

Drone Image Analysis

Image Classification Process in
QGIS



Interreg



Co-funded by
the European Union

Aurora



SeaMoreEco

Interreg



**Co-funded by
the European Union**

Aurora



SeaMoreEco



County Administrative
Board of Norrbotten



**Länsstyrelsen
Västerbotten**
County administrative board of Västerbotten



Centre for Economic Development,
Transport and the Environment



**Swedish Agency
for Marine and
Water Management**



**REGIONAL COUNCIL
OF LAPLAND**



SGU Geological Survey
of Sweden

Titel: Drone image analysis – image classification process in QGIS
Authors: Julian Horstmann, Carlos Paz von Friesen
Year of publication: 2025
Cite as: Horstmann, J. & Paz von Friesen C. 2025. Drone image analysis – image classification process in QGIS

Preface

The shallow coastal areas of the northern Gulf of Bothnia represent ecologically important areas, distinguished by their rich biodiversity and unique habitats. However, these areas are increasingly vulnerable to the pressures of human activity and the accelerating impacts of climate change. A challenge for management today is the lack of knowledge of occurring species and habitats and practical solutions on how to improve their conservation status.

SeaMoreEco is a Finnish - Swedish initiative that during the years of 2023 - 2025 brought together expertise in marine biology and geology to map coastal habitats as well as develop, test, and demonstrate methods for monitoring, conservation, and restoration in shallow marine environments in the northern Gulf of Bothnia.

This manual is published as one output of the project, with the aim to share the results of the conservation efforts done in the project through monitoring, restoration and population studies of threatened species and habitats in the shallow coastal areas. Through these efforts, the project aims to strengthen the knowledge of marine habitat and their management and foster long-term resilience in the northern Gulf of Bothnia's shallow ecosystems.

The project was financed by Interreg Aurora and co-financed by Regional Council of Lapland and Swedish Agency for Marine and Water Management.

Participating organisations:

County Administrative Board of Norrbotten

County Administrative Board of Västerbotten

Geological Survey of Finland

Geological Survey of Sweden

Centre for Economic Development, Transport and the Environment in North Ostrobothnia

Centre for Economic Development, Transport and the Environment in South Ostrobothnia

Contents

PREFACE	3
CONTENTS	4
INTRODUCTION	5
SETUP	5
Input, Folder- and Naming-structure	5
Coordinate System.....	6
File Format	6
System Settings	7
MAIN WORKFLOW	7
Step 1 Clip out Water	7
Step 2: Segmentation.....	8
Step 3: Calculate Statistics.....	12
Step 4: Add Training- and Validationdata.....	13
Step 5: Join stats to segments, train the classifier and model the segments	14
Step 6: Model Refinement.....	15
Step 7: Validation and Output	16

Introduction

The goal of this manual is to describe a workflow for mapping and quantifying aquatic vegetation, using drone-based orthophotos.

The main interest is statistical information (type, coverage, distribution, patch sizes), rather than perfect cartographic polygons.

The manual is written for users who already have experience with GIS and are familiar with QGIS.

The analysis is performed on RGB orthophotos (Red, Green, Blue).

Setup

Input, Folder- and Naming-structure

A clear and consistent data structure is important to keep the entire workflow organised and reproducible. The analysis is usually performed by seascapes or clusters (e.g. **Kronören**, **Ostnäs**) that are divided into bays and smaller areas. We recommend creating one main folder for each seascape. Avoid using umlauts (ä, ö, ü) anywhere in the file path; a recommended naming convention is:

- **AC_Kronoren**
 - **AC** = abbreviation for the County Administrative Board of Västerbotten
 - **Kronoren** = name of the seascape

For international projects, it may be useful to begin with the country code, for example:

- **SW_AC_Kronoren**

We usually start with one or more large orthophotos, exported as GeoTIFFs (RGB). A suggested naming format is:

- **AC_KronorenLillfjarden_250811_HP**
 - **AC** = Västerbotten
 - **Kronoren** = Study Area
 - **Lillfjarden** = Subsection (If study area is split into subsections)
 - **250811** = Date that the orthophoto was taken
 - **HP** = High-Resolution orthophoto (to distinguish from “**QAD**” = Lower Resolution, “Quick and Dirty”, that we create for field work purposes)

To keep the workflow organised, we encourage you to use the following (or a similar) folder structure:

- MaFa_2025
 - Orthophoto_HP_2025
 - Orthophoto_QAD_2025
 - AC_Kronoren_HP
 - 1_Watermasks
 - 2_Processed_Rasters
 - 3_Segments
 - 4_Stats
 - 5_Trainingdata
 - 6_Classes
 - 7_Validation
 - Final_Result
 - Models
 - Tests

Coordinate System

Always use the official national coordinate system for your region to ensure that layers align correctly and that distance and area calculations are reliable.

In Sweden, the recommended system is **SWEREF 99 TM (EPSG:3006)**.

To verify this in QGIS:

- **Project CRS:** Check the bottom-right corner of QGIS; it should show EPSG:3006 when working in Sweden.
- **Layer CRS:** When loading layers or using them in tools, ensure that they are saved in the correct CRS.

Avoid relying on “on-the-fly” projection. Layers should always be saved in the appropriate coordinate system before proceeding with the analysis.

File Format

For the moment, we recommend using Shapefiles (.shp) during the OTB processing steps, because OTB can still be unstable when reading or writing GeoPackages (.gpkg).

System Settings

By Default, QGIS and its plugins usually don't use the Computer to its full potential. To speed up processing and avoid crashes, check your hardware and adjust the following:

- **Settings → Options → Processing → General**
 - **Max Threads:** Set this to number of CPU cores minus one.
 - For example, on an 8-core machine, use 7 threads.
 - Leaving one core free helps the operating system remain responsive during heavy processing.
- **Settings → Options → Processing → Providers → OTB**
 - **Maximum RAM to use:** Set to about 50-75 % of available RAM

Main Workflow

Step 1 Clip out Water

Purpose

Focus the analysis only on aquatic areas and avoid land pixels. Clip the big orthophoto into smaller Sections (e.g. Bays) with a so-called mask layer (Watermask). Try to make sure the sections aren't too large and clip away areas that are unlikely to be analysable, e.g. very deep water and areas with extreme reflections.

Tool and Process

If there is already a watermask (save the watermasks, so they can be reused in coming years), start with step 2. or 3. Otherwise, create one as follows:

1. Go to **Layer → Create Layer → New Shapefile or GPKG Layer** (To create the Watermask)
 - **Geometry type** = Polygon
 - **Coordinate Reference System** = same as orthophoto (e.g. EPSG:3006 for SWEREF99 TM)
 - **Save** the file in the Watermask Folder for example as: AC_KronorenLillfjarden_Section1_Watermask.shp
2. **Edit** the Watermask layer. (**Pen symbol, enable editing**)
 - Use the “**Add Polygon Feature**” tool to draw the outline of the water body.
 - Close the polygon and save edits.

- If there are islands you want to exclude, use “**Add Ring**” and draw around the island

TIP: Draw it kind of rough first and then use the “**Vertex Tool**” to fine tune

3. Go to **Raster** → **Extraction** → **Clip Raster by Mask Layer**. (Or look it up in the toolbox)
 - **Input raster** = the big orthophoto:
AC_KronorenLillfjarden_250811_HP.tif
 - **Mask layer** = the polygon shapefile of the water area:
AC_KronorenLillfjarden_Section1_Watermask.shp

Parameter Overview

- “Source CRS/Target CRS”: Both set to Project CRS (e.g. EPSG:3006-SWREF99)
- “Keep resolution of input raster”: ensures that the pixel size is identical to the original orthophoto. No resampling of resolution is done.
- “Create an output alpha band”: writes an alpha channel (Band 4) where water pixels are visible (alpha=255) and everything outside is transparent (alpha=0). This prevents black 0,0,0 values outside the mask that could confuse segmentation.
- **Leave the NoData field empty**. If you set NoData=0, QGIS will write 0-values into the RGB bands outside the mask. OTB would then interpret these 0,0,0 pixels as valid data and create straight-line artifacts along the mask edges.

Output

Suggested Name/Format:

- **AC_Kronoren_250811_Section1_Clipped.tif**

Step 2: Segmentation

Purpose

The segmentation step groups neighboring pixels into patches (segments) based on spectral similarity (color/brightness) and spatial context. These segments form the units for statistical analysis.

Tool and Process

Toolbox → **OTB** → **Segmentation** → **Segmentation**

For different resolutions, the parameters have to be adjusted to achieve a comparable result, because most of them are on a pixel-scale. All the parameters, their range and functions are explained under Parameter Overview. The following are some starting point suggestions based on an **RGB with a resolution of 3.5 cm:**

- **Input Image** = _Clipped.tif
- **Segmentation algorithm** = meanshift
- **Spatial radius** = 12
- **Range radius** = 3
- **Convergence Threshold** = 0.15
- **Number of Iterations** = 100
- **Minimum region size** = e.g. 816
- **Processing mode** = vector
- **Writing mode for output** = ulco
- **8-neighbor connectivity** = True
- **Stitch polygons** = True
- **Minimum object size** = e.g. 816 (same as min region size)
- **Simplify polygons** = 0.1 (default)
- **Tile size** = 2048-8192 (see Parameter Overview)

Parameter Overview

- **Segmentation algorithm = meanshift**
 - Iteratively shifts each pixel towards the mean of its neighborhood in both space and spectral (color) space,
 - groups stabilized pixels into segments.

- **Spatial radius (filter.meanshift.spatialr)**
 - How far the algorithm looks in pixel units to define a neighborhood.
 - In pixels, so adequate setting depends on resolution
 - Smaller values → sensitive to local detail, but can fragment.
 - Larger values → smooths over small objects, merges more aggressively.
 - Recommended starting point: **8-20**

- **Range radius (filter.meanshift.ranger)**
 - Tolerance for spectral similarity (pixel values).
 - Lower values (0-5) → separates subtle differences, produces many small segments.
 - Higher values (>15) → merges shades/brightness variations, fewer and larger segments.
 - Recommended starting point: **3-7**

- **Mode Convergence Threshold (filter.meanshift.thres)**
 - Defines when the iterative shifting stops.

- Lower values (<0.05) → strict convergence, more precise clusters, slower.
 - Moderate values (0.1–0.2) → good balance between accuracy and runtime.
 - High values (>0.3) → stops early, risk of under-segmentation.
 - Recommended starting point: **0.1** (safe default).
- **Maximum number of Iterations (filter.meanshift.maxiter)**
 - Upper cap on how many times pixels are shifted before the algorithm stops trying.
 - Higher = more accurate but slower.
 - Normally 100–200 is plenty, only has to be raised if strange blocky artefacts emerge
 - Recommended starting point: **100**
 - **Minimum region size (filter.meanshift.minsize)**
 - Any segment smaller than this number of pixels will be merged into a neighbor. Controls noise and speckles during segmentation.
 - Smaller values preserve finer details, but results in many small polygons. Larger values produce cleaner output, but may remove small real vegetation patches.
 - We have chosen a practical minimum mapping unit of 1 m².

This is in pixels so if you want a certain minimum segment size in sqm, calculate:

- Desired minsize ÷ Image Resolution²
- e.g. we want 1 sqm at a resolution of 3.5 cm → $1 \div 0.035^2 = 816$
- or 1 sqm at a resolution of 0.15 → $1 \div 0.015^2 = 4444$

- **Minimum object size (mode.vector.minsize)**
 - Removes vectorized objects smaller than this pixel area after segmentation.
 - Acts as double check and removes segments smaller than the desired minsize.
 - Keep at **same value as minimum region size** to enforce consistent minimum polygon size.
- **Processing mode**
 - Use **vector mode**.
 - Raster mode is quicker, but the rest of the workflow works with polygons, so we stick with that.

- **Writing mode for output**
 - **ulco (default)**: create a new layer without overwriting existing ones.
 - Only change if you explicitly want to overwrite or update an existing file.

- **8-neighbor connectivity**
 - Enables diagonal pixel adjacency.
 - Results in more natural-looking boundaries.
 - Recommended: **enabled**.

- **Stitch polygons**
 - Combines polygon pieces split across tile boundaries when they belong to the same segment.
 - Helps reduce hard edges and seams.
 - Recommended: **enabled**

- **Simplify polygons**
 - Simplifies vector geometry for smoother shapes.
 - Lower values preserve detail; higher values may degrade segmentation quality.
 - Recommended: **0.10**

- **Tiles Size**
 - OTB is good at multithreading, meaning efficiently using multiple cores at the same time, which speeds up the Segmentation Process significantly.
 - It splits the whole raster into tiles and then segments them in parallel across multiple cores
 - Size you enter here depends on the available RAM, number of available Cores and the size of the raster you want to segment.
 - Too small: creates too many tiles, increasing overhead and producing too many boundary artifacts (especially below 2048).
 - Too large: reduces multithreading efficiency (whole raster is just two tiles for example) and can slow down processing if tiles no longer fit comfortably into memory.
 - The goal is to choose a tile size that's large enough to avoid artifacts but small enough to keep all cores busy.

Examples for a large raster:

- With 8 GB RAM, use **2048** to stay memory-safe.
- With 16 GB RAM and 8 cores, a tile size of **2048–4096** is usually good.
- On a powerful machine (32–64 GB RAM, 16–32 cores), try **4096–8192** – this keeps all cores busy while minimizing artifacts.

Common issues/frustrations

- **Too many visible straight line artifacts along tile boundaries**
-> Increase tile size if RAM allows.
- **Very slow runtime**
-> Check that tile-size isn't too large to split processing
-> Reduce spatial radius
- **OTB fails or crashes on large high-resolution rasters**
-> Lower tile size
-> Check OTB memory limit in QGIS settings.
- **Oversegmentation, too many small segments**
-> Increase range radius and spatial radius slightly
- **Undersegmentation, segments that span multiple classes**
-> Decrease range radius and spatial radius slightly
- **Large bare bottom segments with vegetation patches that don't get separated**
-> Decrease spatial radius first

Step 3: Calculate Statistics

Purpose

Once the segments are ready, we need to calculate statistics for each segment. This step connects the pixel values (from the different bands/layers) to the segments. The statistics of the different bands are later used to train the model. The OTB ZonalStatistics tool by default calculates: Mean, Standard Deviation, Minimum and Maximum.

Tools and Process

Toolbox: OTB → Image Manipulation → Zonal Statistics

- **Input Layer** = AC_Kronoren_250811_Section1_Clipped
- **Type of Input** = vector
- **Input vector data** = AC_Kronoren_250811_Section1_Segments
- **Output pixel type** = float (allows for decimals)
- **Output** = AC_Kronoren_250811_Section1_Stats.shp

Step 4: Add Training- and Validationdata

Purpose

Prepare the data needed for training and validating the classification model by selecting the reference points (training and validation data) that fall within the specific project section. This ensures that only relevant and spatially corresponding samples are used for model development and accuracy assessment.

Tools and Process

1. Load the big Layer with the Referencedata Points into QGIS, then select the ones in the Section you are working on by:
 - **Menu Bar:** Vector → Research Tools → Select by Location:
 - **Input layer** = Referencedata
 - **Intersect layer** = Watermask
 - **Select** → points that fall inside section.
 - **Save the selected points as a new layer:**
AC_Kronoren_250811_Section1_Referencedata
2. Split the reference points into: 70 % Training; 30 % Validation
 - **Toolbox:** Vector selection → Random selection
 - **Method:** Percentage of selected features → 70
 - **Save the selected points as a new layer:**
AC_Kronoren_250811_Section1_Trainingdata
 - **Invert selection and save as:**
AC_Kronoren_250811_Section1_Validationdata
3. Create a new Shapefile for the training data that we feed into the model
 - **Menu Bar:** Layer → Create Layer → New Shapefile Layer
 - **File name:** AC_Kronoren_250811_Section1_Modelfeed
 - **Geometry Type:** Polygon
 - **New field** called HUB_Value; Type = Integer
 - **New field** called Grov_Value; Type = Integer
 - **OK**
 - **Layer:** Properties → Manage Custom Forms and Field Editor
 - Click **“Reuse last entered value”** for the fields HUB_Value and Grov_Value
4. Select segments used for training by drawing a polygon over them.

- Set the Modelfeed-Layer in **Edit-Mode** and click: **Add Polygon Feature**
- Draw a polygon over the Segments you want to add as a class for model training
- Different Classifiers work best with different training data.
 - **SVM:** Works best when the image is clear and the habitats are relatively homogeneous, with consistent water colour and light conditions. It requires few but very representative training segments, and classes must be clearly separated without much overlap or noise.
 - **Random Forest:** Performs better on more challenging images, for example when water colour varies, habitat transitions are messy, or there is noise such as shadows, reflections or mixed vegetation. It benefits from larger and more varied training datasets and handles overlap, unbalanced classes and noisy samples more robustly.

Important:

- Make sure all classes have a similar number of training segments, otherwise the model tends to misclassify rarer classes.
- Make sure the polygons don't accidentally overlap segments that are very different from the class you are adding. Like adding a bright rock to a dark chara patch.

Step 5: Join stats to segments, train the classifier and model the segments

Purpose

Next, the training polygons are linked to the statistics we calculated, so each segment obtains its attribute values. These combined features are used to train the classifier.

Tools and Process

1. Join: Link the training data (reference points/polygons) to the segments so that each segment gets a class.
 - **Toolbox:** Vector General → Join attributes by location
 - **Join to features in** = AC_Kronoren_250811_Section_1_Zstats
 - **Features they** = Intersect
 - **By comparing to** = AC_Kronoren_250811_Section_1_Modelfeed
 - **Join type** = one-to-one
 - **Discard records which could not be joined** = yes
 - **Joined layer** = AC_Kronoren_250811_Section_1_Join

2. Train classifier: Use the training segments to train our classification model
 - **Toolbox:** OTB → TrainVectorClassifier
 - **Input Vector Data** = AC_Kronoren_250811_Section1_Join
 - **Field names for training features** = All mean and std
 - Min and Max rarely help and often just add noise.
 - **Field containing the class integer for supervision** = HUB_value
 - **Classifier to use for training** = libsvm
 - If the classification struggles (for example due to variable water colour, shadows, or mixed habitats), switch to Random Forest and add more and more varied training polygons. Random Forest handles noisy or complex data more robustly.
 - **Output model** = AC_Kronoren_250811_Section1_Model

3. Model segments: Use the model to predict classes of the other segments
 - **Toolbox:** OTB → VectorClassifier
 - **Input Vector Data** = AC_Kronoren_250811_Section1_Stats
 - **Model File** = AC_Kronoren_250811_Section1_Model
 - **Fields to be calculated** = HUB_Value
 - **Output** = AC_Kronoren_250811_Section1_Classes

Different Classifiers work best with different training data.

SVM: Works best with few but very clean and representative training segments. Classes should be clearly separated, without much overlap or noise (e.g. sunglints or dark shadows).

Random Forest: Performs better with more and more varied training data. It handles overlap, unbalanced classes, and noisy samples much more robustly. But it also requires more training segments.

Step 6: Model Refinement

Purpose

After the first modelling run, it is important to check the classification before continuing with validation and output. This is an iterative process: you review where the model performs well or poorly, then refine the training data in the areas that show problems.

Tools and Process

Zoom into different areas and inspect whether the segments have been classified correctly. Look at structured vegetation, mixed patches, shadows, rock surfaces, and areas where classes often overlap (e.g. Myriophyllum and Potamogeton).

If you see clear mistakes or systematic patterns of misclassification:

- adjust the training polygons (add more or remove bad ones)
- make sure each class has enough and representative samples
- refine segments if they are too large or too small for your vegetation types
- rerun the classifier after adjustments

- This step is iterative.
- The first classification run is never the final version.

Usually a few rounds of improving the training data and rerunning the model are needed until the result looks consistent and reliable across the whole section.

Tip:

- If a habitat appears in visually different forms within the same section (for example Chara that looks brownish or bare bottom that becomes darker with depth), consider creating an additional class for these variants.
- Distinguishing visually different sub-types can reduce confusion in the model and significantly improve classification accuracy.

Step 7: Validation and Output

Purpose

Once the iterative refinement is complete and the classification results are satisfactory given the image quality, it is time to evaluate the accuracy of the model, document the results, and export the final files. These outputs can then be used for reporting, mapping, and future analyses, such as monitoring habitat changes over time.

Process

Load the necessary layers together in QGIS:

- AC_Kronoren_250811_Section1_Classes (final classification)

- AC_Kronoren_250811_Section1_Validationdata (30 % of the reference points)
 - Orthophoto of the section (visual comparison)
1. Rasterize the final classification:
 - **Toolbox:** Rasterize (vector to raster)
 - **Input** = _Classes.shp
 - **Burn-in field** = HUB_Value/Grov_Value
 - **Output cell size** = original ortho resolution
 - **Output type** = UInt8
 - **Output** = _Classes_HUB.tif/_Classes_Grov.tif
 2. Compute confusion matrix.
 - **Toolbox:** OTB → ComputeConfusionMatrix
 - **Input** = _Classes.tif
 - **Output format** = confusionmatrix
 - **Ground truth** = vector
 - **Input reference vector data** = _Validationdata
 - **Field name** = HUB_Value/Grov_Value
 - **Output** = _CM_HUB.csv/_CM_Grov.csv
 3. Save result files
 - **Folder:** Final_Result
 - Save the best classification results as **raster**, both in HUB and in Grov, together with their respective confusion matrix, e.g.:
 - AC_Kronoren_250811_Section1_Classes_Grov.tif
 - AC_Kronoren_250811_Section1_CM_Grov.csv
 - AC_Kronoren_250811_Section1_Classes_HUB.tif
 - AC_Kronoren_250811_Section1_CM_HUB.csv

Interpretation

The **confusion matrix** shows, for each class, how the validation points are distributed between:

- their true class (reference data)
- the predicted class (classified raster).

Rows represent the reference classes and columns the predicted classes. From this table you can see which classes are confused with each other, how many points are correctly or incorrectly classified, and the user's and producer's accuracy for each class.

- **User accuracy:** How often the mapped pixels for a class are correct.
 - If the map labels a pixel as this class, what is the chance it is right?
- **Producer accuracy:** How well the model found all real occurrences of a class.
 - Of all true reference points for this class, how many were correctly mapped?

Below the matrix, the OTB ConfusionMatrix tool reports two summary statistics:

- **Kappa value** indicates how your classification compares with a completely random classification. However, the usefulness of this value is questionable and don't be too discouraged should you get a low value.
- **Overall accuracy** shows the percentage of reference points that match the predicted class in the raster. Keep in mind that overall accuracy can be influenced by several factors:
 - Reference points may have positional uncertainty.
 - Some habitats are visually very similar, making them difficult to separate.
 - Accuracy values are usually higher for coarse classes (e.g., vegetation vs. bare bottom) and lower for detailed habitat classes.

Interpretation guidelines for kappa and accuracy values:

<0	= worse than random
<0,2	= poor
0,21-0,4	= fair
0,41-0,6	= moderate
0,61-0,8	= good
>0,8	= very good

```
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Confusion matrix (rows = reference labels,
columns = produced labels):
[11] [33] [37] [38] [39] [49] [62]
[11] 22 1 0 7 0 4 0
[33] 0 9 0 3 0 1 0
[37] 1 1 6 2 0 0 0
[38] 4 0 0 30 0 2 0
[39] 0 0 0 0 1 0 0
[49] 2 0 0 5 0 11 0
[62] 0 0 0 1 0 0 0

2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Precision of class [11] vs all: 0.758621
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Recall of class [11] vs all: 0.647059
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: F-score of class [11] vs all: 0.698413

2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Precision of class [33] vs all: 0.818182
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Recall of class [33] vs all: 0.692308
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: F-score of class [33] vs all: 0.75

2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Precision of class [37] vs all: 1
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Recall of class [37] vs all: 0.6
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: F-score of class [37] vs all: 0.75

2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Precision of class [38] vs all: 0.625
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Recall of class [38] vs all: 0.833333
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: F-score of class [38] vs all: 0.714286

2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Precision of class [39] vs all: 1
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Recall of class [39] vs all: 1
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: F-score of class [39] vs all: 1

2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Precision of class [49] vs all: 0.611111
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Recall of class [49] vs all: 0.611111
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: F-score of class [49] vs all: 0.611111

2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Precision of class [62] vs all: 0
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Recall of class [62] vs all: 0
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: F-score of class [62] vs all: 0

2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Precision of the different classes:
[0.758621, 0.818182, 1, 0.625, 1, 0.611111, 0]
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Recall of the different classes: [0.647059,
0.692308, 0.6, 0.833333, 1, 0.611111, 0]
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: F-score of the different classes:
[0.698413, 0.75, 0.75, 0.714286, 1, 0.611111, 0]

2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Kappa index: 0.596725
2024-11-25 16:29:30 (INFO) ComputeConfusionMatrix: Overall accuracy index: 0.699115
Execution completed in 0.97 seconds
```