNNs with more complex layers did not improve our performance, this tells us that, instead of focusing on experimenting with different algorithms, we should focus on improving the quality of the input data we are providing the models with.
3. While our models did not reach the prediction accuracy we were hoping for, we still showed that a ML approach to this problem is feasible and could be pursued.

Here are some of the possible next steps that could be taken in order to further improve the whole project:

- We could extract new features that would improve the performance of the first models, some examples may include:
  - The distance to the closest building.
  - A greater number of percentile values.
  - The distance from the city center.
- We could experiment with different Ensamble Methods (e.g. Random Forest) which could further improve the accuracy of the feature-based models.
- We could combine different NN architectures in order to provide the models both with the LiDAR point cloud and some features (e.g. footprint area).
- We could provide the CNN also with the satellite image of the buildings.
- We could increase the area of our analysis in order to include more data-points from different cities.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.