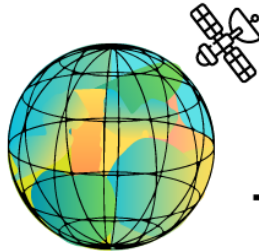




POLITECNICO DI MILANO

Geoinformatics Project
M.Sc. Geoinformatics Engineering
a.y. 2022/23



THEMATIC CLASSIFICATION

Thematic Classification QGIS Plugin

<https://github.com/Koren96/ThematicClassification>

Authors:

Mattia Koren - m. 993021

Gaia Vallarino - m. 996164

Supervisor:

Prof. Marco Gianinetto

version 1.0

Image Classification - QGIS Plugin

Mattia Koren, Gaia Vallarino

July 18, 2023

Contents

1	Description of Project	4
1.1	Purpose	4
1.2	Goals	4
1.3	Scope	5
1.4	Tools and Resources Used	6
1.4.1	GDAL	6
1.4.2	scikit-learn	6
1.4.3	Plugin Builder	7
1.4.4	Qt Designer	8
2	Dataset Description	9
3	Plugin	12
3.1	Installation Instructions	12
3.1.1	Requirements	15
3.2	Plugin Overview and GUI Guide	15
3.3	Overview of Algorithms	20
3.4	Workflow Examples	20
3.4.1	Common Workflows	20
3.4.2	Common Errors	21
4	Support and Contacts	23
A	Definitions & Acronyms	24
B	Case Study	25
B.1	Introduction	25
B.2	Dataset Description	25
B.3	Case 1: Classification with Creation of Training Model	26
B.3.1	Results	28
B.3.2	Possible Areas of Improvement	30
B.4	Case 2: Classification with Pre-Trained Model	31
B.4.1	Results	32

B.4.2 Conclusions	32
C Programming Guide	34

Abstract

In the QGIS environment, the Thematic Classification plugin is a powerful tool for classification tasks on geospatial data.

This plugin makes use of GDAL (Geospatial Data Abstraction Library) and scikit-learn, two popular libraries for machine learning and geospatial analysis. The quick loading, processing, and manipulation of geographic data is made possible by GDAL, while a wide range of classification techniques and assessment measures are offered by scikit-learn.

A user-friendly interface is provided by the plugin for data input, model training, preprocessing, and result visualization. It supports a number of classification algorithms, such as Support Vector Machines, K-Means, and Gaussian Naive Bayes.

Users may quickly categorize geospatial data, assess model performance, and produce classified outputs for additional analysis and decision-making with the help of the Thematic Classification plugin.

1 Description of Project

1.1 Purpose

This is the user documentation for the QGIS plugin **Thematic Classification**, which has as a main goal the classification (both supervised and unsupervised) of land cover classes on satellite imagery.

The purpose of this document is to help the user familiarize, learn how to use the plugin, and identify the main errors that might arise from its use.

In the appendices of this document it is also possible to find a case study scenario "(**Appendix B**: Case Study)" , and a guide for anyone who wants to improve the existing code "(**Appendix C**: Programming Guide)" .

In "**Appendix A**: Definitions and Acronyms", you can find an overview of terms and abbreviations that might be unfamiliar to the beginner user.

1.2 Goals

The *nameplugin* QGIS plugin aims to provide a user-friendly and efficient tool for performing supervised and unsupervised classification of land cover classes starting from satellite imagery on the QGIS environment.

The plugin uses scikit-learn, a famous python machine learning library, to provide a selection of classification algorithms best suited for Earth Observation tasks and the analysis of geospatial data. The plugin also employs the gdal library (Geospatial Data Abstraction Library) to enhance the performance and efficiency of the classification.

A more in-depth overview of the libraries and tools used in the project can be found in "Section 1.4 - Tools and Resources Used".

The primary goals of the project are the following:

1. **Easy-to-Use Interface**: the GUI is built following a specific work-

flow that strives to be the most user-friendly, without requiring extensive knowledge of machine learning topics. The goal would be for the user to directly start working with the application without needing to have an in-depth study of the documentation, ensuring a familiar user experience.

2. **Diverse Classification Algorithms:** the application wants to support a wide range of classification algorithms. While at the moment mostly supervised classification are implemented (e.g., Support Vector Machines, Gaussian Naïve-Bayes, Neural Networks, and many others), the plugin also supports unsupervised classification methods – like K-Means Clustering, and it is available for further implementations, thanks to our handy programming guide.
3. **Flexible Data Handling:** the plugin can handle different types of data commonly used in supervised classification for earth observation, like satellite imagery and remote sensing data. It provides ample functionalities to load and pre-process the input data, including feature extraction and data splitting.
4. **Result Visualization and Assessment:** the application is able to provide interactive maps and statistical metrics to better evaluate and understand the accuracy of the results, so that the user can take more informed decisions.
5. **Documentation and Support:** the project wants to provide comprehensive documentation to everybody approaching the plugin, be them a perspective user or someone who wants to enhance it, address issues, or expand its functionalities.

1.3 Scope

The goal of this project is to create a classification tool utilizing machine learning methods for remote sensing data. Users of the tool are able to create rule images, classify image data using the tool, and train models. It supports classification techniques like Gaussian Naive Bayes and K-Means.

Large datasets are processed individually by the tool, which also offers updates on its progress during the training and classification phases. It also has features such as log reporting, accuracy evaluation, and model storing.

The project's goal is to offer an efficient and practical approach for classifying land cover in remote sensing applications.

1.4 Tools and Resources Used

1.4.1 GDAL

The GDAL library is used by the Thematic Classification plugin to increase classification effectiveness. An open-source library called GDAL provides capabilities for manipulating geographical data.

GDAL is crucial to the plugin in the following areas:

1. **Data loading and access:** GDAL offers methods for importing and using geographic datasets, such as satellite images. The plugin easily reads and extracts data using GDAL. Data transformation and manipulation capabilities are provided by GDAL for preparing and modifying input data. Operations like reprojection, resampling, cropping, and data subset extraction fall under this category
2. **Support for Data Formats:** GDAL enables users to deal with a variety of sources by supporting a number of geographic data formats. The plugin makes use of GDAL's format support for simple workflow integration with classification.
3. **Performance Optimization:** GDAL's optimized algorithms and data structures enhance classification speed, even for large datasets. This optimization improves efficiency in analyzing and classifying geospatial data.

By incorporating GDAL, the Thematic Classification plugin provides advanced capabilities, performance optimizations, and a smoother workflow, enhancing user experience and productivity.

1.4.2 scikit-learn

Scikit-learn, a powerful machine learning package, is employed by the Thematic Classification plugin to improve the classification procedure. For data analysis and modeling, scikit-learn offers a comprehensive set of tools and algorithms for data analysis and modeling.

Scikit-learn is essential to the plugin's operation in the following areas:

1. **Algorithm Selection and Training:** scikit-learn provides a variety of classification techniques, including support vector machines, random forests, and decision trees. These algorithms are used by the plugin to train classification models with user-specified parameters.

2. **Model Evaluation and Validation:** scikit-learn offers tools for assessing and testing the effectiveness of categorization model performance. These features are used by the plugin to evaluate the precision and dependability of the trained models.
3. **Predictive Analysis:** Once trained, scikit-learn’s classification models can be used to predict the classes of incoming input data. This ability is used by the plugin to categorize geospatial data using the taught models.
4. **Integration and Compatibility:** scikit-learn is a favoured option for machine learning jobs because of its smooth integration with other scientific libraries in the framework of Python. To provide seamless integration and interoperability with other elements of the classification workflow, the plugin makes use of scikit-learn’s compatibility.

The Thematic Classification plugin uses scikit-learn to give users access to a variety of classification methods, model evaluation tools, and prediction capabilities, enabling accurate and efficient geospatial data classification.

1.4.3 Plugin Builder

Plugin Builder had the following role in the creation of the Thematic Classification plugin:

1. **Project Structure:** PB assists in building a well-structured project with specified directories and files, laying the groundwork for plugin development.
2. **Plugin Template:** PB provides a plugin template that contains the required template code, enabling developers to easily begin constructing the plugin without having to start from scratch.
3. **Plugin Configuration:** PB offers a user-friendly interface for setting plugin metadata, defining dependencies, and specifying plugin settings, which helps with plugin configuration.
4. **Code Generation:** To save time and effort during development, Plugin Builder automates the creation of code snippets for plugin functionality such menu items, toolbar buttons, dialogs, and event handling.
5. **Documentation Generation:** PB assists in the generation of documentation templates, making it simpler to record the features, applications, and API of the plugin for future use.

By utilizing Plugin Builder’s features, the Thematic Classification plugin’s development process is made more simplified, effective, and maintainable. This frees up developers to concentrate on creating the plugin’s essential features.

1.4.4 Qt Designer

The Thematic Classification plugin uses Qt Designer, which comes together with the QGIS software, in the following ways:

1. **Visual interface Design:** Qt Designer is a potent tool for designing graphical user interfaces (GUIs). It offers a drag-and-drop interface for adding and organizing UI elements, allowing us to design and alter the plugin's aesthetic appearance.
2. **Widget Customization:** Qt offers a large selection of pre-built widgets that may be quickly modified to match the desired design. It offers possibilities to change widget settings, specify event handling, allowing us to design a user interface that is both aesthetically pleasing, efficient and user-friendly.
3. **Signal-Slot system:** The signal-slot system enables smooth communication between UI elements and underlying functions. Qt makes it easier to implement this mechanism by easing the way in which user actions, like button clicks or menu selections, are linked to the associated plugin actions and functions.
4. **Integration with Qt Framework:** The Qt framework, on which the creation of the plugin is based, is smoothly integrated with Qt Designer. The UI design developed in Qt Designer and the code used in Qt-based apps are compatible and consistent thanks to this connection.
5. **Simple Synchronization Design Process:** We can quickly prototype, test, and improve the plugin's interface thanks to Qt Designer's support for an iterative design approach. It is simple to sync changes made in Qt Designer with the code base, enabling productive cooperation between designers and engineers.

We may design a visually beautiful and simple user interface for the Thematic Classification plugin using Qt Designer.

2 Dataset Description

The geospatial datasets that the Thematic Classification plugin works with often contain raster data, such as satellite imagery or data from remote sensing. These databases gather data about the surface of the planet and offer important insights for a variety of applications, such as the classification of land cover, vegetation study, urban planning, and environmental monitoring.

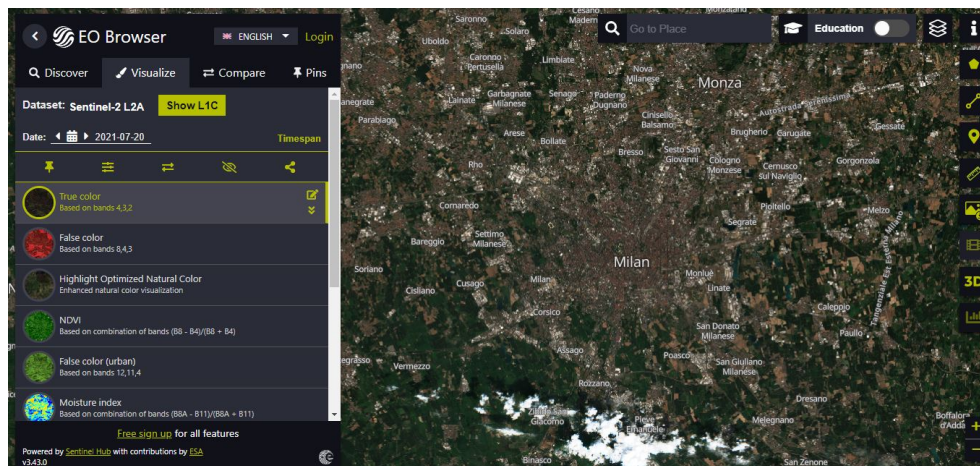
Geospatial datasets often include the following essential components:

- **Coordinates:** Geospatial datasets are fundamentally spatially referenced, which means they include details on the location and size of the collected data. For the data to be correctly analyzed and understood in its geographic context, this information is essential.
- **Spectral Bands:** Sensors that record data in a variety of spectral bands are frequently used to acquire satellite pictures and remote sensing data. Each band denotes a particular range of wavelengths and offers particular knowledge about the Earth's surface. These spectral bands may include the visible, NIR, SW, and TIR bands.
- **Pixel Values:** The data in the dataset is represented as pixel values, which are the values for each spectral band that have been measured or estimated at a particular place. Depending on the data pre-processing techniques used, these pixel values can range from digital numbers to radiance or reflectance values.
- **Metadata:** Geospatial datasets frequently have metadata, which gives extra details about the dataset, such as the date of capture, sensor characteristics, the coordinate reference system, and any quality assessment metrics that may be provided. Understanding the data's origin, quality, and constraints requires this metadata.

It is crucial to adequately preprocess and prepare the data before using it for classification. To maintain consistency and compatibility across various datasets, this may involve operations like data normalization, atmospheric correction, geometric correction, and spatial resampling.

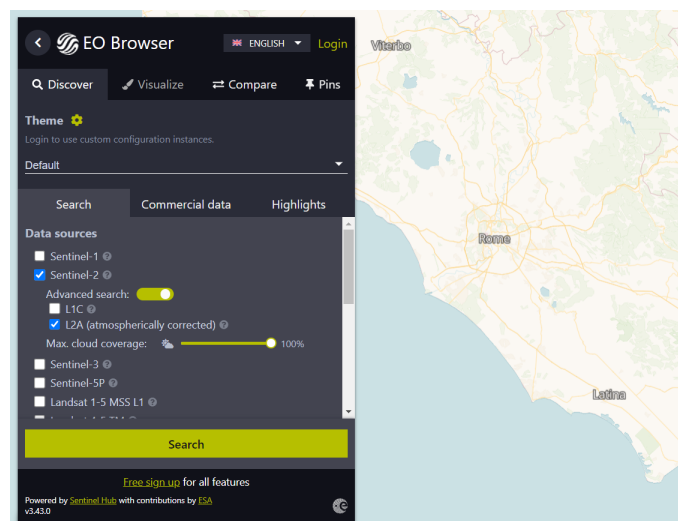
The dataset used for the construction of the plugin had resolution of 10m, size of 10980x10980 pixels, were multiband (4 bands: Blue, Green, Red and NIR). The file is a raster and the value contained in each pixels is a digital number which is obtained from the BOA reflectance.

In particular, the plugin expects as input single band raster layers. The ones we have used in Section 3 and for the Case Study come from the Sentinel Hub EO Browser. The page presents as follows:

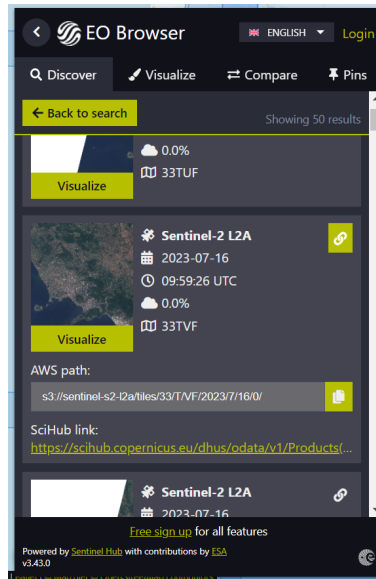


You then need to go on the left-hand menu and click on **Discover**.

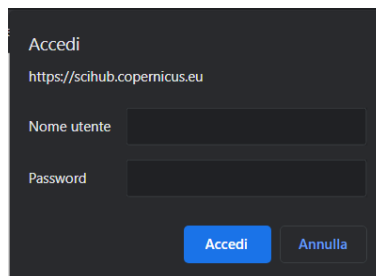
On the page that opens up, you need to select **Data Sources**, Sentinel 2 (or any other satellite you might be interested in), select **L2A** and finally choose a percentage of **maximum cloud coverage** and hit **Select**.



Afterwards you have to select an image, and click on the icon on the right-hand corner.



You might need to create an account on EO Browser if you don't already have one. The download will immediately start after you input your credentials.



The input image will be download in .j2p format. The plugin will output the classified images in a georeferenced .geotiff file.

The rule images, which are one of the possible outputs of our plugin, are images containing values representing the probability or accuracy of the single classified pixel, will be discussed more in depth later on in Section 3.

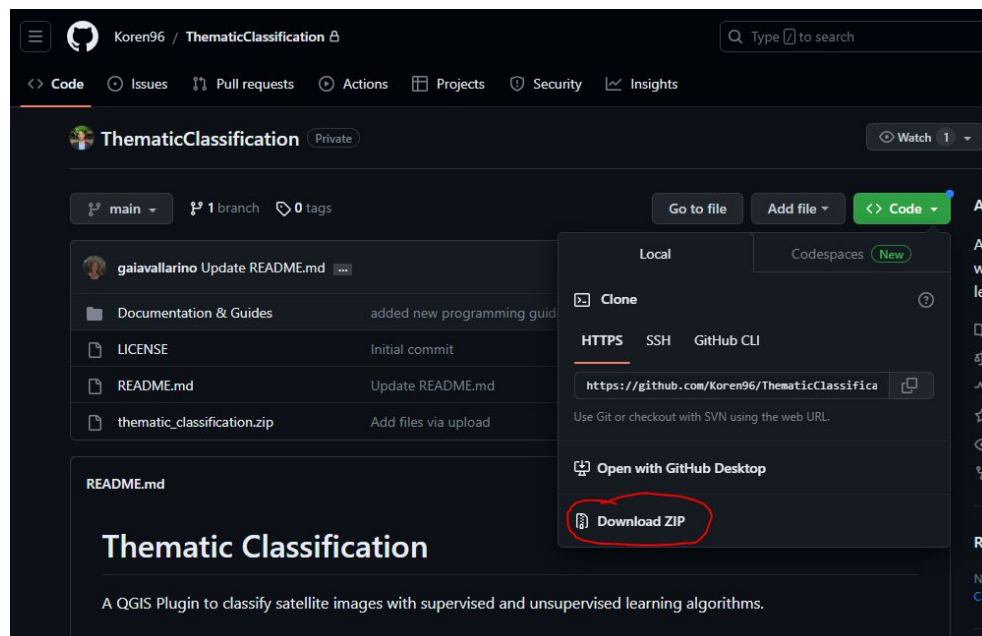
3 Plugin

3.1 Installation Instructions

Please use the following methods to install the Thematic Classification plugin:

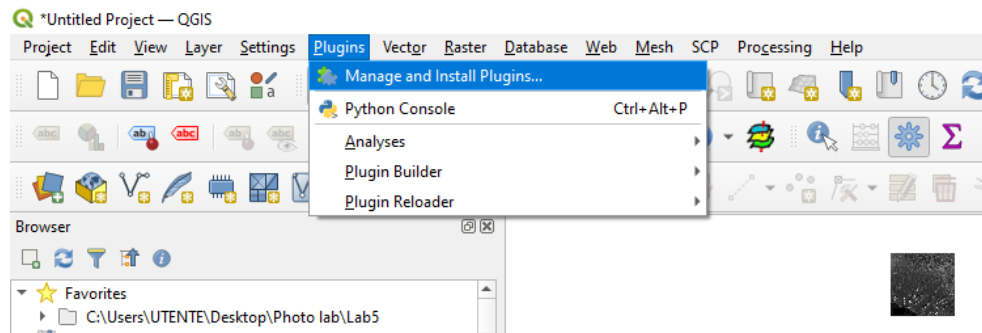
1. How to Download the Plugin:

- Visit the GitHub repository at:
<https://github.com/Koren96/ThematicClassification>
- To download the complete repository as a .zip file, click on "Download ZIP" from the "Code" menu.
- Save the .zip file onto your computer.



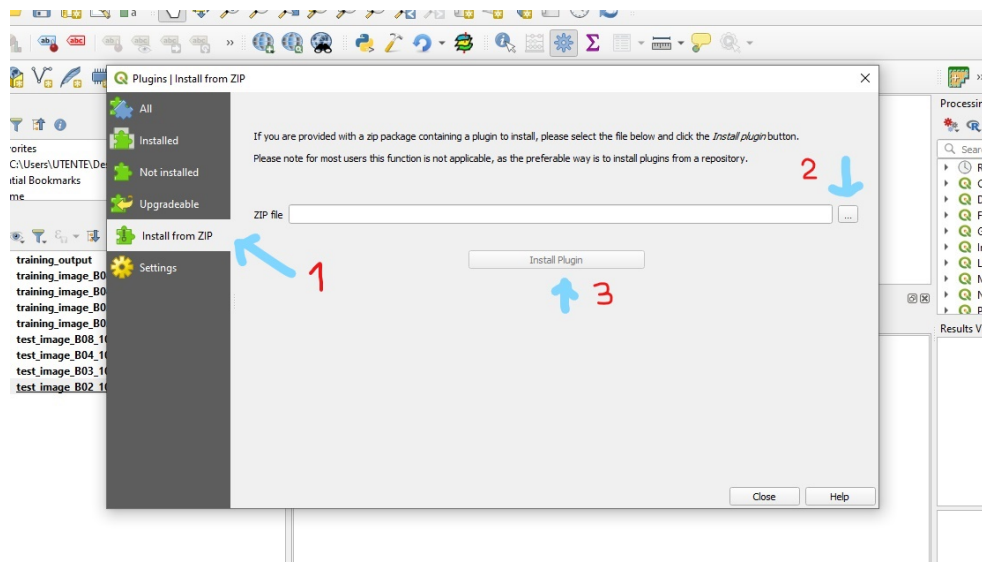
2. QGIS Plugin Manager launch:

- Start QGIS
- Access the Plugin Manager, from Plugins menu - Manage and Install Plugins



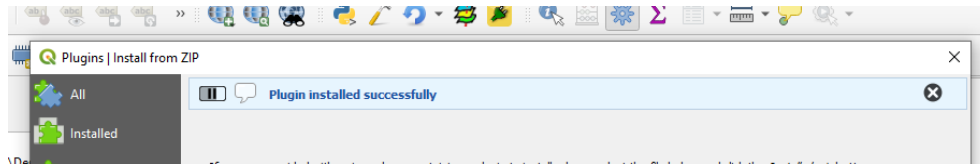
3. Installing from ZIP file:

- Find the "Install from ZIP" section in the Plugin Manager. (1)
- Next to the input field, click the "..." button. Browse to and choose the.zip file from step 1's download. (2)
- To start the installation procedure, click on "Install". (3) If the installation procedure was successful, you will see a blue bar on top of the dialog window, as shown in the figure below.



4. Taking Care of Installation Errors:

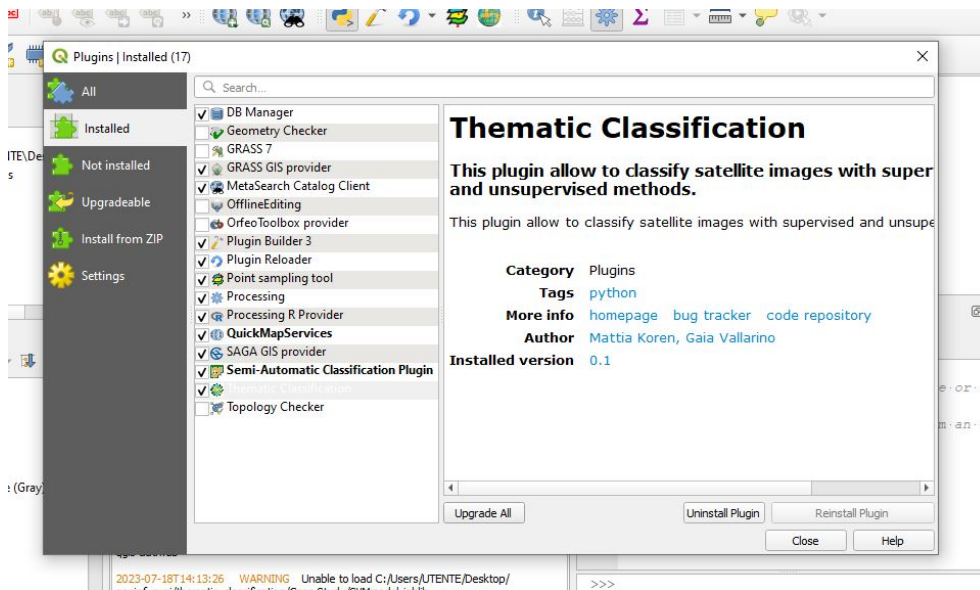
- If an installation issue occurs, it can be due to a dependency or missing library. Make sure your system has all the necessary libraries installed.



- You can install the dependencies from the *OSGeo Shell* (you can find it in the same folder where QGIS was installed) or from the *command line*. Further information about this can be found in the section "**Requirements**"

5. Open the Plugin:

- Close the Plugin Manager once the installation is complete. If you want to double-check the successful installation, you can go on the "Installed" tab of the plugin manager. It should be in the list of plugins.



- Go to the "Plugins" area of the QGIS menu and choose "Thematic Classification" from the list of available plugins.
- The Thematic Classification plugin may now be used to carry out classification tasks inside of QGIS.

3.1.1 Requirements

In order to avoid installation issues, here is a list of required libraries and modules that need to be present before installing the plugin:

Name	Version	Command
scikit-learn	latest	<code>pip install sklearn</code>
GDAL	LSR	<code>pip install gdal</code>
numpy	1.25 (LSR)	<code>pip install numpy</code>

On **Linux**, QGIS uses the main Python installation; therefore, one should only run `pip install my-package` (pip must be installed) in the command line.

On **Windows**, QGIS uses its own version of Python, so the libraries must be installed using the right version of it. You should navigate in your folders to `C:/QGIS/apps/Python27/` or `C:/QGIS/apps/Python37/`. You should then open the powershell (OSGeo Shell) and type: `python -m pip install my-package`. You should substitute `my-package` with whatever library name you wish to install.

Please note that GDAL should automatically be implemented in QGIS and thus should not require installation.

3.2 Plugin Overview and GUI Guide

In the above figures you can see the main view of the Thematic Classification plugin.

In the following paragraph the different widgets and buttons and their functioning will be explained in detail. The description will proceed in the order in which the widgets appear.

1. **Import Classification Model:** this is an optional widget that allows the user to use a pre-trained classification model, in case they already have one they want to use on the classification they are about to perform. Clicking on this button will enable the "Select a Model" function underneath, which would be otherwise non-selectable.

Attention

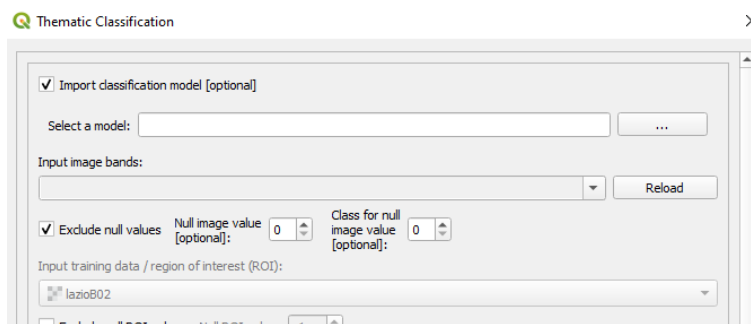
If this option is selected, the other options discussed below, up to "Output Classification Image" will be non-selectable



2. **Input Image Bands:** this is a mandatory selection widget, which allows the user to choose among the imported layers on QGIS which ones the classification is going to be performed on. The "Refresh" button next to the widget allows for new layers to be added to the pull-down list, in case any were added while the plugin window was open.

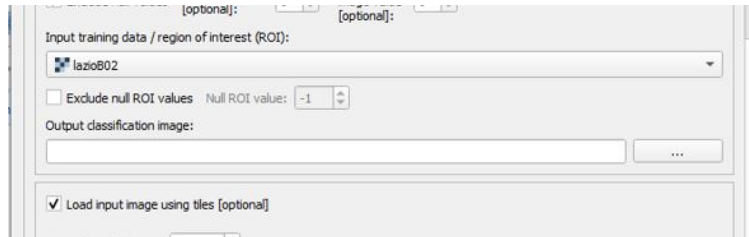
The plugin expects in input singleband raster images in raster format. In particular, the developers have described in section "Dataset Description" where to download the same files used by them during the development of the use cases and case study.

- Null Image Value: is an optional function that allows to select which value in the input images will be recognized as "null" in the image
- Class for Null Image Value: is an optional function that allows the user to select which class the null value are mapped to.

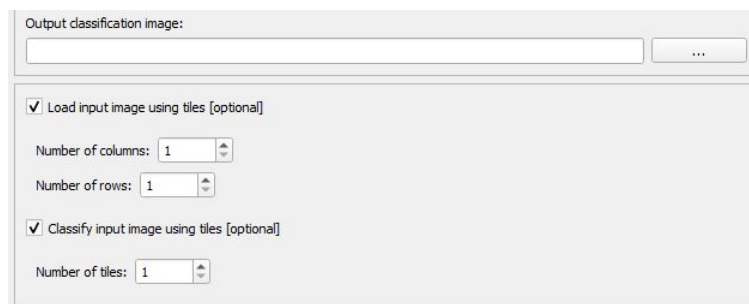


3. **Input Training Data / ROI:** In this widget it is possible to select the layer with already classified samples that will be used to perform the supervised classification in case a pre-existing model was not selected. This layer will have to be open in QGIS.

- **Null ROI Value:** if in the classification there were any samples of null classes, this is where the user can input the relevant value in said layer.



4. **Output Classification Image:** in this part of the GUI, the user can select where the output classified image will be saved. The output will be saved in .tiff format.
5. **Load Input Image Using Tiles:** this is an optional command. If the user selects it, they can choose how many rows and columns to divide the input image in when it gets loaded into memory. This is done to ensure a smooth and efficient pre-processing of the image, in particular in case of big files, which would otherwise be quite computationally heavy.
6. **Classify Input Image Using Tiles:** this is an optional command. If the user selects it, they can choose the number of tiles to divide the file in when going through the classification phase. This is done to ensure a smooth and efficient classification of the image, in particular in case of big files, which would otherwise be quite computationally heavy.



7. **Select Classification Algorithm:** in this section, the user is prompted to select a classification method.
 - The user may choose between unsupervised and supervised methods.
 - The user is able to review the documentation of the method on the scikit-learn page, via the "Help" button that becomes clickable once an algorithm is selected.

- The user may choose to use custom parameters in the selected algorithm, by clicking the "Use Custom Classification Parameters" button, and then switching the parameters to be employed.

Disclaimer

We highly suggest that only expert users modify the parameters, as this requires a level of expertise that would otherwise risk making the procedure too slow, or the machine non-responsive.

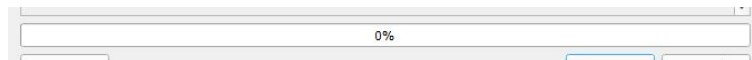
A more in-depth analysis of the classification algorithms can be found in subsection "Overview of Algorithms"

8. **Calculate Classification Model Accuracy:** by selecting this option, the user will receive a percentage value of accuracy of its classification model. This is possible only if the user has not selected to use a pre-existing model.
9. **Save Classification Model:** by selecting this button, the user will save the classification model that was just trained. It will be saved as a .joblib file.
10. **Save Rule Images:** this button allows the user to generate and save the rule images for the classification that is taking place. A rule image contains values of probability of each pixel of belonging to a certain class. It is a useful tool in case the user wants to have a more in-depth look of why one pixel was classified under a certain class.

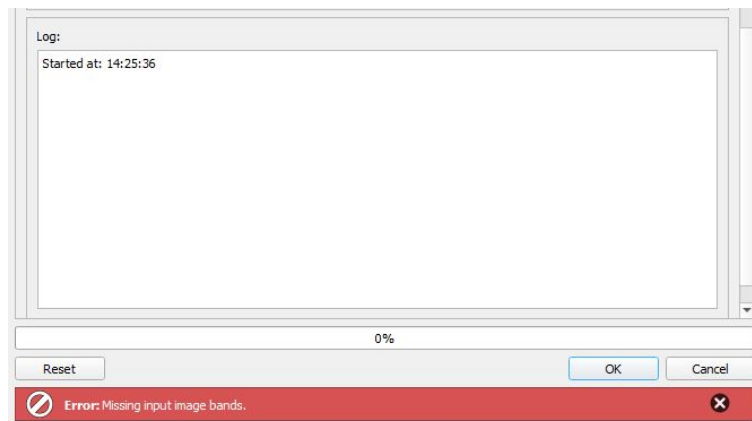
11. **Log:** the Log panel shows certain clou aspects of the process, such as the timesteps for the different processes, an overview of the parameters used, and much more useful information. Via the "Save Log (.txt)?" button, the user can decide to save the log file, which might come useful later on to remember what was set during a particular run of the application.



12. **Progress Bar:** the progress bar is implemented for the user to see the progress of the process once the OK button is pressed and the code is running.



13. **Message Bar:** the message bar only comes out when a critical error or a status update happens, after pressing OK. The critical errors may be reviewed at the section "Common Errors" of this document.



3.3 Overview of Algorithms

Specifics				
Algorithm	Type	RI Output	RI type	RI format
K-Means	Unsupervised	Float 32-bit	$\propto Distance$.tiff
Gaussian Naive-Bayes	Supervised	Int 16-bit	Probability	.tiff
K-Nearest Neighbors	Supervised	Int 16-bit	Probability	.tiff
Support Vector Machine	Supervised	Float 32-bit	Decision Function	.tiff
Neural Networks	Supervised	Int 16-bit	Probability	.tiff
Logistic Regression	Supervised	Int 16-bit	Probability	.tiff
Random Forest	Supervised	Int 16-bit	Probability	.tiff

In particular, in the RI we can have three different metrics:

- **Probability:** uses the function *predict_proba*
- **Decision Function:** uses a score that is proportional to the distance between the point and the hyperplan/border of the SVM
- **Distance:** uses the function *transform*, which gives the distance between the point and the cluster centroid.

3.4 Workflow Examples

3.4.1 Common Workflows

This plugin can help the user with two main type of workflows:

- The classification using an ad hoc trained model
- The classification of an image using a pre-existing, pre-trained model.

Both cases are well explained, with step-by-step instructions and comments in Appendix C: Case Study.

3.4.2 Common Errors

In this section we will discuss some of the most common errors that might happen while using the plugin.

Most of the errors that could come up would derive from the wrong selection and/or input of commands before even running the program. The alerts for these errors will come out directly in a message bar in the plugin GUI.

These include:

- **Missing input image bands:** the process was run but no input image were selected. Go back and check which image to perform the training on.
- **Missing output classification image name:** the path to the savefile was not created, so the output has no name and cannot be saved anywhere. Go back and select where to save the output and what to name it.
- **Number of Rows and Columns must be greater than 1:** the user has selected an incorrect number for the opening of the file. The number must allow for a division to exist, so it must be greater than or equal to 1. It is trivial that it also has to be integer.
- **Number of Tiles must be greater than 1:** for the same reasons as the above-mentioned, the number of tiles to divide the image in must be integer and positive.
- **Missing classification model name:** the user desires to perform a classification using a pre-trained model, but has chosen no model. Go back and select the model.
- **The loaded model expects X features:** there is an incongruence between the model chosen to perform the classification and the input layer to be classified. For example, if the model was trained on 4 bands, and you input an image with only 2 bands, it won't be able to perform the classification.
- **Missing output classification image:** the user has not chosen where to save the output of the classification and, consequently, its name. Go back and choose where to save the output and how to call it.

Some other common errors that the user may incur into is if they decide to change the parameters of the algorithms.

Some of the most common mistake that may present are, for example, if the user decides to set a parameter to "none" instead of "None". In this case, the program won't be able to identify the parameter, and the procedure will raise an alert in the python console of QGIS, that will abort the procedure.

Other common errors with the parameters might be inserting invalid values, for example strings instead of integer numbers, and so on.

For this exact reason, we suggest that beginner users, or people who have not read the documentation of scikit-learn, abstain from changing the parameters, as we cannot foresee the outcome of a misuse of these (e.g., putting a number too big or too small of layers while using the Neural Networks method might incur in erroneous results, or in the machine not responding).

4 Support and Contacts

In case you have any doubts or questions about the project, you can find us at the following email addresses:

Name	Email Address
Mattia Koren	korenmattia@gmail.com
Gaia Vallarino	gaiavallarino1@gmail.com

The repository for this project can be found **here**.

The project was supervised by Prof. Marco Gianinetto of DABC - *Department of Architecture, Built environment and Construction engineering* of Politecnico di Milano..

A Definitions & Acronyms

Specifics	
Term	Description
RI	Rule Image: is a multiband image that gives for each pixel the probability with which it was predicted to be part of a certain class. If you have 10 classes, each pixel will contain ten values of probability. In case of SVM and XXXXX the values stored inside the pixels are not probabilities, but distances.
ROI	Region of Interest: refers to a specific area or subset of the remote sensing data that is selected for further analysis. In our case, it is a polygon with a class label assigned, that will be used for training purposes.
BOA	Bottom of Atmosphere: Lowest point of the atmosphere, where the ground meets with the Troposphere

B Case Study

B.1 Introduction

In this case study we are going to classify two different Sentinel-2 images using a Supervised method for classification from the Thematic Classification plugin.

For the first image we will create a training dataset from scratch, fit it into the model, and then classify the entire image through the trained model.

For the second case we will directly classify the image through the same trained model.

B.2 Dataset Description

Sentinel-2 L2A images of an area of the Liguria region in Italy:

- *Resolution*: 10m
- *Size*: 10980x10980 pixels
- *Bands*: liguriaB02.jp2 (Blue), liguriaB03.jp2 (Green), liguriaB04.jp2 (Red), liguriaB08.jp2 (Near Infrared).
- *Value*: Digital numbers obtained from bottom of atmosphere reflectance (BOA).

Sentinel-2 L2A images of an area of the Lazio region in Italy:

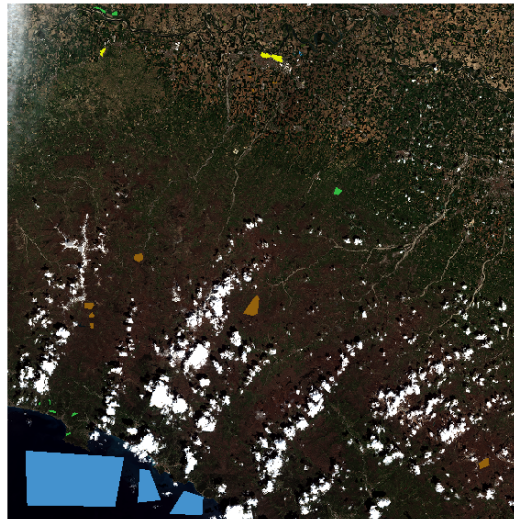
- *Resolution*: 10m
- *Size*: 10980x10980 pixels
- *Bands*: lazioB02.jp2 (Blue), lazioB03.jp2 (Green), lazioB04.jp2 (Red) , lazioB08.jp2 (Near Infrared).
- *Value*: Digital numbers obtained from bottom of atmosphere reflectance (BOA).

B.3 Case 1: Classification with Creation of Training Model

Firstly, we are going to create the training dataset. In order to do this, we create a vector layer in which we draw polygons. In each polygon we assign an integer value for the attribute "class" going from 1 to 7; these correspond to a land cover class according to this legend:

1. Water (Blue)
2. Vegetated (Green)
3. Not vegetated (Brown)
4. Urban (Yellow)
5. Medium-probability clouds (Grey)
6. High-probability clouds (White)
7. Shadow (Black)

Afterwards, we rasterize this vector layer, setting null ROI values equal to -1, and saving it as *trainingdata.tiff*.



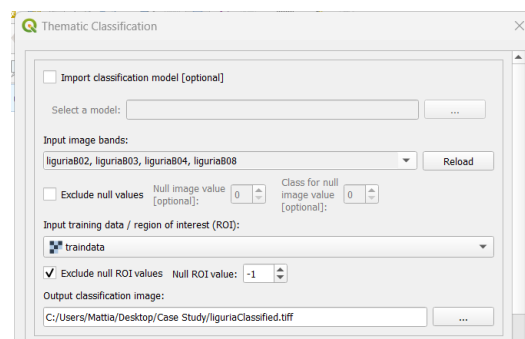
Now we have all the elements necessary to perform the classification, so we start the Thematic Classification plugin and select the following layers as input image bands:

- liguriaB02.jp2

- liguriaB03.jp2
- liguriaB04.jp2
- liguriaB08.jp2

As input training data / region of interest (ROI) we select *trainingdata.tiff*, then we check "exclude null ROI value" and we set "null ROI value" equal to -1.

We select a path to save the classified image and name it *liguriaClassified.tiff*.

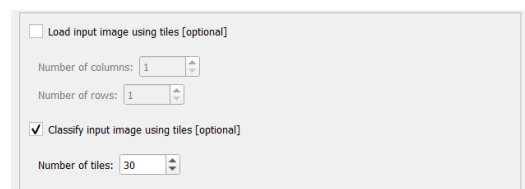


We choose to load the whole image in memory, because its size and number of bands are small enough to allow it; thus, we leave unchecked "load input images in tiles".

Instead, we check "classify image in tiles" and we set the relevant value equal to 30.

The combination of these two choices causes the bands to be loaded as a whole into memory but to be classified into 30 separate tiles, which speeds up the process quite a bit.

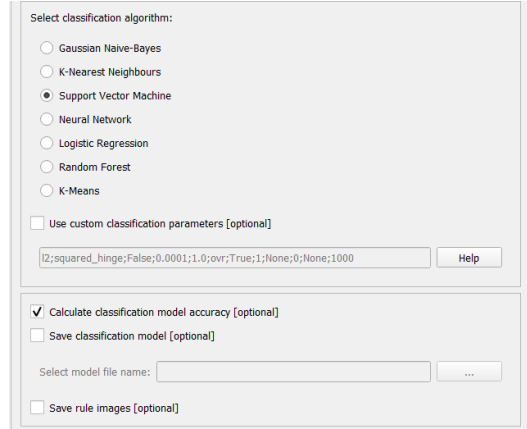
However, this option is not always feasible; indeed, if we were to use images with a very large size or a high number of bands, we could run into a memory error.



We select **Support Vector Machine** as the algorithm and leave the default

parameters. As mentioned earlier in the main document, we suggest that the more inexperienced users leave the default parameters.

Finally we check "calculate classification model accuracy", "save classification model" as SVMmodel.joblib, and click on **Run**.



The screenshot shows a dialog box titled "Select classification algorithm:". It contains a list of radio buttons for different algorithms: Gaussian Naive-Bayes, K-Nearest Neighbours, Support Vector Machine (which is selected), Neural Network, Logistic Regression, Random Forest, and K-Means. Below this list is a checkbox for "Use custom classification parameters [optional]", which is currently unchecked. If checked, it would reveal a text input field containing a complex parameter string and a "Help" button. Below the custom parameters section, there are three checkboxes: "Calculate classification model accuracy [optional]" (checked), "Save classification model [optional]" (unchecked), and "Save rule images [optional]" (unchecked). The "Save classification model" checkbox is followed by a text input field for "Select model file name:" and a button with three dots "..." for file selection.

B.3.1 Results

Once the classification is completed we can discuss the results obtained.

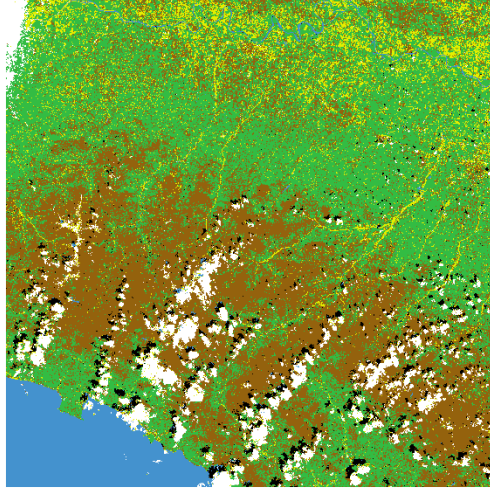
The first element analyzed is the accuracy. From the log we can read a value of 98.2% - this can be considered as a *test error*. Indeed, this value is evaluated during the training phase where, following the typical machine learning procedure, the training dataset is divided into two parts according to these percentages:

- **30%** of dataset is used as testing dataset
- **70%** of dataset is used as training dataset

After this, the model is fit on the training dataset and subsequently used to predict the classes of the test dataset, of which it's known the true value of the classes.

By comparing the predicted values with the true ones we can evaluate the accuracy. This is described as the ratio between the number of correctly classified samples and number of total samples, multiplied by 100. Having obtained a value of **98.2%** we can confidently state that our model has an excellent ability

to classify data previously unseen; therefore we consider it as reliable for the classification of the whole image.



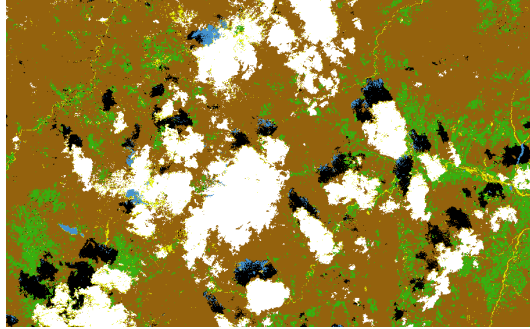
If we look instead at the classified image first, we can see that the areas where vegetation is present are classified correctly in most cases. This is due to the presence of band 8, which corresponds to the *near infrared spectrum*.

Healthy vegetation in fact has a **high albedo** in the near infrared spectral region, low albedo in the green visible region, and even lower in the blue and red ones. Therefore the spectral signature of the vegetation is well represented by the four chosen bands and it is easily distinguishable from the signature of the other classes. This, in turn, makes the model more precise in classifying the vegetation.

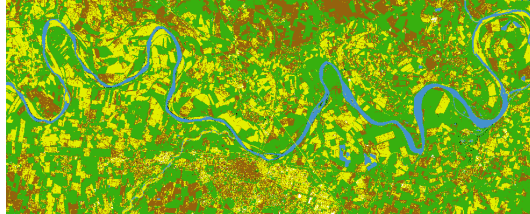
For the same reason the clouds too are classified in a very precise way. In this case, the reflectance is very high for all the bands; however, the difference between the spectral signatures of the medium probability clouds and high probability classes is very low and consequently the classifier easily recognizes clouds with respect to the other non-cloud classes. The same does not happen with the distinction between high probability clouds from medium probability clouds.

A similar reasoning applies to the water and shadows classes, where both have low reflectances in all four bands - in the case of water the minimum reflectance is in the blue band. This makes them easily recognizable by the classifier but can mistake shadow for water.

It is also important to note that the model has very precisely classified the river in the upper part of the image even though there were no samples in the training



dataset taken from that region.



Finally, for what concerns the non-vegetated and urban classes, they are also well recognized, if compared to other classes, but can get easily mistaken for each other. This is due to the high variety of spectral signatures within these classes, which makes the separation between them less clear.

B.3.2 Possible Areas of Improvement

To reduce these errors we could increase the number of bands, in order to take into account more points of the spectral signatures and subsequently making the classes more separable. However, this way we would be increasing the space taken up in memory, the duration of the process and we could incur in overfitting.

In the case of urban and non-vegetated classes we could instead increase the number of training samples and differentiate among them as much as possible, which could in turn slow down the training phase but increase the precision in the classification of elements belonging to these classes.

B.4 Case 2: Classification with Pre-Trained Model

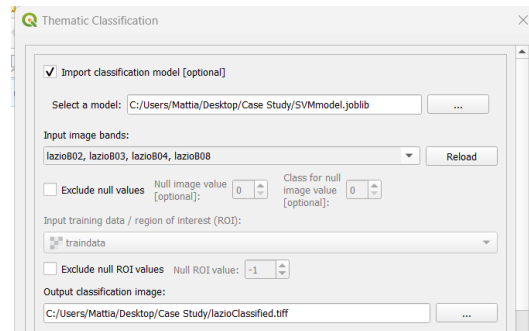
In this case we will use the pre-trained model, which was created in Case 1, to classify an image taken from the Lazio region.

Let's start by checking "*import classification model*" and selecting the path to the file named *SVMmodel.joblib*.

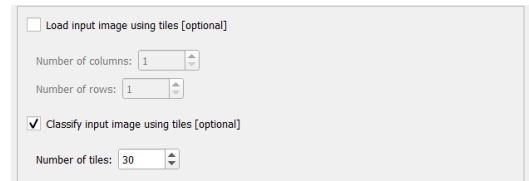
As input image bands we select the following images:

- lazioB02.jp2
- lazioB03.jp2
- lazioB04.jp2
- lazioB08.jp2

We select a path to save the classified image and name it *ClassifiedImage2.tiff*.



As for the previous case, we do not check "*load input images in tiles*", but we check "*classify image in tiles*" and set value equal to **30**.



Afterwards, we only have to click on run and await the results.



(a) Satellite Image



(b) Classified Image

B.4.1 Results

In this situation we do not have data whose class are known a priori, therefore we cannot evaluate the accuracy. To analyze the behavior of the classifier therefore we can only rely on a visual comparison with the true color image.

In general the same issues observed in Case 1 arise: shadows are classified as water in some cases, urban and non-vegetated are often misclassified as one another, etc. Overall we can be satisfied with the classification.

This is a very important result that increases our confidence in the model because it has correctly classified a completely new and unseen image overall with the big advantage of not having to go through a training phase again. This returns results much faster than the first case.

It is equally important to note, however, that this result is due to the strong similarity in terms of land cover of the two images, which are two distinct but geographically close areas that have the similar biomes and characteristics.

B.4.2 Conclusions

In conclusion, after having analyzed the results in both cases, we can be satisfied with our work and can consider the model reliable in classifying images of coastal areas of Italy; however, we must be careful when we use it on areas with vastly different terrains and characteristics, such as the more mountainous areas of the Alps. In this case, there may be areas of snow and ice that our model has not been trained to recognize and we therefore expect that these will be most

probably misclassified.

C Programming Guide

We have decided to include in this project a guide that explains the code implementation and its functionalities. This was done in order to increase its expandability, in case anybody would desire to enhance its functionalities.

The document, as well as the source jupyter notebook, can be found in the same repository as this guide.