Pixel-Wise Classification of Multispectral Satellite Images Using Machine Learning

Author: LIANG ZHONGYOU

Student ID [redacted]
Advisor: Ludovico Giorgio Aldo Biagi
Co-advisor: Mariagrazia Fugini
Academic Year: 2024-25

Contents

Introduction

This project, titled "Pixel-Wise Classification of Multispectral Satellite Images Using Machine Learning", was conducted by Liang Zhongyou from April to September 2024. It centers on the development of a cloud-based application for performing pixel-level classification of multispectral satellite images through machine learning methods. The tool offers both supervised and unsupervised classification options, providing flexible analysis solutions for various applications.

For supervised classification, the application uses k-nearest neighbors (KNN) algorithm with Euclidean distance, allowing users to upload images in GeoTIFF format, select training samples, and classify the image based on spectral signatures of different land cover types. For unsupervised classification, the algorithm uses K-means, allowing users to define the number of classes to do classification.

The primary objective of the project is to streamline the process of image classification, specifically tailored for geospatial engineers and researchers working with multispectral data. The tool can classify images into different classes based on selected sample polygons, with the ability to generate results in a user-friendly graphical interface.

1 Related work

In the field of remote sensing, pixel-based classification is a commonly employed technique to extract meaningful information from satellite imagery. Various tools and methods have been developed, such as:

- **Google Earth Engine (GEE)**: Provides powerful cloud-based geospatial analysis but requires users to write code.
- **QGIS**: Offers robust GIS tools but can be complex for users who require only image classification.

Table 1.1: Comparative view

Feature	Google Earth Engine (GEE)	QGIS	This Application
Usability	Requires coding	User-friendly GUI, some features require plugins	Very simple and intuitive
Customization	Requires extensive scripting for custom workflows	Limited to built-in functions and plugins	No scripting needed
Cost	Free with quota limitations	Free and open-source	Completely free
Deployment and Accessibility	Cloud-based, accessible from anywhere	Desktop application, must be installed locally	Cloud-based, accessible from anywhere

Google Earth Engine and QGIS, though powerful, may be too complex for users with limited technical skills. Our project aims to bridge this gap by creating a simpler, free, cloud-based alternative.

2 Innovation of the project

The innovation of this project lies in its simplicity, flexibility, and accessibility. Unlike existing tools that often require substantial geospatial expertise or programming knowledge, this web application empowers users with minimal technical experience to classify multispectral images. It's an open-source solution without complicated deployment, providing support for both supervised and unsupervised classification, a feature typically found in paid software.

3 Project development

3.1. Application Architecture

The application follows a client-server architecture. The user interface (UI) is built in HTML, CSS, and JavaScript, while the backend is powered by Flask (Python). The system allows users to upload GeoTIFF images, classify pixels, and download the classified results.

The application's backend uses flask, a lightweight web framework for Python, designed to be simple yet highly flexible. It provides the essentials for building web applications, allowing developers to structure their projects exactly how they want without enforcing any particular pattern.

For the frontend, the application is presented to users in the form of a web page, with HTML used to construct the framework of the web page, CSS used to define the style of it and JavaScript used for interactivity with users.

The application is deployed on Microsoft Azure, which provides free servers with good performance. Considering the balance between cost and performance, it is a good option.

3.2. Used Tools

3.2.1. Library and Module

- **Numpy**: For numerical operations on image data.
- **Sklearn**: Provide KNeighborsClassifier, train_test_split, confusion_matrix, classification_report and K-means.
- **KneighborsClassifier**: For k-nearest neighbors (KNN) algorithm implementation.
- **Train_Test_Split**: To split the collected data into training and testing sets.
- **Confusion_Matrix**: To compare the predicted labels against the actual labels. It shows how many pixels were correctly classified and how many were misclassified.
- **Classification_Report**: It provides key performance metrics for each class,such as precision, recall and f1-score.
- **Rasterio**: To read and process GeoTIFF images.
- **Base64**: To convert the image to base64 image data, eliminates the need to save the file and reduces the storage overhead of the server.
- **Shapely**: Used to generate sample points and polygons in frontend.
- **Matplotlib**: Used to generate visual outputs of classified images.
- **K-Means Clustering**: For unsupervised classification.
- **Leaflet**: To create interactive layers in the frontend.

3.2.2. Platforms

- **Microsoft Azure**: Azure App Service for hosting the application.

3.3. Supervised Classification

Users upload a GeoTIFF image and manually select areas of interest (AOIs) by drawing polygons on the image. The selected polygons act as training samples for pixel classification, which is carried out using k-nearest neighbors (KNN) algorithm with Euclidean distance.

The following are the main steps and logic of supervised classification.

3.3.1. Data Preparation

•**Input Data**: The user provides labeled sample regions, which are polygons drawn over the image. These polygons define the areas that belong to specific classes. Each polygon has an associated label.

•**Pixel Extraction**: The application extracts pixel values (spectral values for each band in the image) that fall within the provided polygons. Each pixel is represented as a feature vector that includes its intensity values across the various spectral bands.

• **Labeling**: For each extracted pixel, a label corresponding to the polygon is assigned. These labels represent the class the pixel belongs to.

This data collection results in two arrays:

•**X**: An array of feature vectors (pixel values from the sample regions).

•**y**: The corresponding labels (the class each pixel belongs to).

3.3.2. Model Training

•**Training Data Split**: The collected data (X and y) is split into training and testing sets using the train_test_split method from sklearn. 80% of the data is used for training and 20% for testing, ensuring that the model is evaluated on unseen data.

•**KNN Classifier**: The classifier used is KNeighborsClassifier from sklearn, with the number of neighbors set to 3 (n_neighbors=3) and the distance metric set to Euclidean. For each pixel, the classifier looks for the 3 closest neighbors (based on Euclidean distance in the feature space) and predicts the class based on the majority vote of the neighbors. The classifier is trained using the training data (X_train, y_train), which contains the pixel values as features and their corresponding labels as targets.

3.3.3. Pixel Classification

•**Flattening the Image**: After training, the entire image is reshaped (flattened) into a 2D array of pixels, where each row represents a pixel and each column represents a feature (band value).

•**Masking Black Pixels**: Pixels that are black (value [0, 0, 0]) are excluded from the classification process because they are considered uninformative.

•**Classifying Valid Pixels**: The valid (non-black) pixels are classified using the trained KNN model. Each pixel is assigned a class based on the majority vote of its 3 nearest neighbors.

•**Reconstructing the Classified Image**: The classification results are reshaped back into the original image shape (height, width), with class labels assigned to each pixel.

3.3.4. Evaluation

•**Test Data Prediction**: The classifier is evaluated using the test data (X_test, y_test), and the predicted labels (y_pred) are compared to the actual labels.

•**Confusion Matrix**: The confusion matrix compares the predicted labels (y_pred) against the actual labels (y_test). It shows how many pixels were correctly classified and how many were misclassified. The matrix is normalized to account for class imbalances by dividing each row by the sum of its elements (which represents the total number of pixels in that class).

• **Classification Report**: The classification report is generated using sklearn.metrics.classification_report. It provides key performance metrics for each class, such as:

Precision: The percentage of correctly classified pixels out of all pixels predicted to belong to a given class.

Recall: The percentage of correctly classified pixels out of all pixels that actually belong to that class.

F1-Score: The harmonic mean of precision and recall.

3.3.5. Final Output

•**Classified Image**: The classified image is returned as a base64-encoded PNG. This image shows the pixel labels, where each pixel is colored according to its assigned class.

•**Confusion Matrix**: The normalized confusion matrix is visualized as an image and returned as a base64-encoded PNG. It provides a visual summary of the classifier's performance across different classes.

• **Classification Report**: A detailed classification report is returned as a text summary, including precision, recall, and F1-score for each class.

3.4. Unsupervised Classification

The unsupervised classification uses K-Means clustering to group similar pixels without prior user input. This method allows automatic classification based on inherent spectral patterns in the image.

The following are the main steps and logic of unsupervised classification.

3.4.1. Data Preparation

The input image is received as a base64-encoded image and decoded to a NumPy array. The image is split into its spectral bands (e.g., Red, Green, Blue, or any other spectral bands from a multispectral image). The bands are reshaped into a 2D array (pixels), where each pixel corresponds to a row in the array, and the columns represent the different spectral values.

3.4.2. Masking Non-Valid Pixels

The code checks for non-valid pixels, such as those that are entirely black (i.e., where all spectral bands have zero values). These black pixels are ignored in the clustering process because they don't contribute meaningful data for classification. A mask is created to filter out these invalid pixels, and the classification is applied only to the valid (non-black) pixels.

3.4.3. Perform K-Means clustering

K-Means is applied to the valid pixels (valid_pixels). The algorithm attempts to group the pixels into num_classes (the user-defined number of classes).

K-Means Algorithm works by:

1.Initializing num_classes centroids randomly.

2.Assigning each pixel to the nearest centroid based on Euclidean distance.

3.Recomputing the centroids based on the average of the pixels assigned to each cluster.

4.Iterating the above two steps until the centroids converge (i.e., the assignments no longer change significantly).

3.4.4. Reconstruct the classified image

•**Class Label Assignment**:After clustering, each pixel is assigned a category label. These labels are then used to classify the pixels into their respective clusters.

•**Reshape the Classified Pixels**: The classified pixel labels are reshaped back into the image's original dimensions, so the result is a classification image, where each pixel is labeled according to its assigned class.

3.4.5. Final Output

The classified image is returned as a base64-encoded PNG image for visualization, which represents the unsupervised classification result.

4 System design and documentation

4.1. Use Cases

4.1.1. Upload Images

- **Actors**: User
- **Steps**: User upload image through "Choose File " button, system receive image and show it in the window.
- **Outcome**:The user can check the uploaded image in the window.

4.1.2. Supervised Classify

- **Actors**: User
- **Steps**: User enter a class label, and then click "Save Label" button, system receive the label; then the user start to draw and edit polygons to select training samples, system receive the data of training samples; user click "Supervised Classify" button, system perform supervised classification and show the classified image in the window; user download the classified image.
- **Outcome**: User can download the supervised classified results.

4.1.3. Unsupervised Classify

- **Actors**: User
- **Steps**: User enter the number of classes, system received the number; user click "Unsupervised Classify" button, system perform unsupervised classification and show the classified image in the window; user download the classified image.
- **Outcome**: User can download the unsupervised classified results.

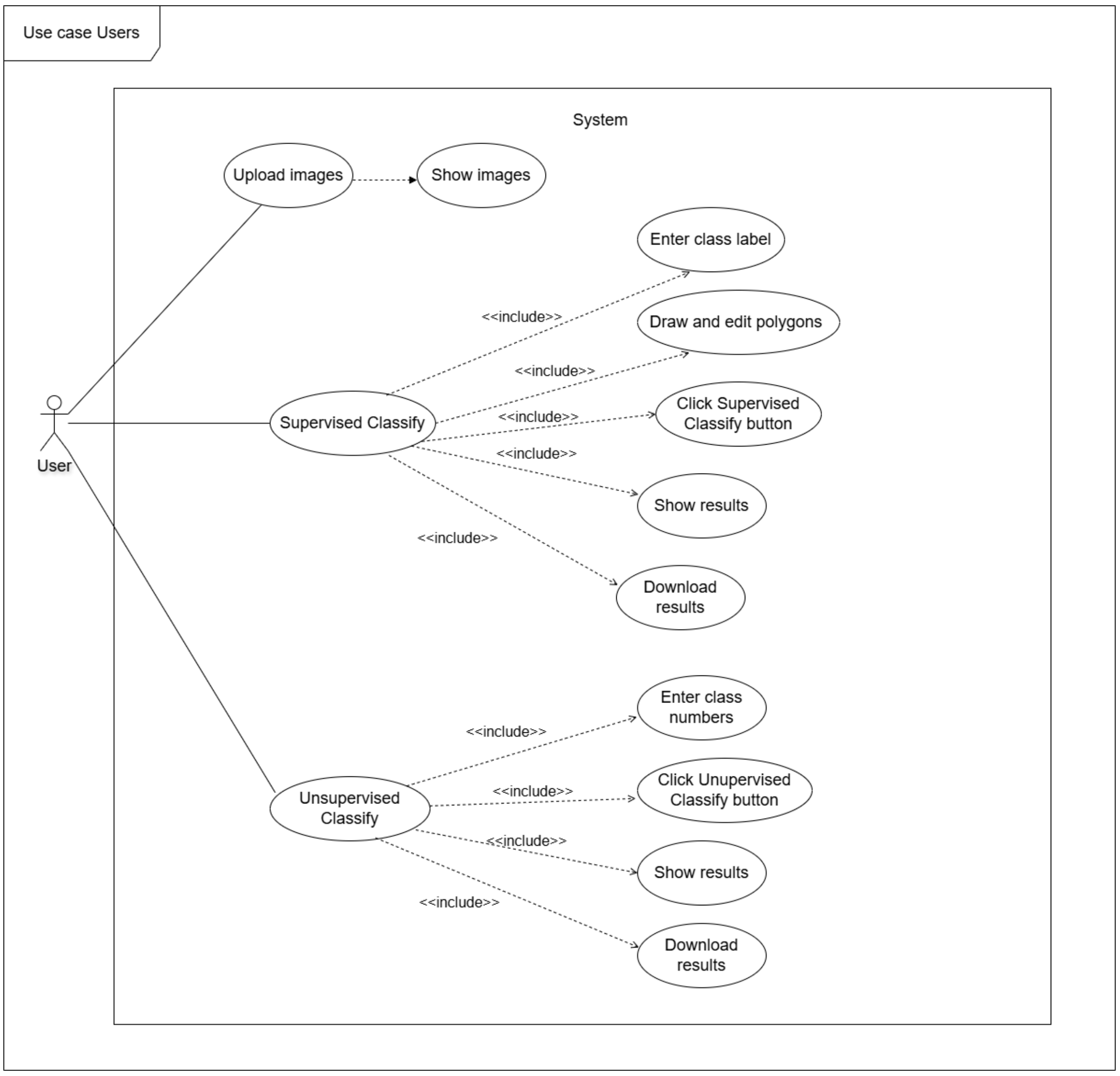


Figure 4: UML of all User cases structure

4.2. Data Schema

4.2.1. Data Structure

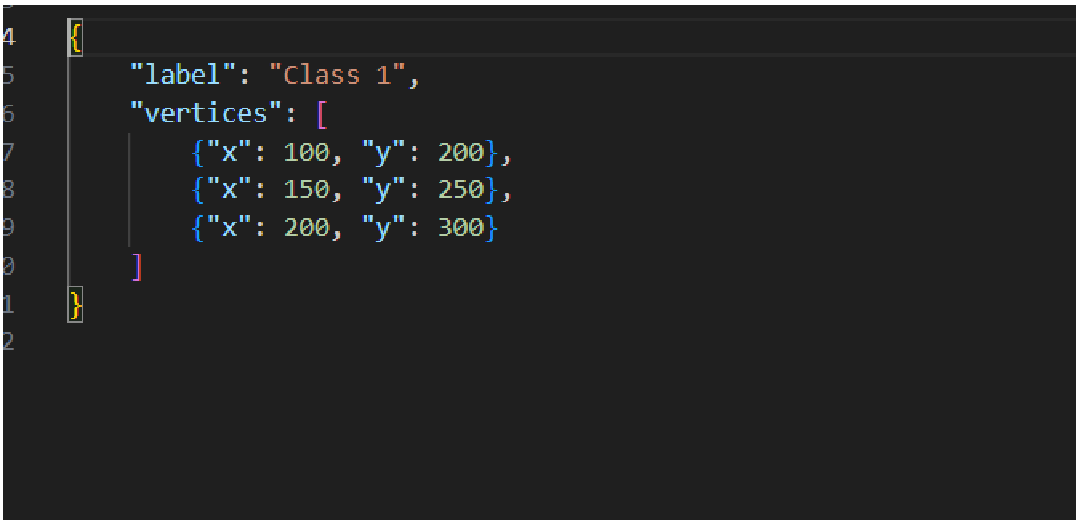
Input Data: Image uploaded by the user is multispectral GeoTIFF file. A GeoTIFF file is a standard format for georeferenced raster images. It is an extension of the TIFF (Tagged Image File Format) that includes spatial metadata, allowing the image data to be located within a specific geographic coordinate system. GeoTIFFs are widely used in geographic information systems (GIS) for mapping, remote sensing, and other geospatial applications. The structure of a GeoTIFF file consists of the following components:

- **Width**: The number of pixel columns in the image, that is, the number of pixels along the horizontal axis (X axis).
- **Height**: The number of pixel rows in the image, that is, the number of pixels along the vertical axis (Y axis).
- **Bands**: Different layers or channels to store image data, each band containing information about a specific spectral range.

The width and height determine the resolution of the image. The pixel data in each band corresponds to each pixel in the image. Each band is a two-dimensional array that matches the width and height of the image. Suppose we have a 1024x768 RGB GeoTIFF image, then its structure will be 1024 pixels in width, 768 pixels in height and 3 bands in red, green and blue.

The application can support from 8-bit to 64-bit GeoTiff file whose size less than 10MB to be uoloaded. But the image will be converted to 8-bit when it shows in the frontend.

Sample Data: User-drawn polygons on a map that are labeled with class information. These polygons represent regions of interest. Each polygon is converted into a collection of vertices, which are stored alongside their labels to define training data for supervised classification algorithms. The structure of Sample data is shown like figure 2.



```
{
    "label": "Class 1",
    "vertices": [
        {"x": 100, "y": 200},
        {"x": 150, "y": 250},
        {"x": 200, "y": 300}
    ]
}
```

Figure 2: The structure of Sample data

4.2.2. Data Storage

The application holds the data in memory while processing it. Image data is stored in base64 format, and labeled polygons and samples are stored as arrays of vertices. The image data is never saved to a database or file system.

4.2.3. Data Flow

The data flow is shown as below.

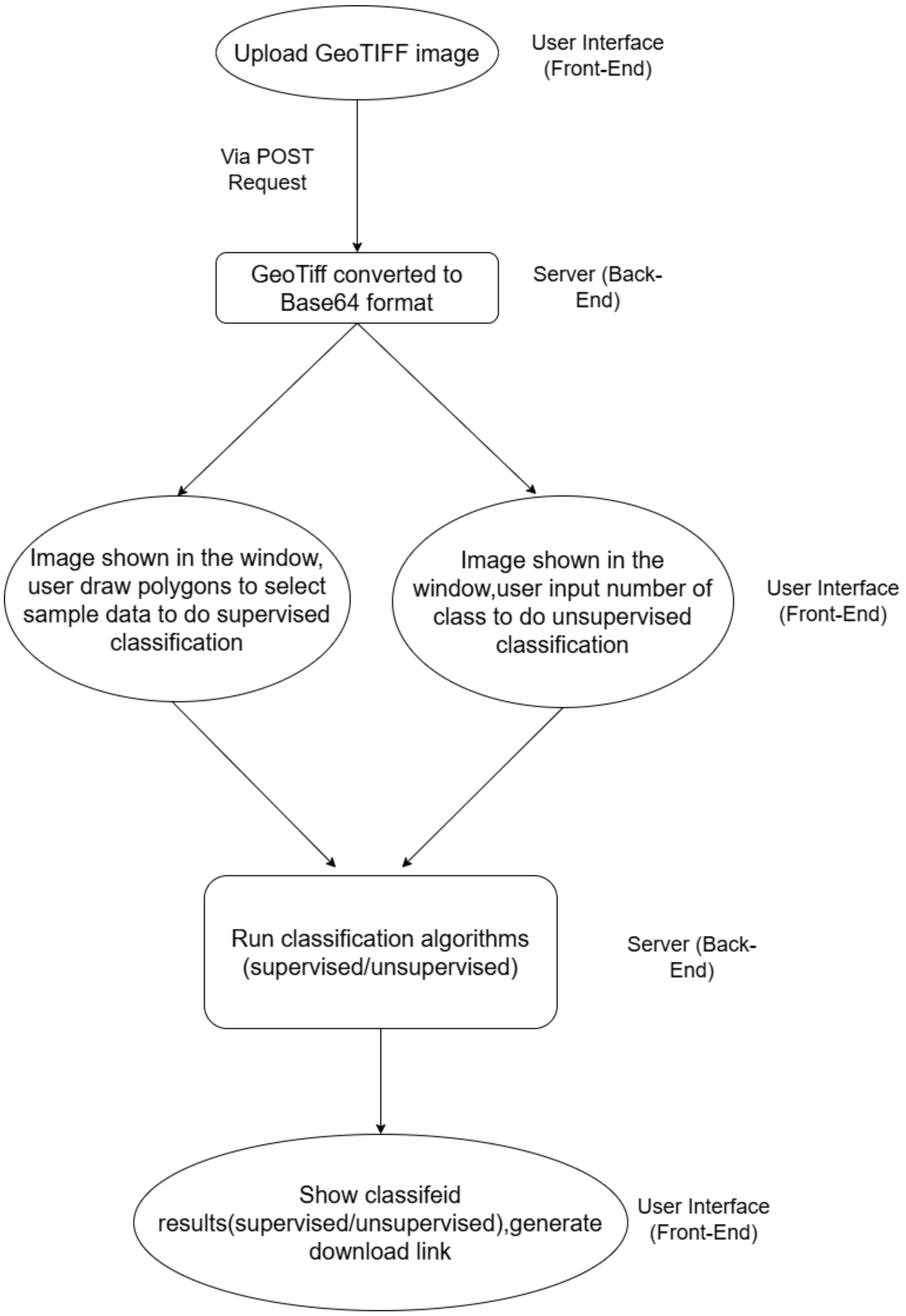

Upload GeoTIFF image
User Interface (Front-End)
Via POST Request
GeoTiff converted to Base64 format
Server (Back-End)
Image shown in the window, user draw polygons to select sample data to do supervised classification
Image shown in the window,user input number of class to do unsupervised classification
User Interface (Front-End)
Run classification algorithms (supervised/unsupervised)
Server (Back-End)
Show classifeid results(supervised/unsupervised),generate download link
User Interface (Front-End)

Figure 3: Block Diagram

4.3. UX Design

This application provides a clean, intuitive and easy-to-use user interface, aimed at offering users a clear operation path.

4.3.1. User Interface (UI) Design Overview

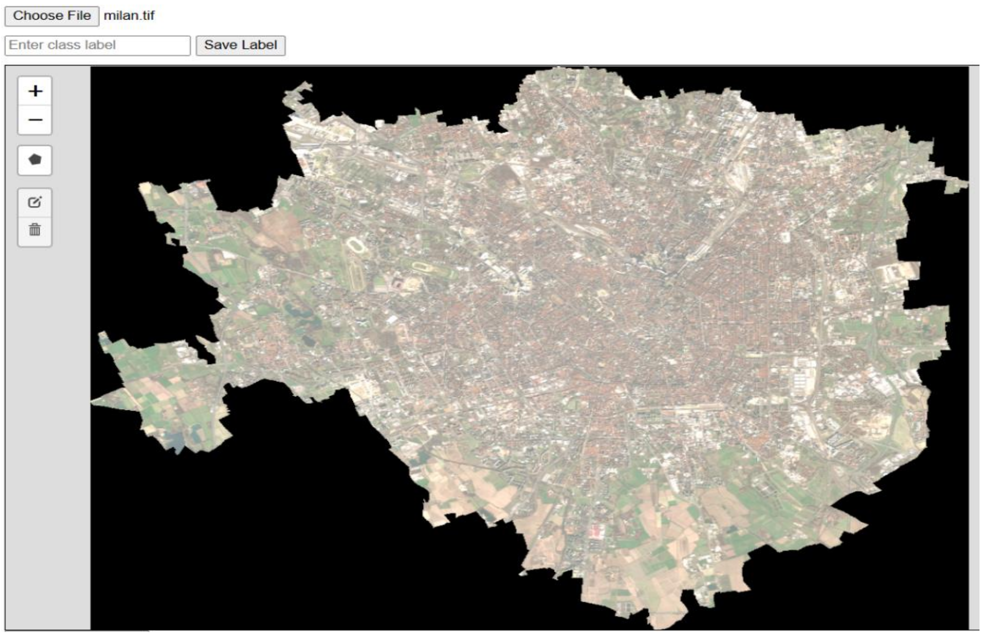


Figure 4: User Interface (After uploading an image)

After opening the application to upload an image, the main interface is shown in Figure 4. In the middle is the Uploaded Image Preview Window. In addition, the interface from top to bottom includes:

- **Choose File Button**: Users can click this button to select a satellite image file stored locally to upload.
- **Class Label Input Box**: Users can input the name of the class label here when doing supervised classification.
- **Save Label Button**: Users click the button to save the label.
- **Zoom In Button**: This helps users to zoom in the image.
- **Zoom Out Button**: This helps users to zoom out the image.
- **Draw Polygon Button**: After clicking this button, users can draw polygons in the image.
- **Edit Layer Button**: After clicking this button, users can modify the already drawn polygons, for example, changing the shape of them.

- **Delete Layer Button**: Users can delete any polygons that they want to abandon by clicking a polygon and then clicking this button.

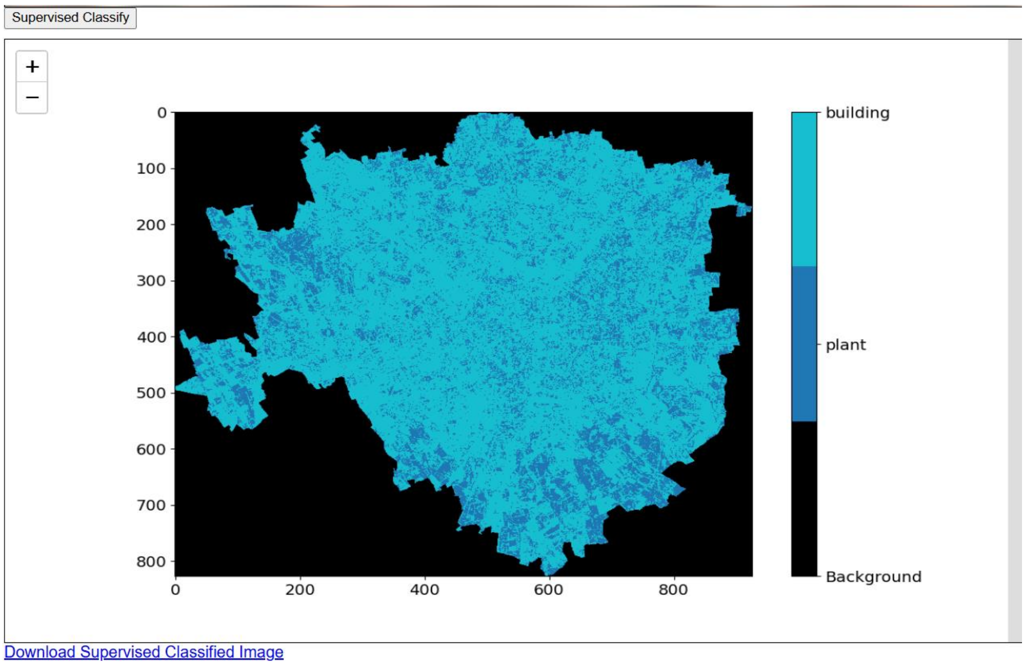
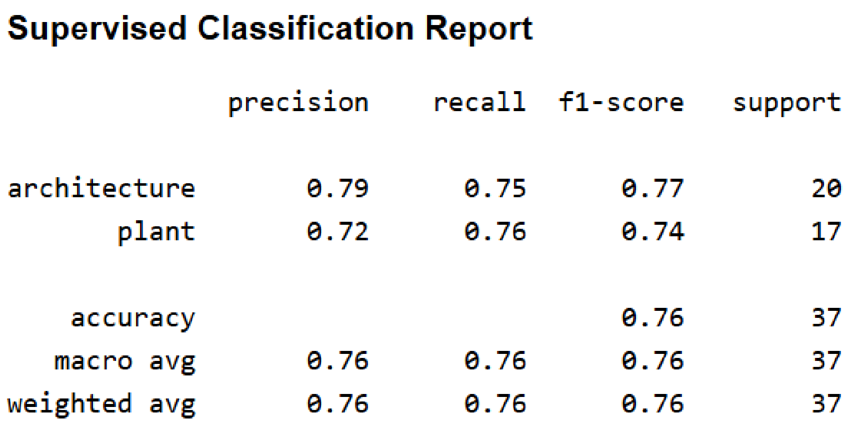


Figure 5: Supervised-classified image display window

Below the Uploaded Image Preview Window is the Supervised-classified Image Display Window which shows the supervised classification image. The interface is shown in Figure 5. From the top to the bottom, it includes:

- **Supervised Classify Button**: After clicking this button, the application start to do supervised classification.
- **Zoom In Button**: This helps users to zoom in the image.
- **Zoom Out Button**: This helps users to zoom out the image.
- **Download Supervised Classified Image Link**: Clicking the link to download the Supervised-classified Image.



Supervised Classification Report

```
              precision    recall  f1-score   support

architecture       0.79      0.75      0.77        20
       plant       0.72      0.76      0.74        17

    accuracy                           0.76        37
   macro avg       0.76      0.76      0.76        37
weighted avg       0.76      0.76      0.76        37
```

Figure 6: Supervised classification report display window

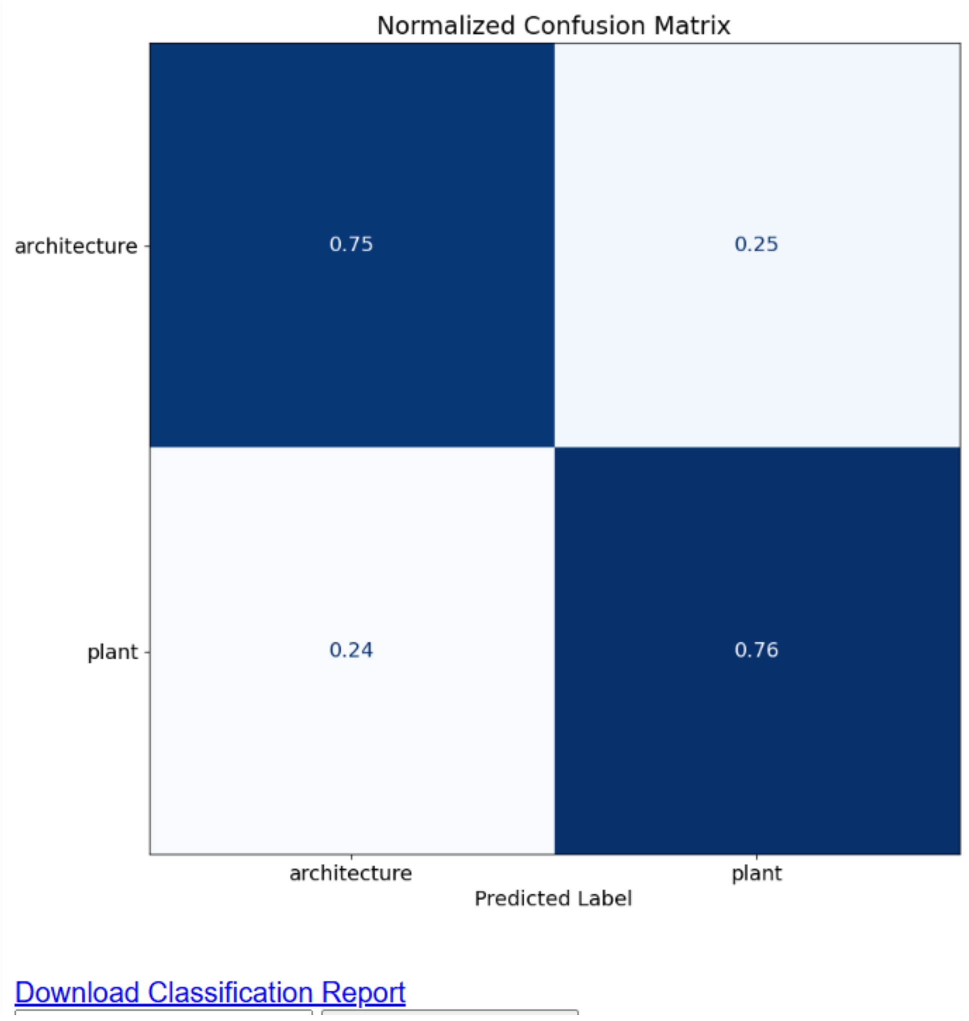


Figure 7: Normalized confusion matrix display window

Below the Supervised-classified Image Display Window are the Supervised classification report and Normalized confusion matrix display window which shows the supervised classification evaluation results. The interface is shown in Figure 6 and Figure 7. Besides, there is a Classification report download link for downloading the Supervised classification report.

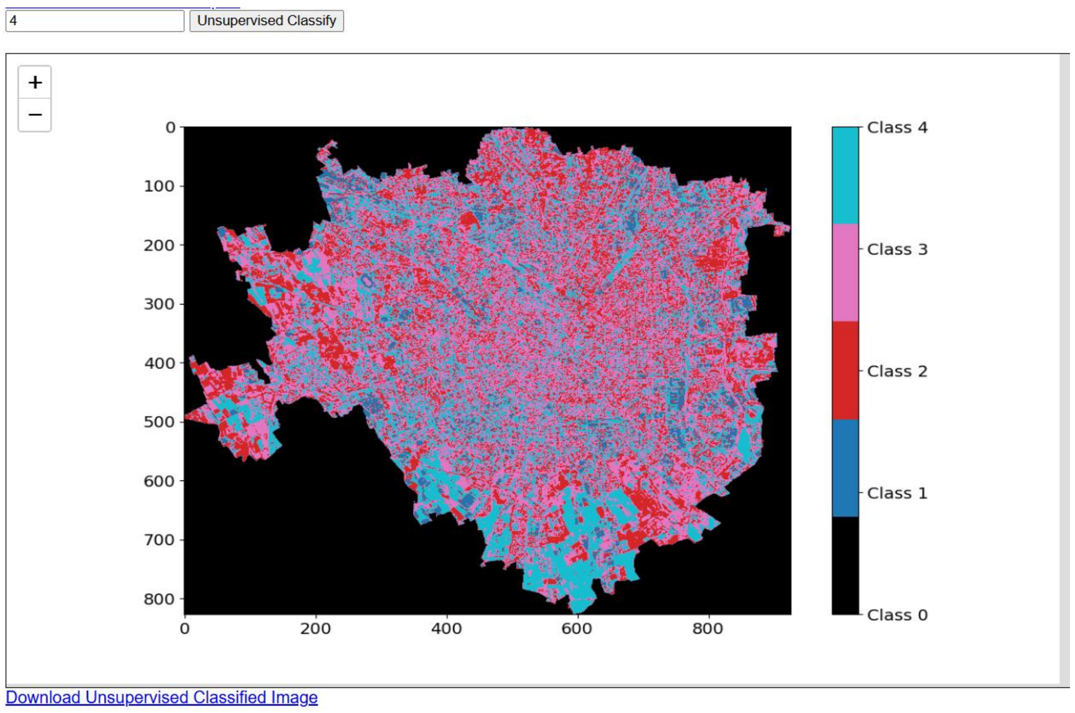


Figure 8: Unsupervised-classified image display window

In the bottom is the Unsupervised-classified Image Display Window which shows the unsupervised classification results. The interface is shown in Figure 8. From the top to the bottom, it includes:

- **Number Of Classes Input Box**: Users can input the number of classes here when doing unsupervised classification.
- **Unsupervised Classify Button**: After clicking this button, the application start to do unsupervised classification.
- **Zoom In Button**: This helps users to zoom in the image.
- **Zoom Out Button**: This helps users to zoom out the image.
- **Download Unsupervised Classified Image Link**: Clicking the link to download the Unsupervised-classified Image.

4.3.2. User Flow

Here is a typical user flow:

1. **Upload Image**: After opening the webpage, users can click the "Choose File" button to select a satellite image file stored locally to upload.
2. **Choose Processing Method**: Users can choose to do supervised classification or unsupervised classification, or both of them. For supervised classification: user need to provide training samples by drawing polygons; for unsupervised classification: users need to input the number of classes.
3. **Image Processing**: After users click “Supervised Classify” or “Unsupervised Classify” button, the application will start to process the image.
4. **View and Analyze Results**: After the image is processed, the classification results are displayed on the corresponding window, where users can check and download.

4.3.3. Usability

The application design prioritizes ease of use:

- **Simple and Intuitive Interface**: The application uses a single-page design, where all operations are completed on one screen, avoiding the confusion caused by frequent page switching.
- **Tooltips**: Main operation buttons and functional areas have tooltips(shown in Figure 9). When users hover over an element, a brief description appears to help them quickly understand the function.
- **Friendly interaction**: As shown in Figure 10, the application will give users appropriate feedback at key steps, allowing users to clearly know the progress of the program.

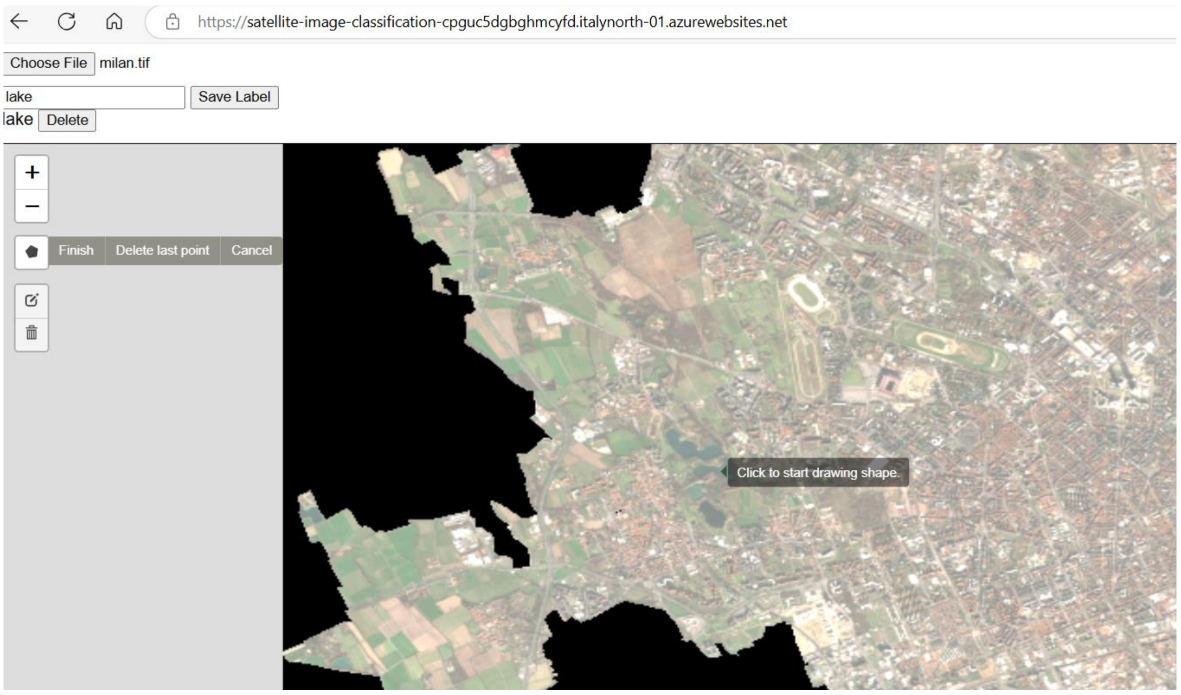
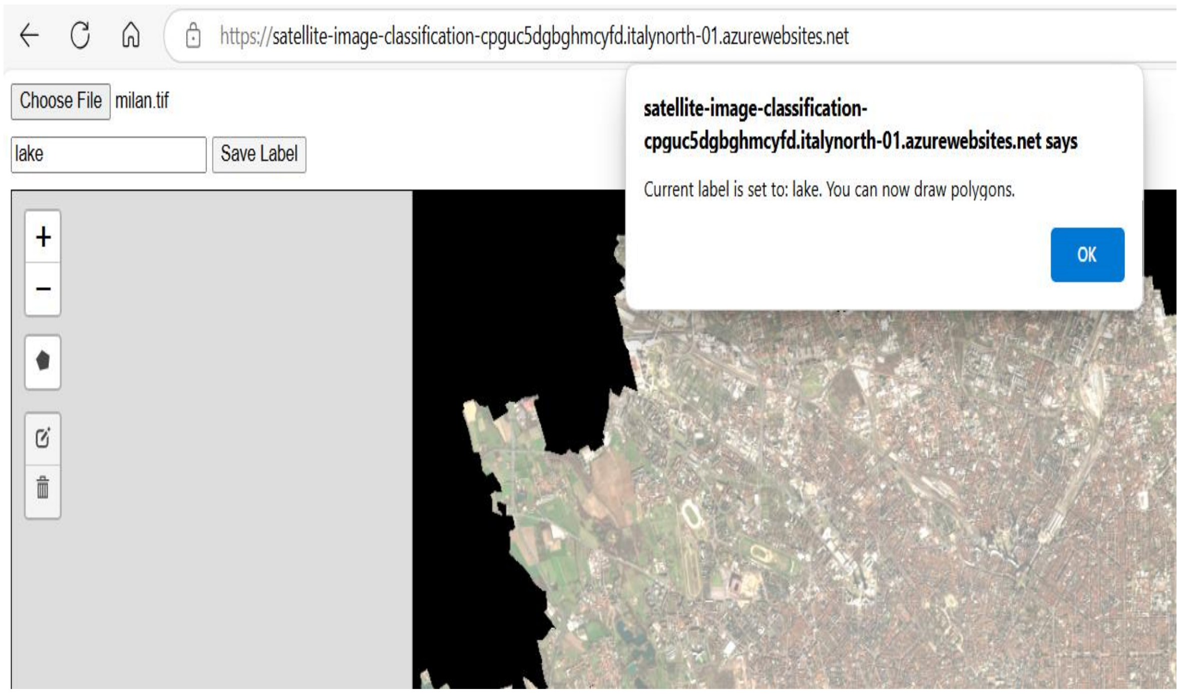

https://satellite-image-classification-cpguc5dgbghmcyfd.italynorth-01.azurewebsites.net
Choose File
milan.tif
lake
Save Label
lake
Delete
+
−
Finish
Delete last point
Cancel
Click to start drawing shape.

Figure 9: An example showing tooltips


https://satellite-image-classification-cpguc5dgbghmcyfd.italynorth-01.azurewebsites.net
Choose File
milan.tif
lake
Save Label
+
−
OK

satellite-image-classification-cpguc5dgbghmcyfd.italynorth-01.azurewebsites.net says

Current label is set to: lake. You can now draw polygons.

Figure 10: An example showing friendly interaction

5 Results of Milano Case Study

5.1. Process

I uploaded a 32-bit RGB GeoTIFF image of Milano (shown in Figure 11), about 5 MB, for supervised and unsupervised classification.

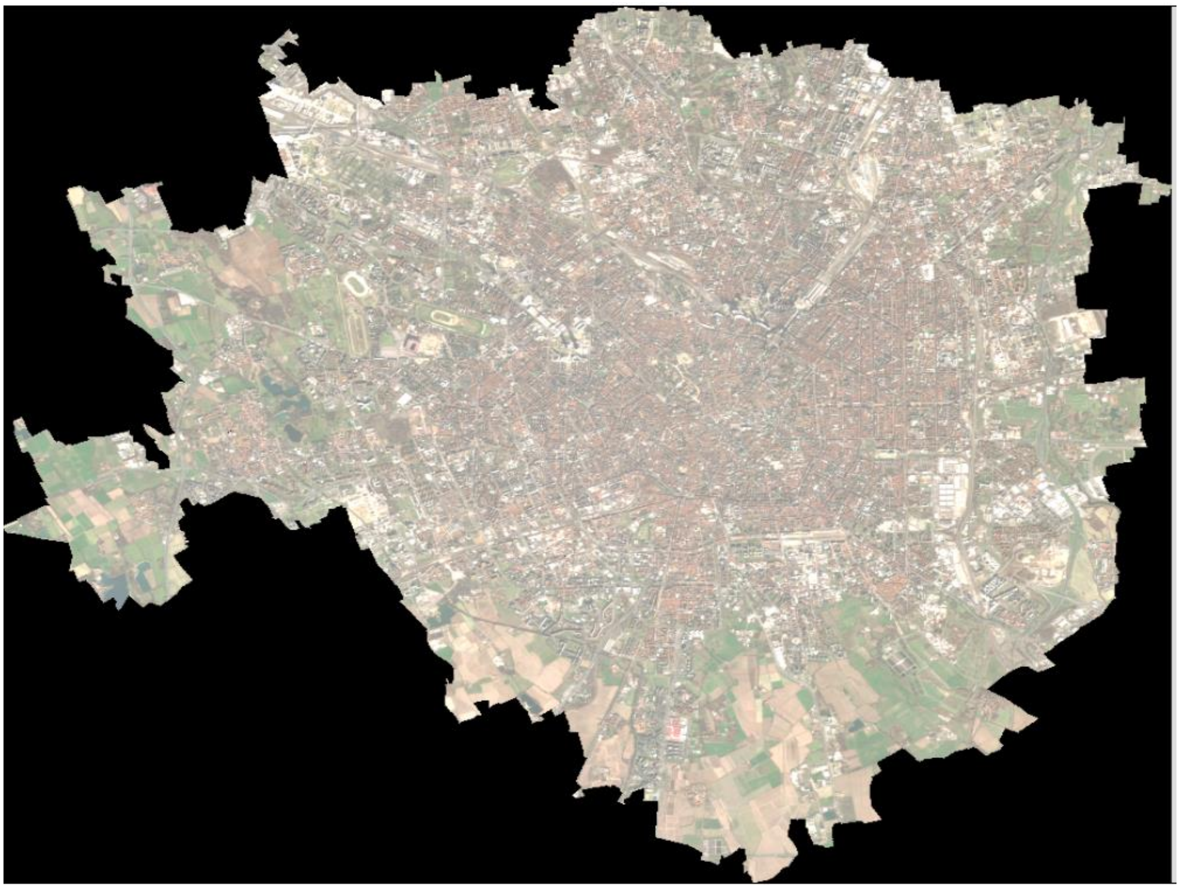


Figure 11: The uploaded image

Below were the steps that I did to get the results through the application.

Supervised Classification:

1. Click "choose file" to upload RGB GeoTIFF image of Milano.
2. Input class labels. Firstly, I input "architecture".
3. Click "draw polygon button", then draw a polygon, click the first point in the end to close the polygon drawing.

4. Repeat 3 to finish all the polygons drawing of "architecture", then enter a new class label to do next one. I did the same thing for "plant" labels. The training samples are shown in Figure 12.
5. After finishing sample data selecting, click the "Supervised Classify" button.
6. View and download the result.

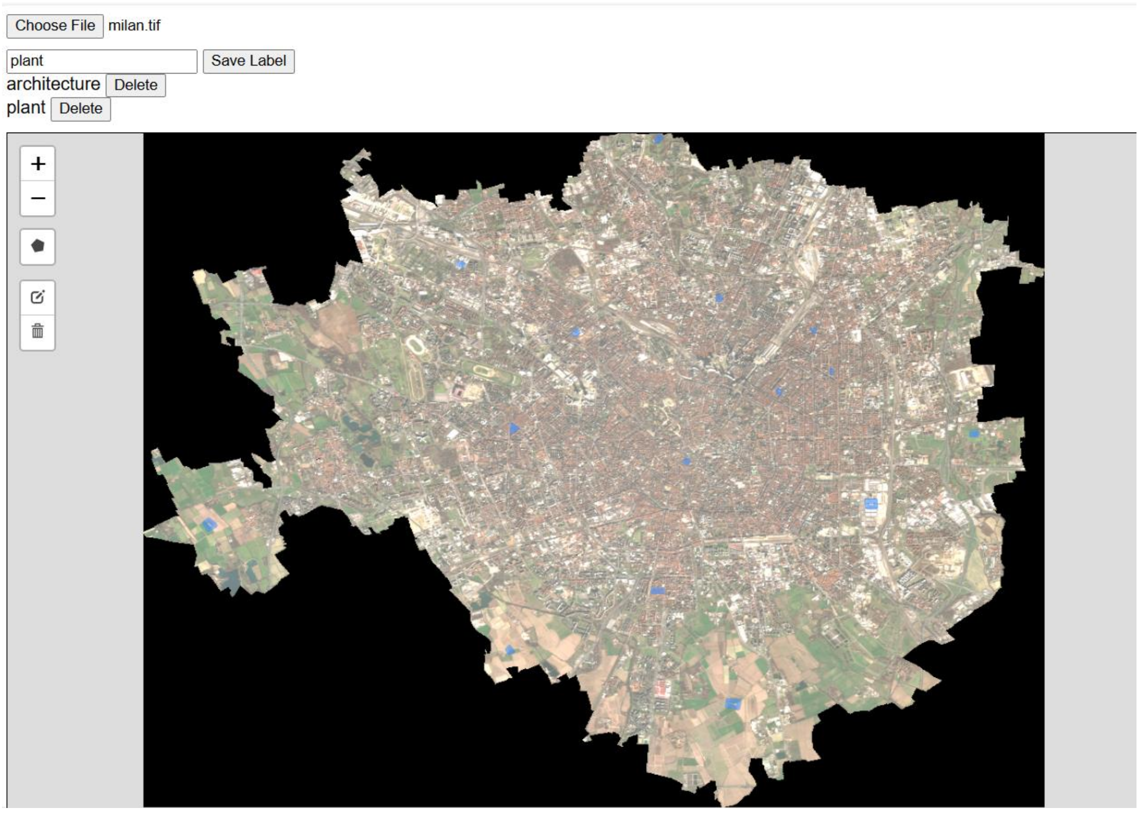


Figure 12: Class labels and training samples of case study

Unsupervised Classification:

1. I Input 6 as the number of classes.
2. Click the "Unsupervised Classify" button.
3. View and download the result.

5.2. Results

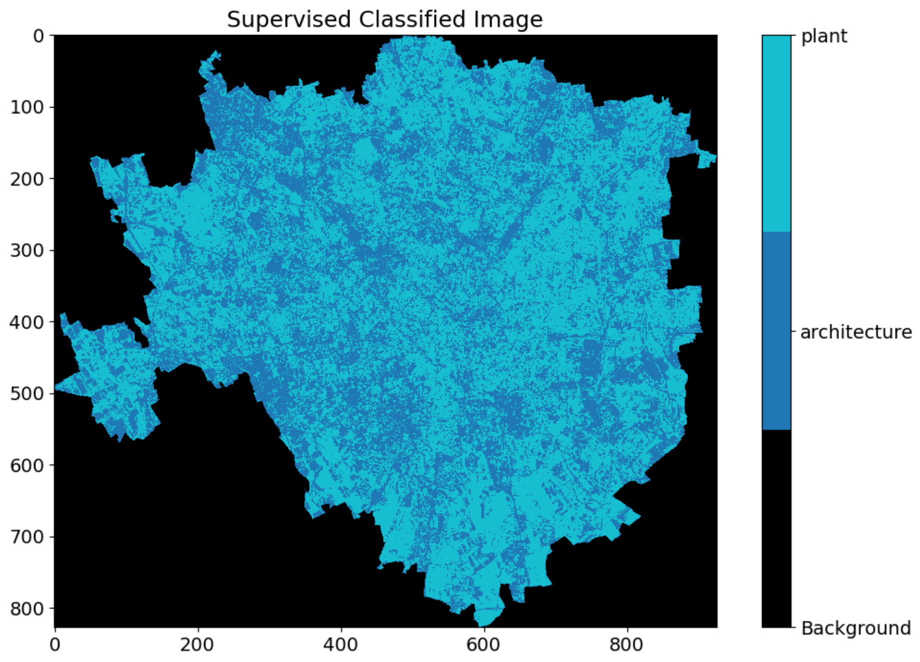
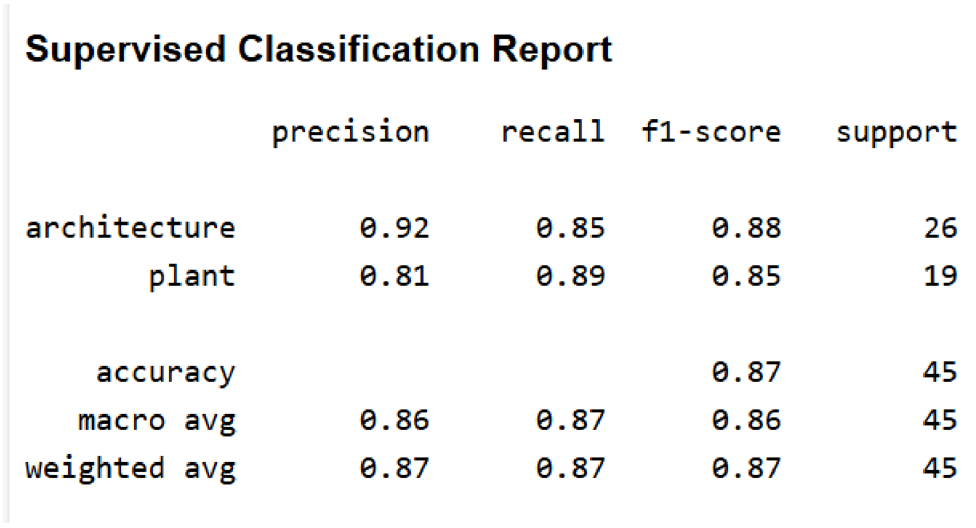


Figure 13: Supervised Classification Image



Supervised Classification Report

```
              precision    recall  f1-score   support

architecture       0.92      0.85      0.88        26
       plant       0.81      0.89      0.85        19

    accuracy                           0.87        45
   macro avg       0.86      0.87      0.86        45
weighted avg       0.87      0.87      0.87        45
```

Figure 14: Supervised Classification Report

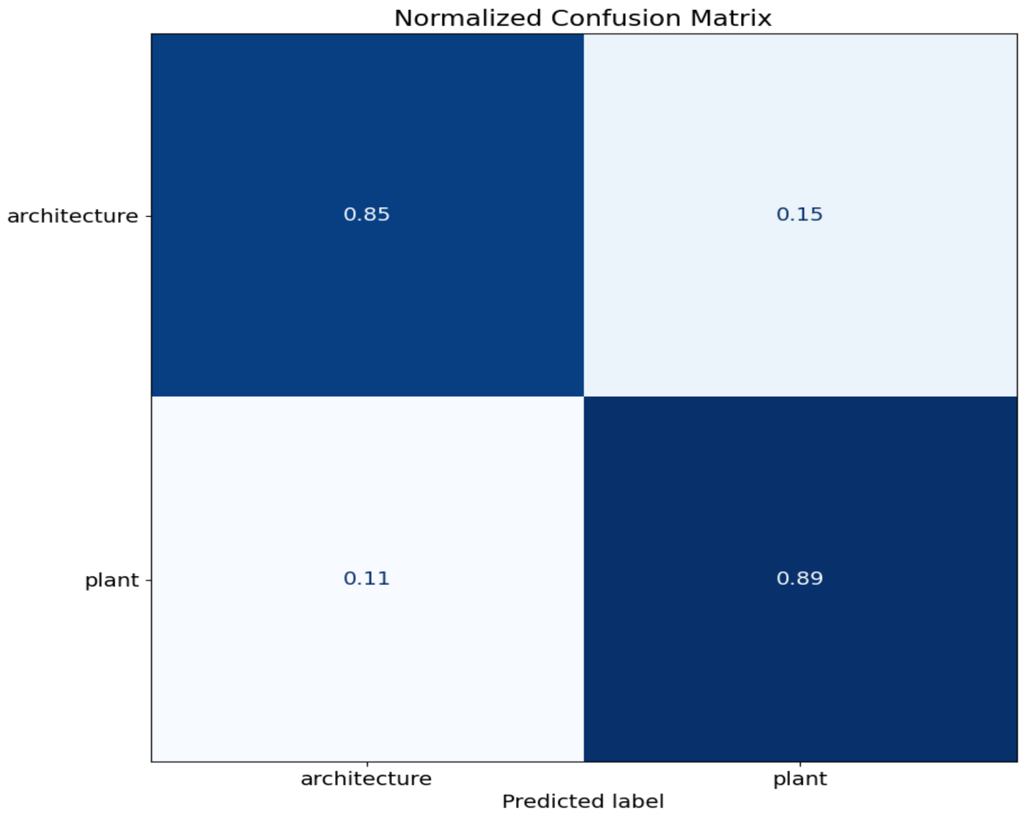


Figure 15: Normalized Confusion Matrix

The model shows high precision for "architecture" (0.92) and strong recall for "plant" (0.89), though some misclassifications occur, particularly with "architecture" being misclassified as "plant" (15%). While the metrics indicate effective performance, the model struggles slightly with feature overlap, leading to lower precision for "plant" (0.81). But, the supervised classification results demonstrate good overall performance with an accuracy of 87% and balanced metrics across both classes.

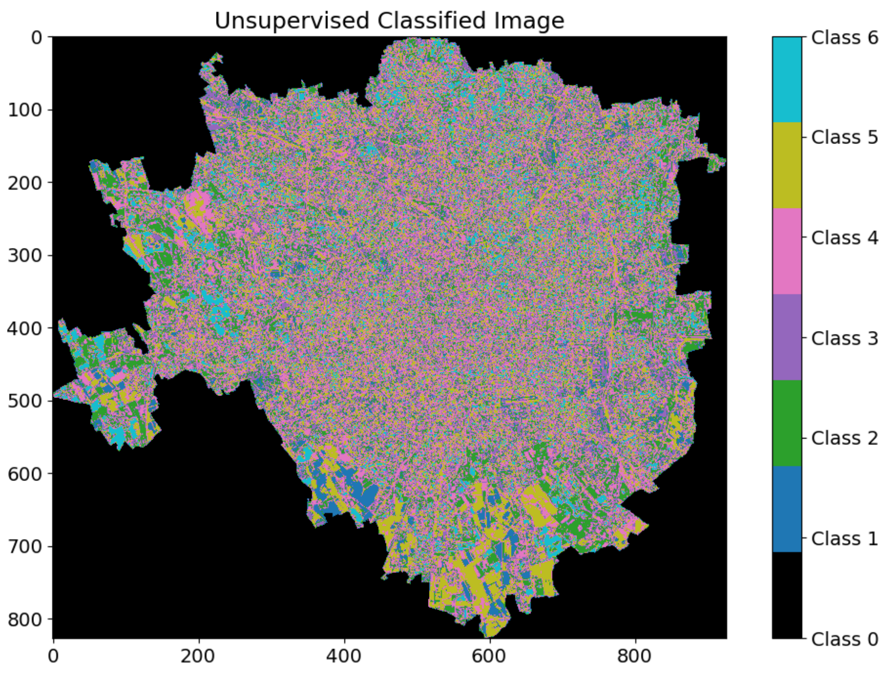


Figure 16: Unsupervised Classification Image

The Unsupervised classification results are displayed with color maps for easy visual distinction between classes.

6 Conclusion

This application simplifies the process of pixel classification in multispectral GeoTIFF images. It offers both supervised and unsupervised classification options with an easy-to-use interface. The project demonstrates the feasibility of providing complex image classification capabilities to a wider audience without requiring expensive software or technical expertise.