

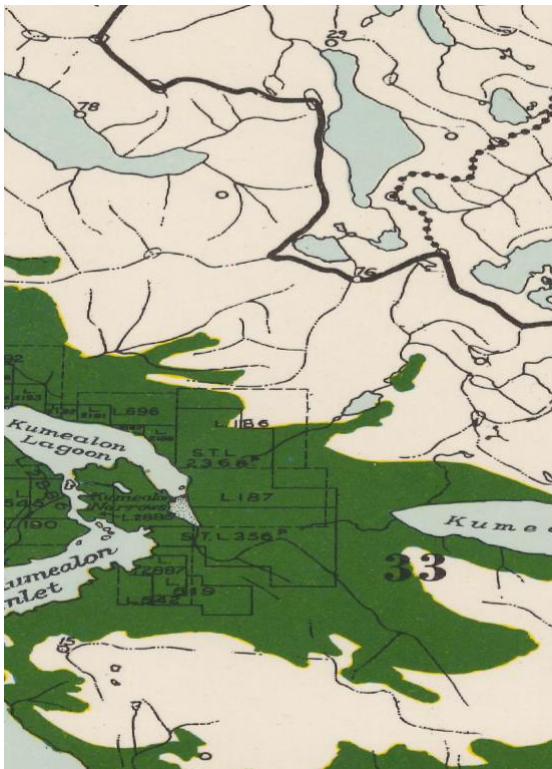
# User Guide

---

## Digital Resurrection of Historical Maps Using Artificial Intelligence Project **Choropleth Map Segmentation Tool**

Designed for the NFIS team at the Pacific Forestry Centre, Natural Resources Canada  
Authored by Team FourTrees, Camosun College ICS Capstone 2020

*Version 0.03 (August 2020)*



# Table of Contents

Introduction .....	1
CMS Tool Overview .....	2
Instructions.....	3
Installation & Setup .....	4
File Management .....	7
Warning.....	7
Preparing Map Files for Input .....	8
A. Direct Download .....	8
B. Local or Network Copying .....	8
Using the CMS Tool.....	9
A. Single Image Processing Option.....	10
B. Batch Image Processing Option.....	13
Example of Segmentation Results.....	15
Viewing Shapefiles.....	16
Summary .....	24
Conclusion .....	25
Appendix .....	26
Appendix A: K-Means vs HSV Segmentation Methods.....	1
Appendix B: A Guide to Adapting the Choropleth Map Segmentation Tool.....	2
Introduction .....	2
1. Determine the Hue-Saturation-Value (HSV) Intensities of the Map Colours .....	3
2. Modify the CMS Tool .....	5
A. Adjust the Colour Category Variables .....	5
B. Adjust the HSV Ranges .....	8

# List of Figures and Tables

Figure 1 – Install Anaconda for your operating system. ....	4
Figure 2 – Confirm Anaconda installation. ....	4
Figure 3 – Download the CMS Tool git repository. ....	5
Figure 4 – Folder structure of the cloned CMS Tool git repository. ....	5
Figure 5 – Install the conda environment. ....	5
Figure 6 – Confirm creation of drhmai environment. ....	5
Figure 7 – Activate the conda environment. ....	6
Figure 8 – Download a map geoTIFF using the command-line interface. ....	8
Figure 9 – CMS Tool process flowchart. ....	9
Figure 10 – Select the number of input images to process. ....	10
Figure 11 – Choose to crop an image. ....	10
Figure 12 – Choose a segmentation method. ....	11
Figure 13 – Complete the HSV segmentation and file conversion processes. ....	11
Figure 14 – Choose a threshold value. ....	11
Figure 15 – The threshold value displayed on a pixel intensity histogram plot. ....	12
Figure 16 – Choose a 'k' value for the number of clusters. ....	12
Figure 17 – Complete the k-means segmentation and file conversion processes. ....	12
Figure 18 – Select the batch processing option. ....	13
Figure 19 – Example of processing output files after image processing is complete. ....	14
Figure 20 – Example of segment output files after image processing is complete. ....	14
Figure 21 – Example of shapefile output after image processing is complete. ....	14
Figure 22 – Example input image 092O_NESE_P_35_cropped.tif cropped to size 2000 x 2000 pixels. ....	15
Figure 23 – Example output after HSV segmentation. ....	15
Figure 24 – Example output after k-means segmentation with a threshold of 80 and k-value of 5. ....	15
Figure 25 – Create a new project in QGIS. ....	16
Figure 26 – Add a vector layer. ....	16
Figure 27 – Select a shapefile to view. ....	17
Figure 28 – Map shapefile after being loaded into QGIS. ....	17
Figure 29 – Navigate to the vector layer properties. ....	18
Figure 30 – Navigate to the Load Style menu. ....	18
Figure 31 – Load style from file. ....	19
Figure 32 – Symbolology tab after loading style file. ....	19
Figure 33 – Map shapefile with style applied. ....	20
Figure 34 – Add a raster layer. ....	20
Figure 35 – Select a raster data source. ....	21
Figure 36 – Navigate to the raster layer properties. ....	21
Figure 37 – Modify raster opacity. ....	22
Figure 38 – Map raster and shapefile layers blended together. ....	22
Figure 39 – Detailed map section with raster and vector layers blended together. ....	23
Figure 40 – Detailed map section of the raster layer. ....	23
Figure 41 – Detailed map section of the vector layer. ....	23
Figure 42 – CMS Tool workflow. ....	24
Table 1 – Comparison Summary between K-Means Clustering and HSV Segmentation Methods. ....	1

# Introduction

Welcome to the User Guide for the Choropleth Map Segmentation Tool.

The Choropleth Map Segmentation (CMS) Tool is a command-line interface tool that is designed to convert a specific digitized set of hand-drawn, historical choropleth maps into a modern, analysis-ready format.

This guide will provide you with introductory information and walk you through the necessary steps to install and run the tool on your machine.

- To learn more about the CMS Tool, refer to the **CMS Tool Overview**.
- To install the tool, refer to **Installation & Setup**.
- To run the tool, refer to **Using the CMS Tool**.
- To view output files, refer to **Viewing Shapefiles**.

Details regarding tool development and source code are available in the **Appendix** section at the end of this document and may be of interest to advanced users. Please note that it is not necessary to review or understand this material in order to use the CMS Tool.

## **CMS Tool Overview**

The Choropleth Map Segmentation (CMS) Tool is designed to convert a specific digitized set of hand-drawn, historical choropleth maps into a modern, analysis-ready format. These maps date back to the 1950s and contain data on historical forest cover in British Columbia. They currently exist as scanned geo-referenced Tagged Image File Format (geoTIFF) files; however, this format has limited potential for research and analysis. Converting the map geoTIFFs to a geospatial vector data format – such as a shapefile – can help resolve these limitations as shapefiles offer greater flexibility for data analysis and storage.

The CMS Tool reads in map geoTIFF images and analyzes the colour data across all pixels. It applies image segmentation methods to identify forest cover categories by inspecting and labelling each pixel with the appropriate category based on the original legend for the historical map set. Pixels that cannot be classified into these categories are grouped and removed whereas categorized pixels – or segments – are de-noised to remove any misclassified or outlier pixels. Finally, the finished colour segments are converted and exported separately as shapefiles.

The CMS tool is a product of the Digital Resurrection of Historical Maps Using Artificial Intelligence (DRHMAI) project. This project is sponsored by the National Forest Information System (NFIS) team at the Pacific Forestry Centre for Natural Resources Canada, and is part of the Camosun College Information & Computer Systems Capstone 2020 showcase.

# Instructions

The instructions documented in this section cover the installation and operation of the CMS Tool on Unix-based operating systems.

It is assumed that you have access to a computer, a working internet connection, basic familiarity with using the command-line interface, and access permissions to perform the following actions:

- Download files from GitHub.
- Install open-source software and associated libraries.
- Create directories and files.
- Run commands using the command-line interface.

This section is organized sequentially with instructions grouped by topic:

- To view instructions on retrieving the CMS Tool source code and installing Anaconda and Python, refer to **Installation & Setup**.
- To learn about the file structure of the CMS Tool, refer to **File Management**.
- For instructions on gathering and preparing geoTIFF images for processing, refer to **Preparing Map Files for Input**.
- For instructions on running the CMS Tool using command-line, refer to **Using the CMS Tool**.
- For instructions on how to view the CMS Tool output files, refer to **Viewing Shapefiles**.
- For a quick overview of the entire instruction process, refer to **Summary**.

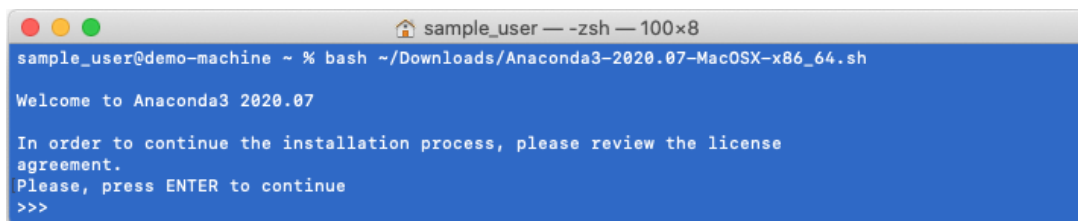
## Installation & Setup

The CMS Tool runs inside an Anaconda environment that contains all the Python libraries it requires to operate. This pre-configured environment is provided with the CMS Tool source code in the file `drhmai_environment.yml` so that it can easily be duplicated.

Please note that installing Anaconda may take a few minutes and about 3.5 GB of storage space.

- Step 1. Download the latest version of Anaconda for your operating system from the [Anaconda website](#)<sup>1</sup>.
- Step 2. Install Anaconda by following the instructions provided in the [Anaconda documentation](#)<sup>2</sup>. An example of the command-line installation for macOS (v10.15.5) is shown in Figure 1.

**Note** – Python 3 is installed with Anaconda and does not have to be set up separately.



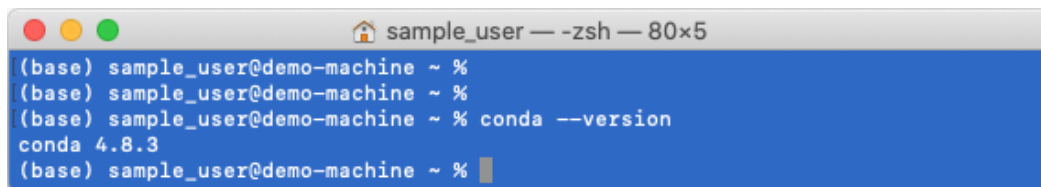
```
sample_user@demo-machine ~ % bash ~/Downloads/Anaconda3-2020.07-MacOSX-x86_64.sh

Welcome to Anaconda3 2020.07

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
```

Figure 1 – Install Anaconda for your operating system.

- Step 3. Restart the command-line terminal application.
- Step 4. Verify that the installation was successful by entering the command `conda --version`, as shown in Figure 2.



```
(base) sample_user@demo-machine ~ %
(base) sample_user@demo-machine ~ %
(base) sample_user@demo-machine ~ % conda --version
conda 4.8.3
(base) sample_user@demo-machine ~ %
```

Figure 2 – Confirm Anaconda installation.

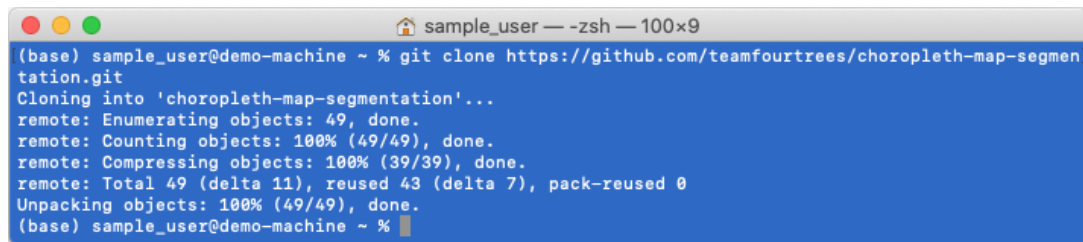
- Step 5. Clone the CMS tool git repository by running the command `git clone https://github.com/teamfourtrees/choropleth-map-segmentation.git`, as shown in Figure 3.

**Note** – If you are unable to use the git command, check that you have [Git installed](#)<sup>3</sup>.

<sup>1</sup> Anaconda download page: <https://www.anaconda.com/products/individual>

<sup>2</sup> Anaconda installation guide: <https://docs.anaconda.com/anaconda/install/>

<sup>3</sup> Git installation guide: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>



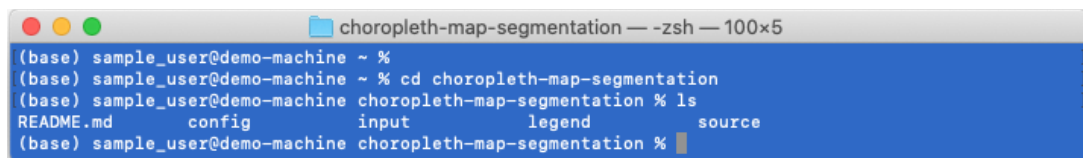
```

sample_user — -zsh — 100x9
(base) sample_user@demo-machine ~ % git clone https://github.com/teamfourtrees/choropleth-map-segmentation.git
Cloning into 'choropleth-map-segmentation'...
remote: Enumerating objects: 49, done.
remote: Counting objects: 100% (49/49), done.
remote: Compressing objects: 100% (39/39), done.
remote: Total 49 (delta 11), reused 43 (delta 7), pack-reused 0
Unpacking objects: 100% (49/49), done.
(base) sample_user@demo-machine ~ %

```

Figure 3 – Download the CMS Tool git repository.

- Step 6. Navigate to the cloned project folder with `cd choropleth-map-segmentation`.
- Step 7. Run the `ls` command to list all available folders and files.
- Step 8. Confirm that the folders appear as shown in Figure 4.



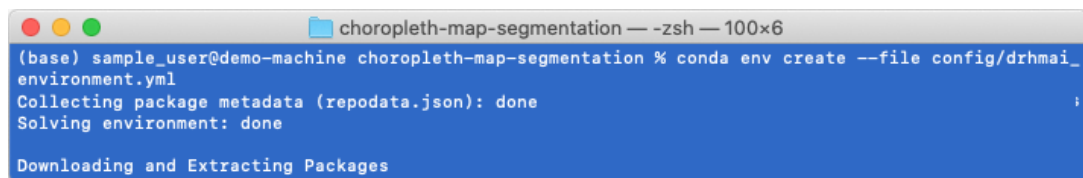
```

choropleth-map-segmentation — -zsh — 100x5
(base) sample_user@demo-machine ~ %
(base) sample_user@demo-machine ~ % cd choropleth-map-segmentation
(base) sample_user@demo-machine choropleth-map-segmentation % ls
README.md      config          input           legend          source
(base) sample_user@demo-machine choropleth-map-segmentation %

```

Figure 4 – Folder structure of the cloned CMS Tool git repository.

- Step 9. Run the command `conda env create --file config/drhmai_environment.yml` to configure the CMS Tool environment as shown in Figure 5.



```

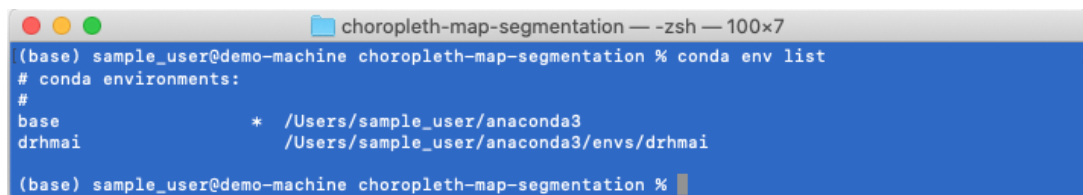
choropleth-map-segmentation — -zsh — 100x6
(base) sample_user@demo-machine choropleth-map-segmentation % conda env create --file config/drhmai_environment.yml
Collecting package metadata (repodata.json): done
Solving environment: done

Downloading and Extracting Packages

```

Figure 5 – Install the conda environment.

- Step 10. Run the command `conda env list` to confirm that the environment has been successfully created. The resulting list should include the `drhmai` environment, as shown in Figure 6.



```

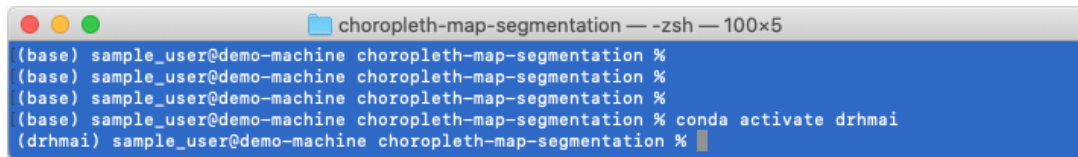
choropleth-map-segmentation — -zsh — 100x7
(base) sample_user@demo-machine choropleth-map-segmentation % conda env list
# conda environments:
#
base                * /Users/sample_user/anaconda3
drhmai              /Users/sample_user/anaconda3/envs/drhmai
(base) sample_user@demo-machine choropleth-map-segmentation %

```

Figure 6 – Confirm creation of drhmai environment.



Step 11. Run the command `conda activate drhmai` to use the new environment. The environment name should appear to the left of the command-line terminal window, as shown in Figure 7.

A terminal window titled "choropleth-map-segmentation — -zsh — 100x5" with a blue background. It shows a sequence of commands and prompts. The first three lines are "(base) sample\_user@demo-machine choropleth-map-segmentation %". The fourth line is "(base) sample\_user@demo-machine choropleth-map-segmentation % conda activate drhmai". The fifth line is "(drhmai) sample\_user@demo-machine choropleth-map-segmentation %" with a cursor at the end.

```
(base) sample_user@demo-machine choropleth-map-segmentation %  
(base) sample_user@demo-machine choropleth-map-segmentation %  
(base) sample_user@demo-machine choropleth-map-segmentation %  
(base) sample_user@demo-machine choropleth-map-segmentation % conda activate drhmai  
(drhmai) sample_user@demo-machine choropleth-map-segmentation %
```

*Figure 7 – Activate the conda environment.*

The environment is now installed and ready for running the CMS Tool.

## File Management

All files relating to the CMS Tool are stored in the `choropleth-map-segmentation` folder, which contains the following subfolders, this user guide, and a readme file:

- `config` – Contains configuration details for installing the anaconda environment and for styling shapefiles.
- `input` – Stores geoTIFF images placed by the user for processing.
- `legend` – Contains images from the original map legend used for analysis of pixel categories. These files are not directly used by the CMS Tool and can be ignored.
- `output` – Stores output for each image processed by the CMS Tool.
- `source` – Contains source code for the CMS Tool and a script to analyze map legend pixels.
- `CMS_Tool_User_Guide.pdf` – Contains step-by-step instructions for the CMS Tool.
- `README.md` – Contains a brief overview of the tool.

The CMS Tool automatically seeks out and reads in any geoTIFF image files stored in the `input` folder. It then creates an `output` folder (if one does not already exist) to store the intermediate and final results of the segmentation process.

The `output` folder contains a sub-folder for every image processed by the tool. Each sub-folder contains a log file (`log.txt`) that records key processing parameters along with all process outputs associated with a specific input image. These output files are organized into:

- A `processing` folder.
  - This contains visual evidence of the changes being made to the image file at each step in the algorithm.
  - These files are saved in `.png` format and are numbered sequentially.
- A `segments` folder.
  - This contains the individual map segments saved as `.tiff` files.
  - The files are named according to their corresponding forest cover category.
  - The category labels are derived from the original map legend for this historical map set.
- A `shapefiles` folder.
  - This contains the map segments in their polygonized form and constitutes the final output of the CMS Tool.
  - Each converted segment has a `.shp`, `.shx`, `.dbf`, and `.cpg` file associated with it.
  - If required, these files can be viewed using GIS software to analyze the geospatial data.

## Warning

The `choropleth-map-segmentation` repository (i.e. CMS Tool's project folder) **is not designed to store processing or output files long-term.**

If the same input image is processed multiple times, all corresponding files in the `output` folder will be overwritten to store the most recent output results. To prevent this, review the output files after each run and copy them to a different location for long-term storage.

## Preparing Map Files for Input

The CMS Tool requires input in the form of TIFF images. It is designed to work with map geoTIFFs, but will accept image files with the following extensions:

- .tif or .TIF
- .tiff or .TIFF
- .gtif or .GTIF
- .gtiff or .GTIFF

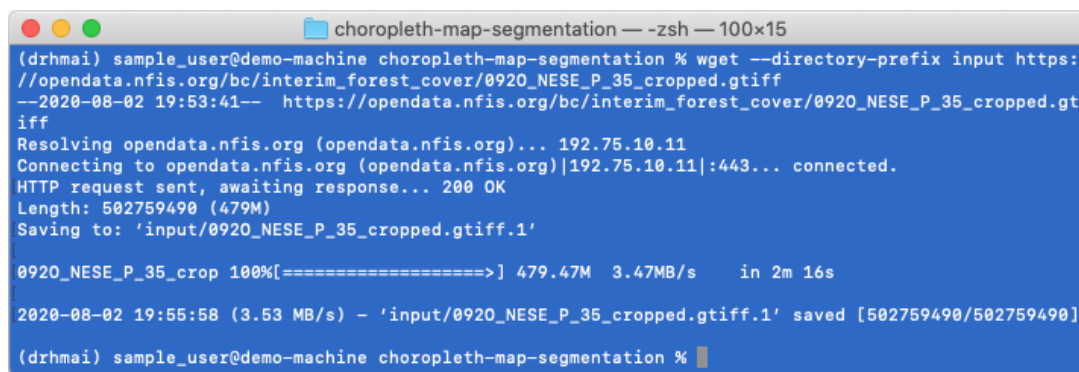
Images can be loaded to the input folder using any of the following ways:

### A. Direct Download

- Step 1. Navigate to the `choropleth-map-segmentation` project folder.
- Step 2. Run the command `wget --directory-prefix input <mapURL>`. Be sure to replace `<mapURL>` with a complete URL such as `https://opendata.nfis.org/bc/interim_forest_cover/0920_NESE_P_35_cropped.gtiff`.

This will retrieve a geoTIFF from a given map URL and download it directly to the `input` folder, as shown in Figure 8.

**Note** – If `wget` does not work, you may need to install it (or an alternative) first.



```

choropleth-map-segmentation — -zsh — 100x15
(drhmai) sample_user@demo-machine choropleth-map-segmentation % wget --directory-prefix input https://opendata.nfis.org/bc/interim_forest_cover/0920_NESE_P_35_cropped.gtiff
--2020-08-02 19:53:41-- https://opendata.nfis.org/bc/interim_forest_cover/0920_NESE_P_35_cropped.gtiff
Resolving opendata.nfis.org (opendata.nfis.org)... 192.75.10.11
Connecting to opendata.nfis.org (opendata.nfis.org)|192.75.10.11|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 502759490 (479M)
Saving to: 'input/0920_NESE_P_35_cropped.gtiff.1'

0920_NESE_P_35_crop 100%[=====] 479.47M  3.47MB/s   in 2m 16s

2020-08-02 19:55:58 (3.53 MB/s) - 'input/0920_NESE_P_35_cropped.gtiff.1' saved [502759490/502759490]

(drhmai) sample_user@demo-machine choropleth-map-segmentation %

```

Figure 8 – Download a map geoTIFF using the command-line interface.

### B. Local or Network Copying

- Step 1. Use the `cp` or `scp` commands to copy map geoTIFFs from another location on your machine or network.

It is best to refer to your organization's policies and procedures on using `ssh` and `scp` for remote sharing or copying of files.

## Using the CMS Tool

The CMS Tool consists of a Python program that is run using the command-line interface. The tool requests input from the user where required, and provides feedback as it runs through the image segmentation process. This process flow can be represented as a flowchart, as shown in Figure 9.

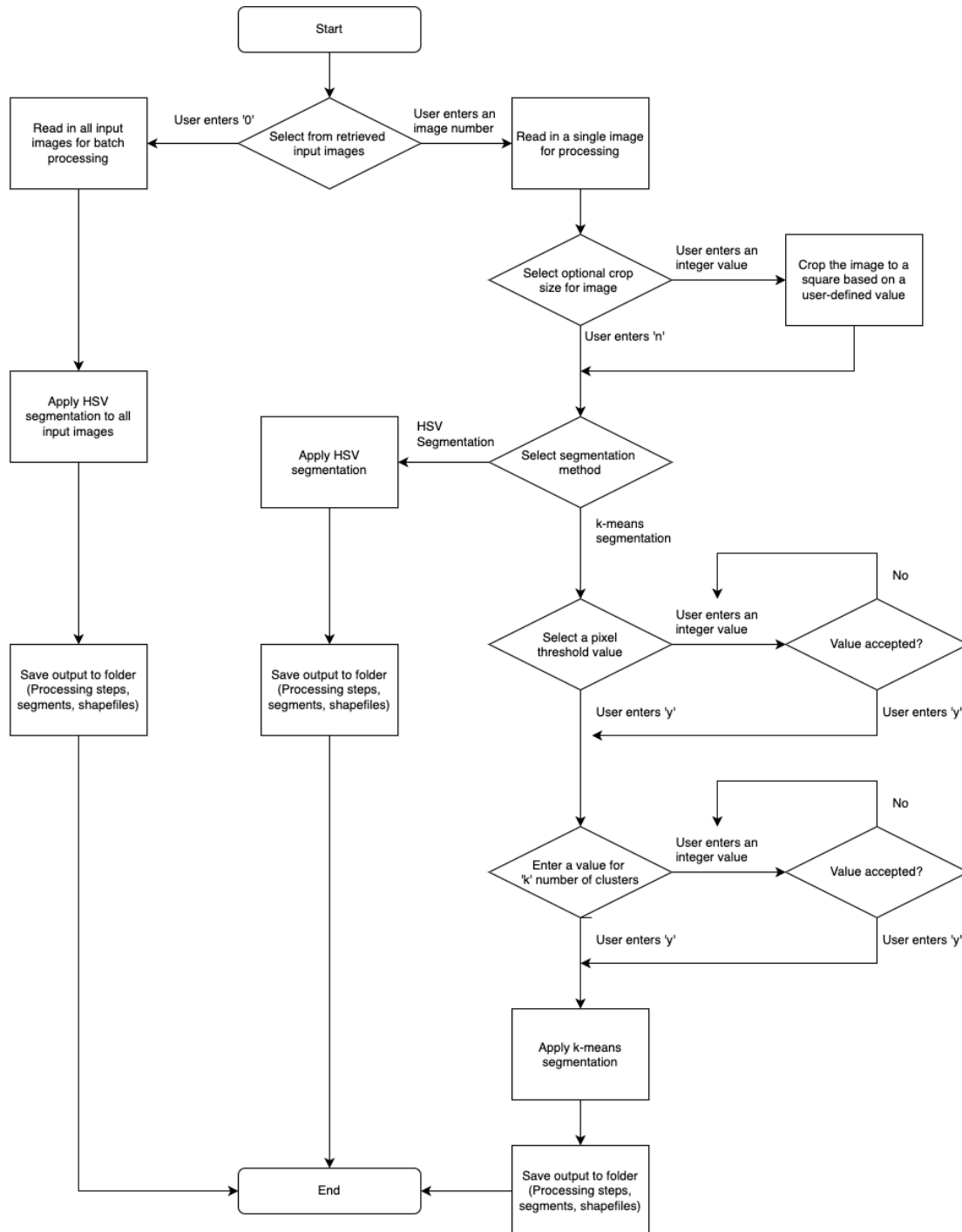
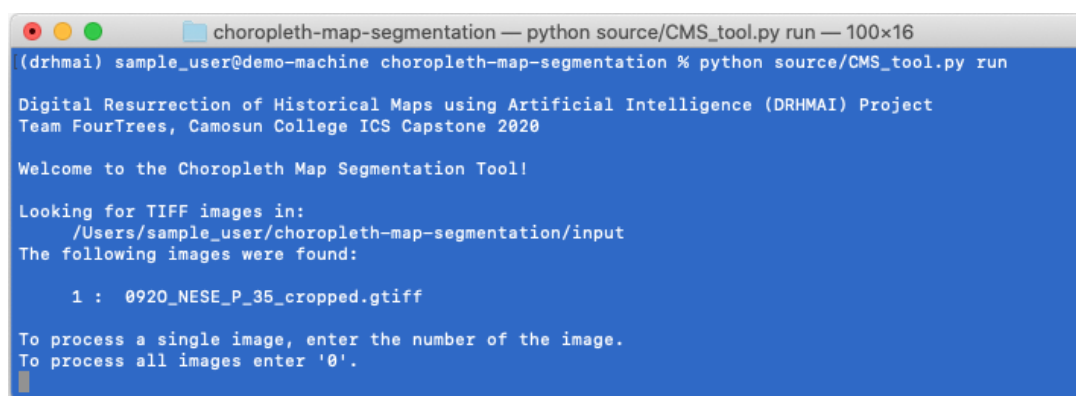


Figure 9 – CMS Tool process flowchart.

- Step 1. Change directory to the `choropleth-map-segmentation` folder.
- Step 2. Activate the environment by running `conda activate drhmai`.
- Step 3. Run the command `python source/CMS_tool.py run` to start the program. You will see a welcome message followed by a list of images retrieved from the `input` directory, as shown in Figure 10.
- Step 4. Read the prompt and make a selection between:
  1. Processing a single image of your choice by entering the corresponding list number.
  2. Processing all the available input images as a batch by entering `0`.

For more details on single image processing, see Steps 5-6. For more details on batch image processing, see Step 7.



```

choropleth-map-segmentation — python source/CMS_tool.py run — 100x16
(drhmai) sample_user@demo-machine choropleth-map-segmentation % python source/CMS_tool.py run

Digital Resurrection of Historical Maps using Artificial Intelligence (DRHMAI) Project
Team FourTrees, Camosun College ICS Capstone 2020

Welcome to the Choropleth Map Segmentation Tool!

Looking for TIFF images in:
/Users/sample_user/choropleth-map-segmentation/input
The following images were found:

1 : 0920_NESE_P_35_cropped.gtiff

To process a single image, enter the number of the image.
To process all images enter '0'.

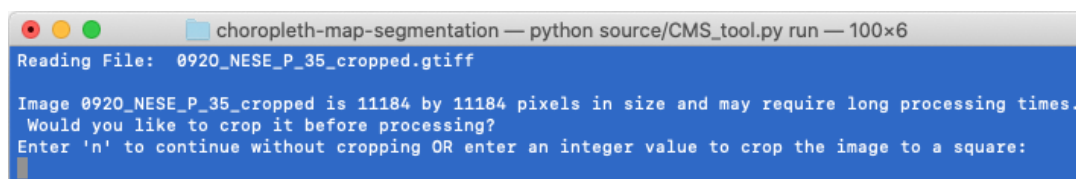
```

Figure 10 – Select the number of input images to process.

#### A. Single Image Processing Option

- Step 5. Read the prompt and choose whether you want to crop the map image to a smaller size, as shown in Figure 11.
  1. To ignore this option and proceed with the full image, enter `n`.
  2. To crop the image, enter an integer value (1000 to 3000 is recommended).

This step is useful for testing or building a proof of concept as it can help reduce processing times and output file sizes. However, this method is limited to cropping the shape outward from the upper left corner, and is not intended for specific area cropping.



```

choropleth-map-segmentation — python source/CMS_tool.py run — 100x6
Reading File: 0920_NESE_P_35_cropped.gtiff

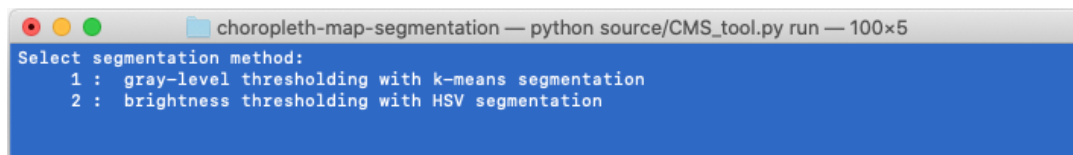
Image 0920_NESE_P_35_cropped is 11184 by 11184 pixels in size and may require long processing times.
Would you like to crop it before processing?
Enter 'n' to continue without cropping OR enter an integer value to crop the image to a square:

```

Figure 11 – Choose to crop an image.

- Step 6. Read the prompt and choose between segmentation methods, as shown in Figure 12.
  1. To select grayscale-based segmentation with k-means clustering, enter `1`.
  2. To select brightness-based segmentation with hue-saturation-value (HSV), enter `2`.

After you have made your selections, the tool will continue to process the image and provide progress updates in the command-line terminal until it is complete.



```

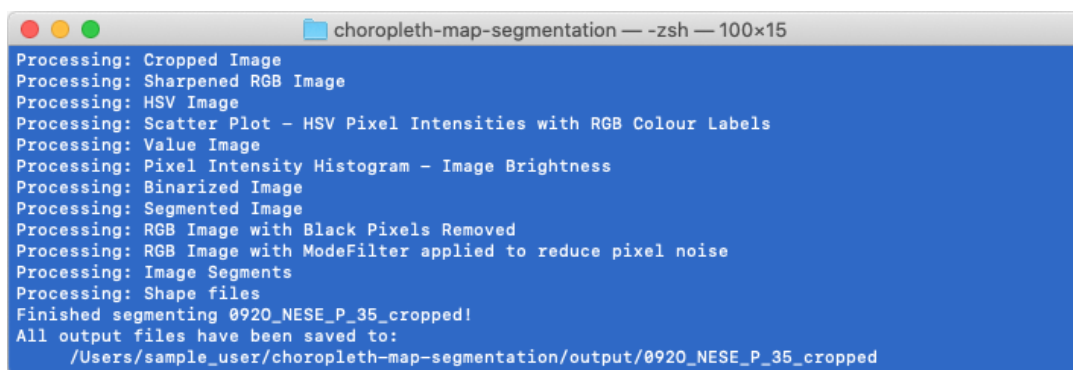
choropleth-map-segmentation — python source/CMS_tool.py run — 100x5
Select segmentation method:
 1 : gray-level thresholding with k-means segmentation
 2 : brightness thresholding with HSV segmentation

```

Figure 12 – Choose a segmentation method.

The **HSV segmentation method is recommended** because it outperforms the k-means method and produces more accurate results since it has been fine-tuned to work with the 1950s map set.

If this method is selected, the resulting terminal output will appear as shown in Figure 13.



```

choropleth-map-segmentation — -zsh — 100x15
Processing: Cropped Image
Processing: Sharpened RGB Image
Processing: HSV Image
Processing: Scatter Plot - HSV Pixel Intensities with RGB Colour Labels
Processing: Value Image
Processing: Pixel Intensity Histogram - Image Brightness
Processing: Binarized Image
Processing: Segmented Image
Processing: RGB Image with Black Pixels Removed
Processing: RGB Image with ModeFilter applied to reduce pixel noise
Processing: Image Segments
Processing: Shape files
Finished segmenting 0920_NESE_P_35_cropped!
All output files have been saved to:
/Users/sample_user/choropleth-map-segmentation/output/0920_NESE_P_35_cropped

```

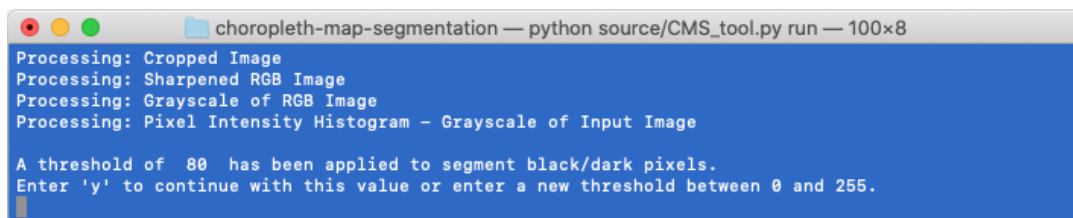
Figure 13 – Complete the HSV segmentation and file conversion processes.

The **K-means segmentation option is less effective**, but is provided as it may be useful for testing and experimentation purposes by advanced users. Selecting this option will require additional user input and decision-making.

- 1) Provide a segmentation threshold value between 0 and 255 as shown in Figure 14.

The threshold is used to establish the boundary between dark and non-dark pixels. Typical values range from 70 to 100. A histogram plot showing the grayscale pixel intensity distribution will be displayed with a line marking the threshold, as shown in Figure 15. You can re-enter the threshold value multiple times to re-plot and observe its location relative to the distribution.

When you are ready to proceed, enter **y**.



```

choropleth-map-segmentation — python source/CMS_tool.py run — 100x8
Processing: Cropped Image
Processing: Sharpened RGB Image
Processing: Grayscale of RGB Image
Processing: Pixel Intensity Histogram - Grayscale of Input Image

A threshold of 80 has been applied to segment black/dark pixels.
Enter 'y' to continue with this value or enter a new threshold between 0 and 255.

```

Figure 14 – Choose a threshold value.

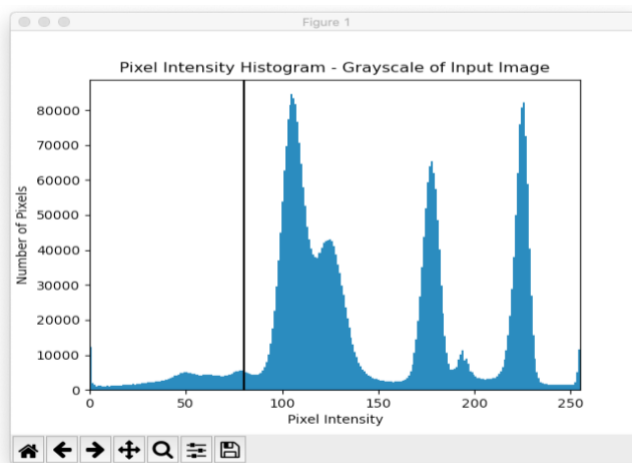


Figure 15 – The threshold value displayed on a pixel intensity histogram plot.

- 2) Enter the number of clusters to use.

This number is the value of 'k' in the k-means algorithm, and represents the number of pixel categories on the map. Typical values range from 3 to 7, depending on the map.

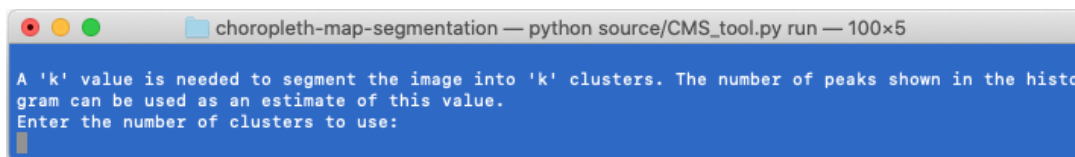


Figure 16 – Choose a 'k' value for the number of clusters.

- 3) Allow the segmentation process to continue and review the resulting output after the process completion message appears, as shown in Figure 17.

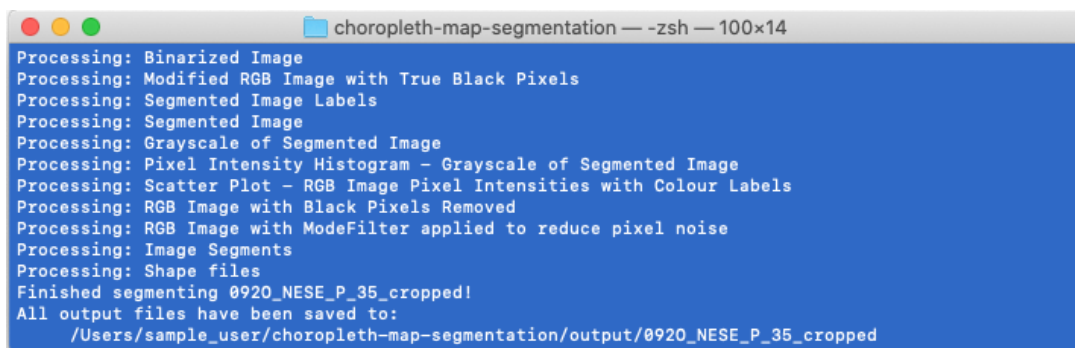


Figure 17 – Complete the k-means segmentation and file conversion processes.

To learn more about the benefits and drawbacks of these two methods, please refer to the **Appendix:**

#### **Appendix A:**

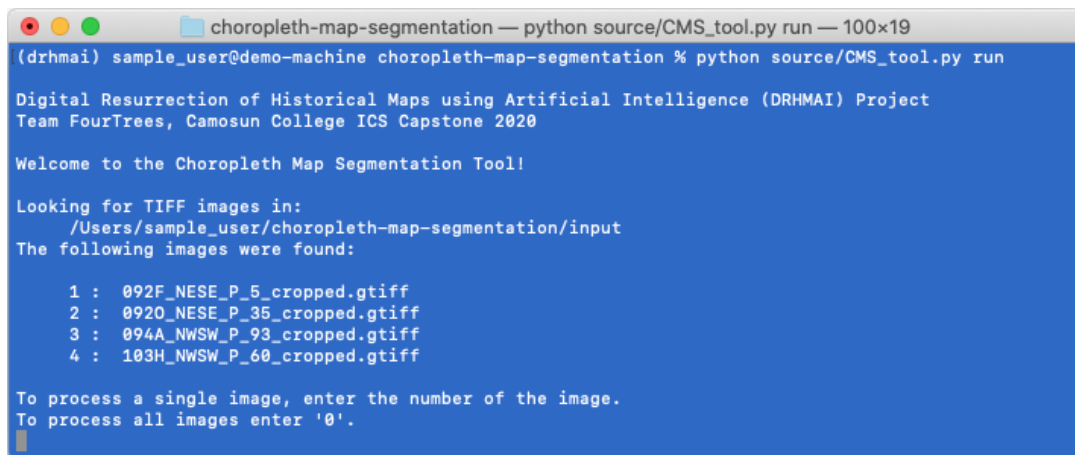
**K-Means vs HSV Segmentation Methods** section.

## B. Batch Image Processing Option

Step 7. Enter `0` to run the batch image processing option and process multiple files at once as shown in Figure 18.

**Note:**

- This option will default to using the (recommended) HSV segmentation method.
- This option does not include the ability to crop an image before processing.
- Images are processed sequentially (i.e. as they appear in the input image list).
- Selecting this option may require long processing times (2 hours or more per image) depending on the computer's hardware and the file size of each geoTIFF.



```

choropleth-map-segmentation — python source/CMS_tool.py run — 100x19
(drhmai) sample_user@demo-machine choropleth-map-segmentation % python source/CMS_tool.py run

Digital Resurrection of Historical Maps using Artificial Intelligence (DRHMAI) Project
Team FourTrees, Camosun College ICS Capstone 2020

Welcome to the Choropleth Map Segmentation Tool!

Looking for TIFF images in:
/Users/sample_user/choropleth-map-segmentation/input
The following images were found:

1 : 092F_NESE_P_5_cropped.gtiff
2 : 0920_NESE_P_35_cropped.gtiff
3 : 094A_NWSW_P_93_cropped.gtiff
4 : 103H_NWSW_P_60_cropped.gtiff

To process a single image, enter the number of the image.
To process all images enter '0'.
0

```

Figure 18 – Select the batch processing option.

Step 8. Confirm that the process is complete by navigating to the `output` directory and reviewing the output files.

The output can be confirmed by viewing the files with a GUI as shown in Figure 19, Figure 20, and Figure 21. These figures show the contents of the different subfolders within the `output` folder after processing. In the absence of a GUI, confirm that output exists by navigating to each of the subfolders in the `output` directory and run the `ls` command to view the enclosed files.

For long-term storage and viewing, copy or export the files to a different location.



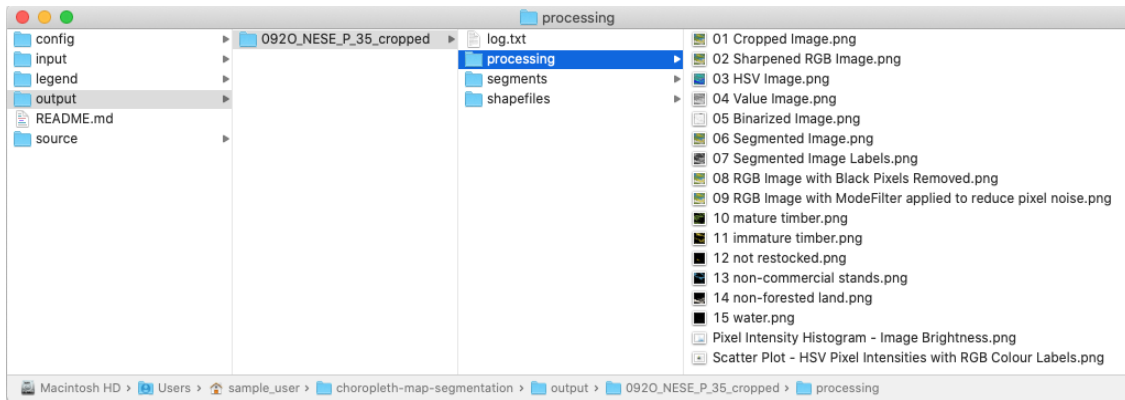


Figure 19 – Example of processing output files after image processing is complete.

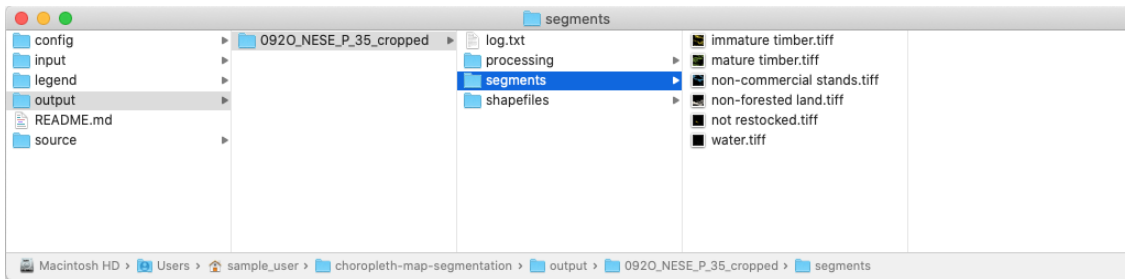


Figure 20 – Example of segment output files after image processing is complete.

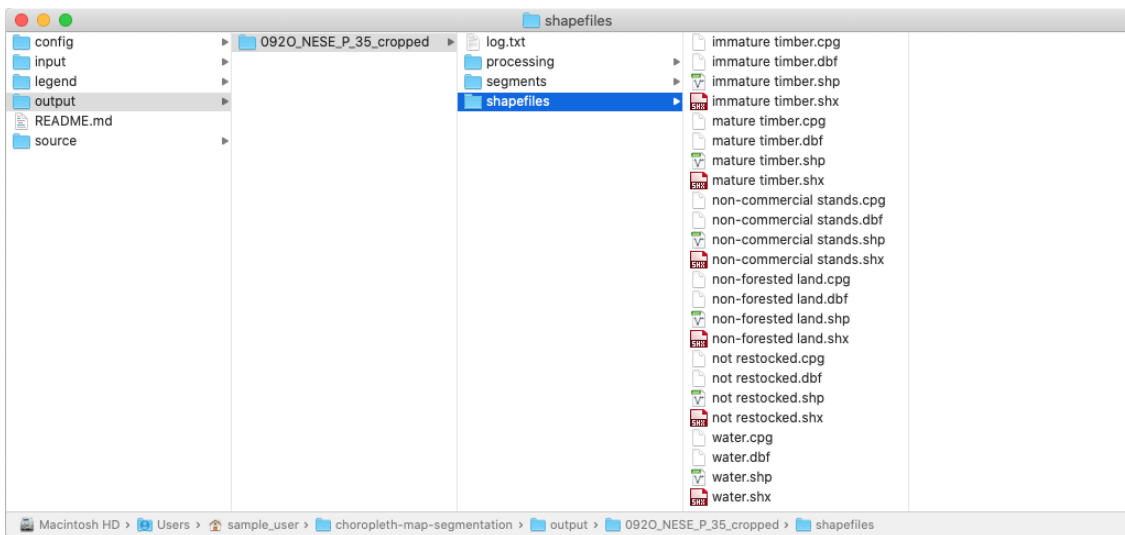


Figure 21 – Example of shapefile output after image processing is complete.

## Example of Segmentation Results

The following graphics present an example of how the CMS Tool modifies and segments an input image (Figure 22) using HSV segmentation (Figure 23) as well as k-means segmentation (Figure 24).

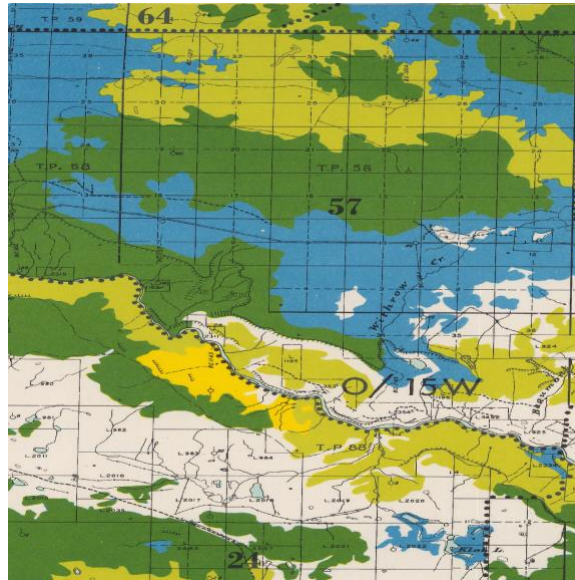


Figure 22 – Example input image 092O\_NESE\_P\_35\_cropped.tif cropped to size 2000 x 2000 pixels.

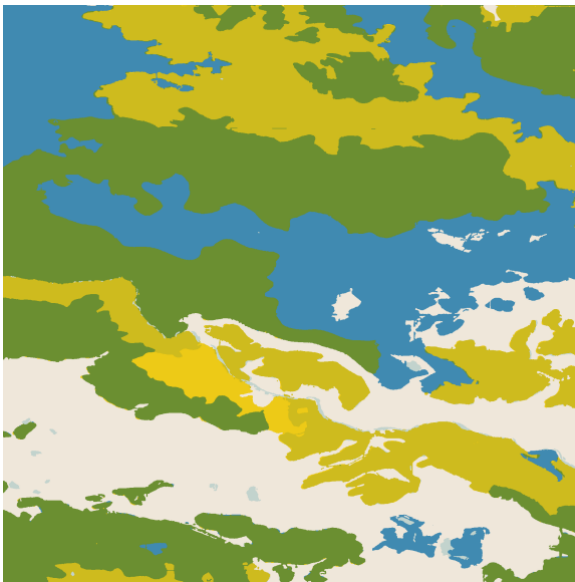


Figure 23 – Example output after HSV segmentation.

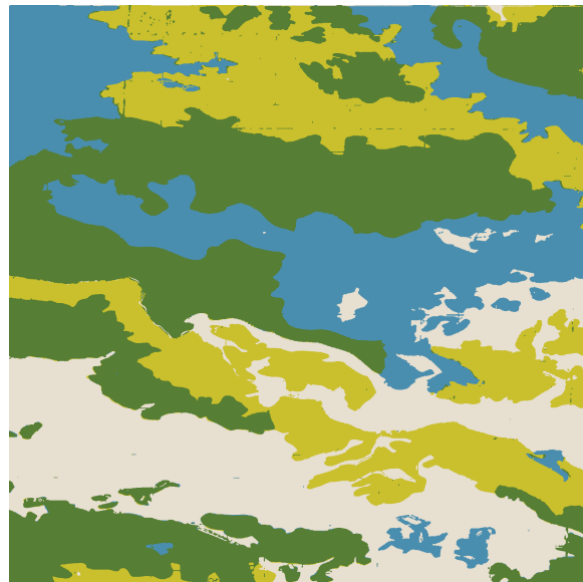


Figure 24 – Example output after k-means segmentation with a threshold of 80 and k-value of 5.

## Viewing Shapefiles

The shapefile outputs obtained from the CMS Tool can be viewed using GIS software. One such example is Quantum GIS (QGIS). The following instructions demonstrate how shapefiles can be viewed and styled within QGIS for further analysis.

- Step 1. Download and install the latest version of QGIS for your operating system from the [QGIS website](https://www.qgis.org/en/site/forusers/download.html)<sup>4</sup>.

**Note** – The examples in this guide use QGIS version 3.10 for macOS.

- Step 2. Launch the application.  
 Step 3. Select *Project* → *New* from the application menu, as shown in Figure 25.

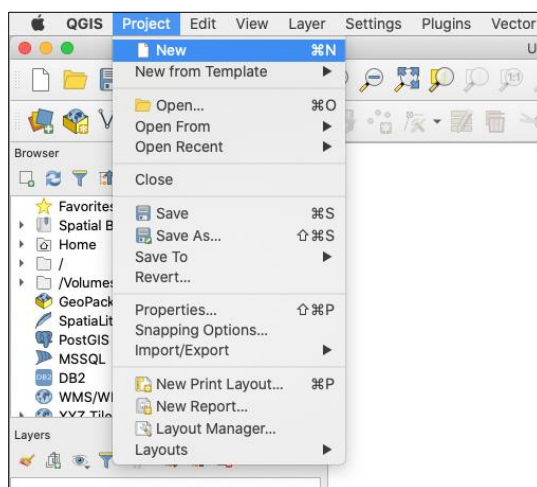


Figure 25 – Create a new project in QGIS.

- Step 4. Select *Layer* → *Add Layer* → *Add Vector Layer*, as shown in Figure 26.

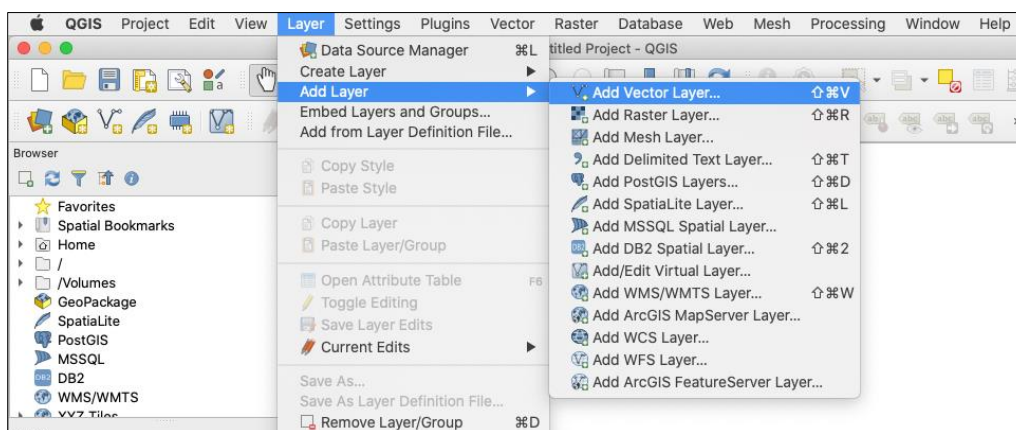


Figure 26 – Add a vector layer.

<sup>4</sup> Download and install QGIS <https://www.qgis.org/en/site/forusers/download.html>

Step 5. Select *File* as the *Source Type*.

Step 6. Choose a shapefile from the CMS Tool's output directory, located in `~/choropleth-map-segmentation/output/<map>/shapefiles/<map>.shp`. Be sure to substitute `<map>` with a specific map of interest (e.g. `0920_NESE_P_35_cropped`), as shown in Figure 27.

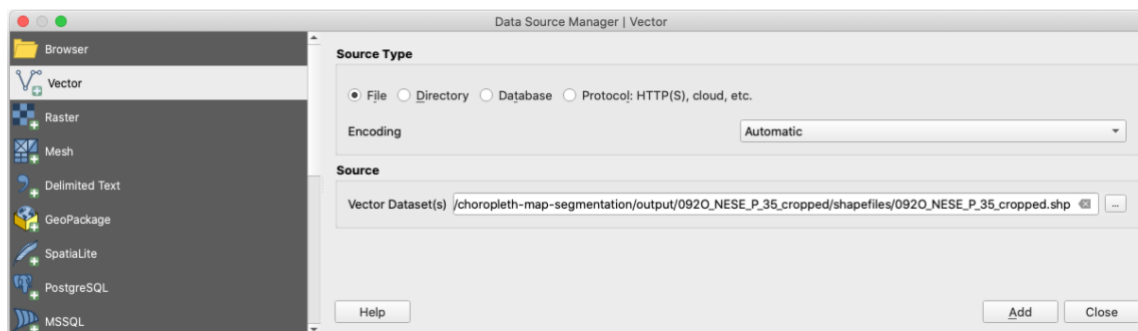


Figure 27 – Select a shapefile to view.

Step 7. Click the *Add* button. The shapefile will appear on the screen, as shown in Figure 28.

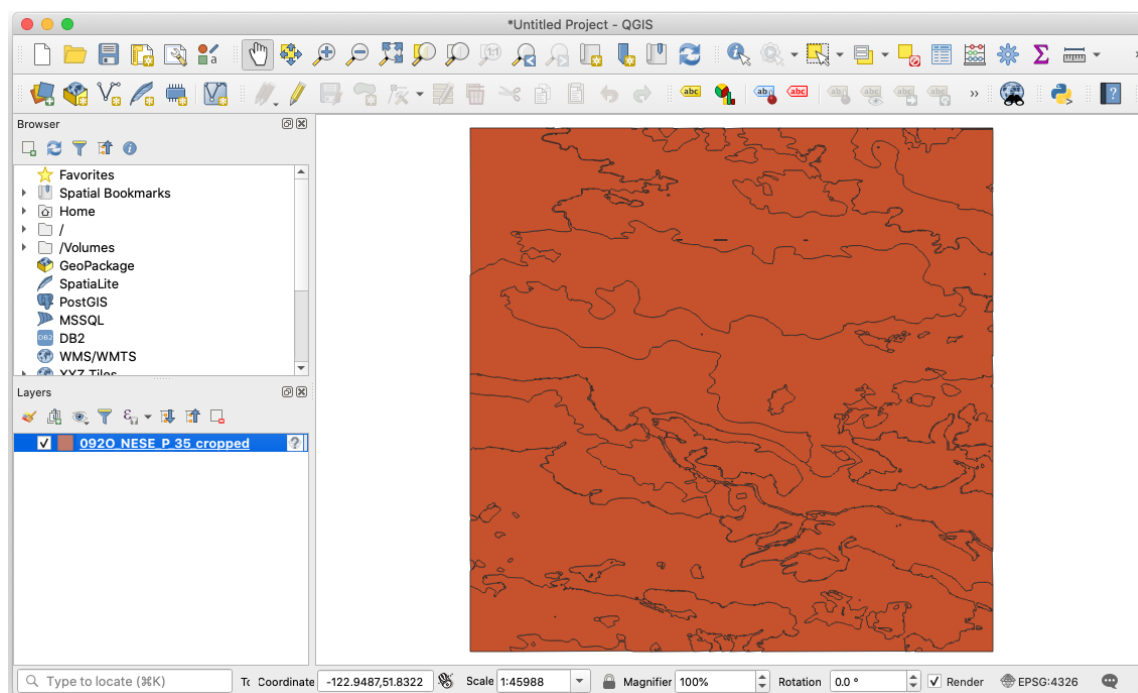


Figure 28 – Map shapefile after being loaded into QGIS.

- Step 8. Right-click on the new layer from the *Layers* panel and select *Properties*, as shown in Figure 29.

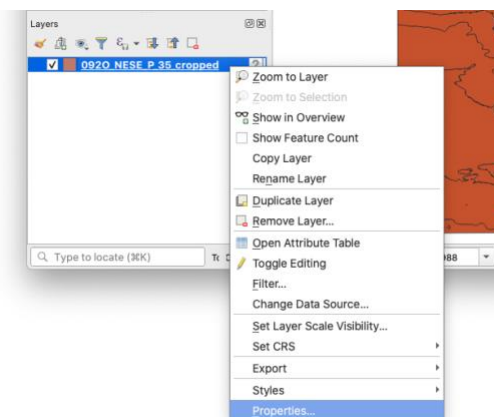


Figure 29 – Navigate to the vector layer properties.

- Step 9. Click *Style* → *Load Style* from the bottom of the *Symbology* tab, as shown in Figure 30.

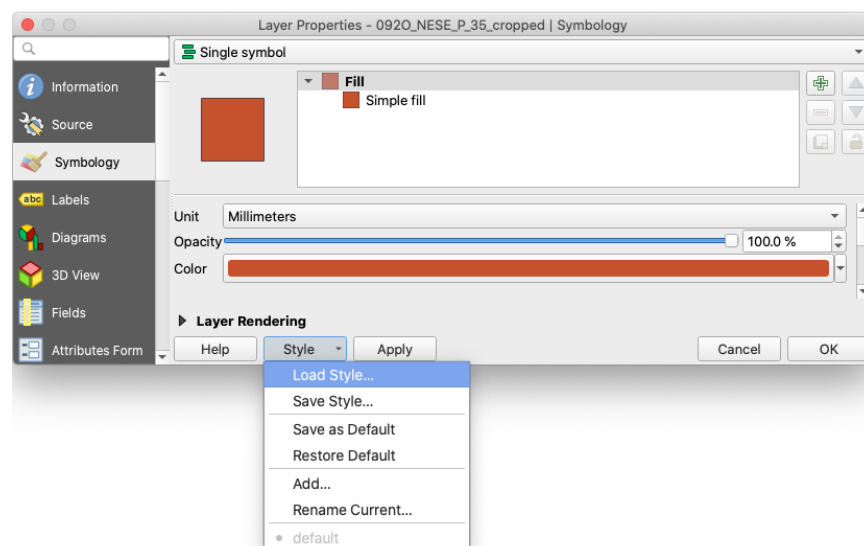


Figure 30 – Navigate to the Load Style menu.

- Step 10. Select the style file `qgis_style.qml` from the `~/choropleth-map-segmentation/config` folder, as shown in Figure 31.

**Note** – This style is configured to work with HSV-segmented shapefiles only. For shapefiles generated using k-means segmentation, a new custom style will need to be created instead (which is beyond the scope of this user guide).

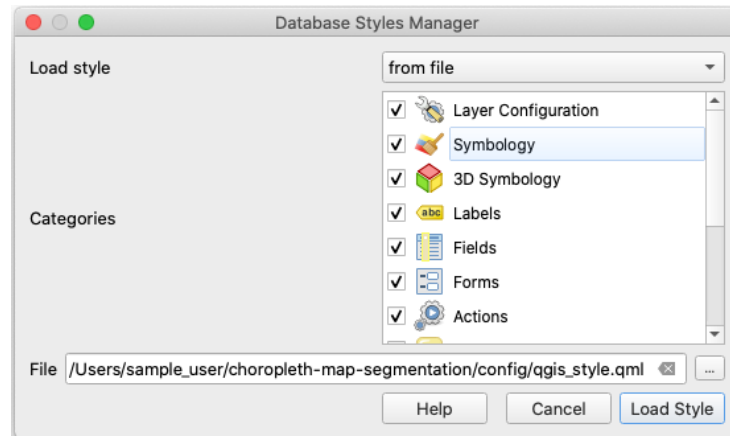


Figure 31 – Load style from file.

Step 11. Click on *Load Style*. The style will be loaded and the *Style Manager* will automatically close.

The *Symbology* tab of the *Layer Properties* menu will now appear as shown in Figure 32.

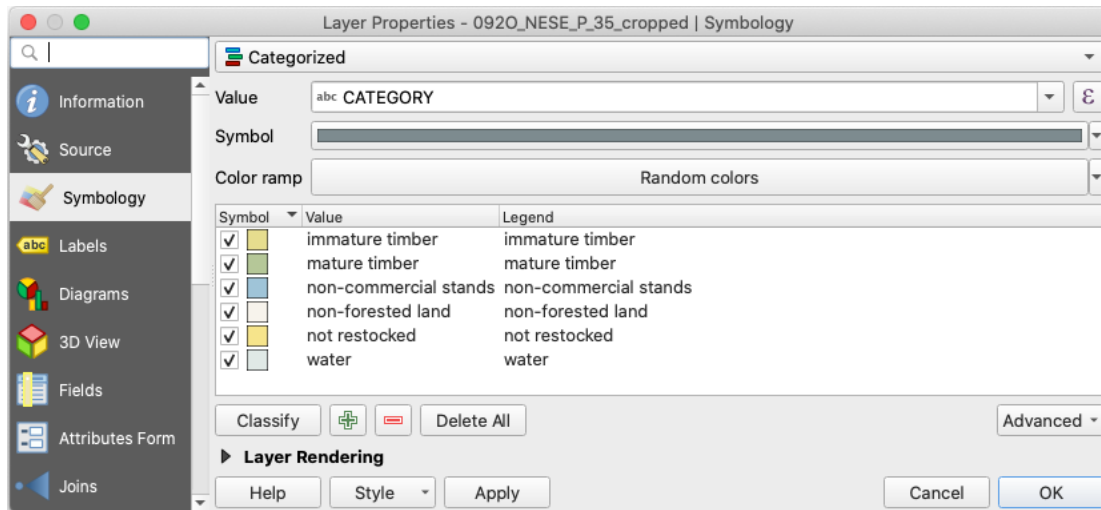


Figure 32 – Symbology tab after loading style file.

Step 12. Click *OK* to close the *Layer Properties* window.

The map will appear as shown in Figure 33 with the different forest cover regions coloured and labelled.

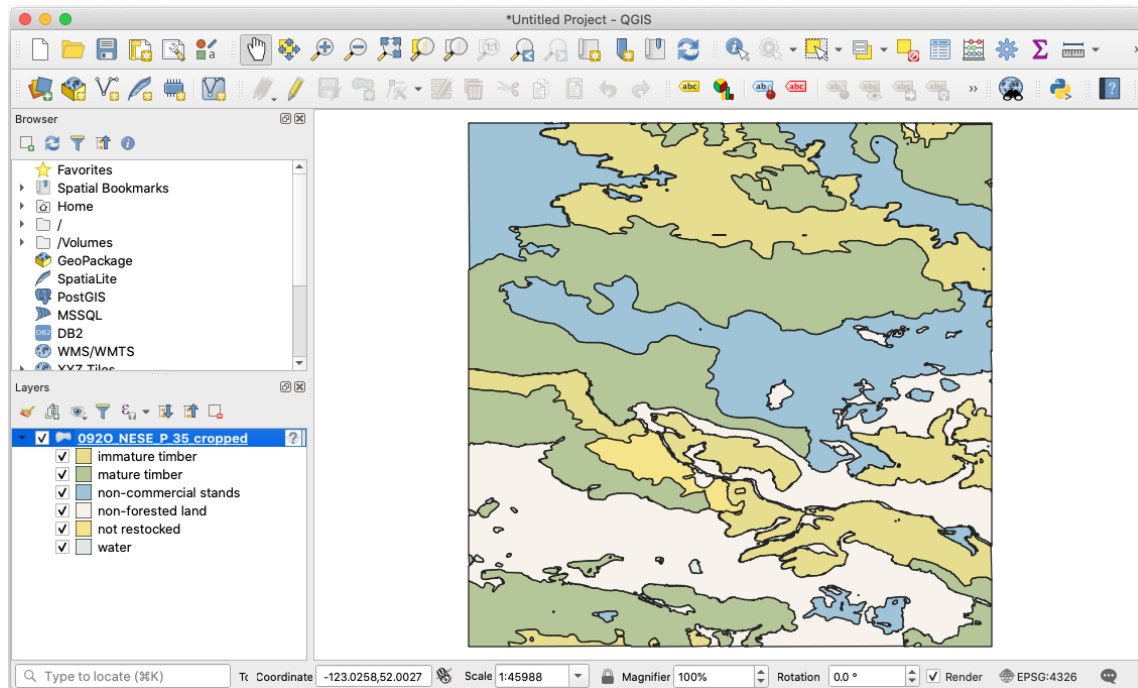


Figure 33 – Map shapefile with style applied.

The shapefile is now ready for analysis. To compare the shapefile to the original raster map, proceed with Step 13 to Step 24.

Step 13. Select *Layer* → *Add Layer* → *Add Raster Layer* from the menu, as shown in Figure 34.

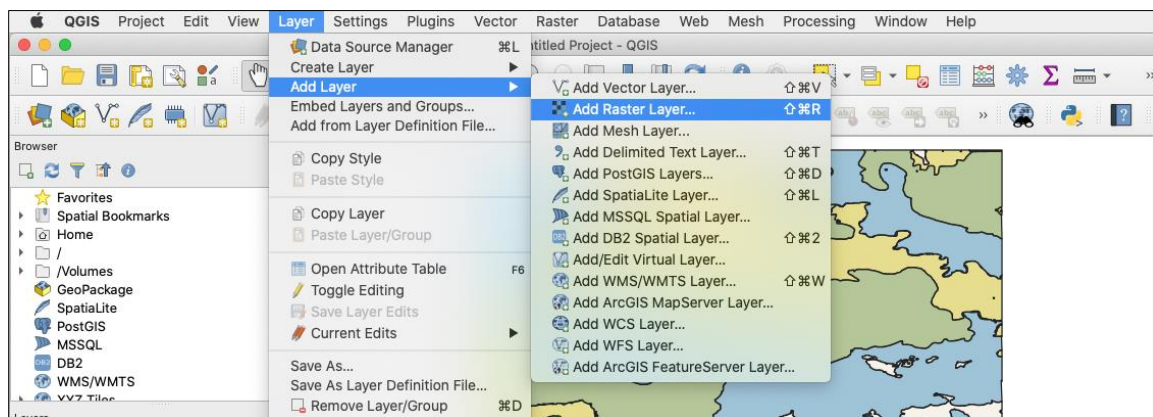


Figure 34 – Add a raster layer.



- Step 14. Select the *Source Type (File or Protocol)*.
- Step 15. Enter the map URL if using the *Protocol Source Type* option, as shown in Figure 35.  
E.g. `opendata.nfis.org/bc/interim_forest_cover/0920_NESE_P_35_cropped.gtiff`.

Alternatively, select a local file from the `~/choropleth-map-segmentation/input` directory or elsewhere, if using the *File Source Type* option.

- Step 16. Click *Add*.
- Step 17. Click *Close*. The map should appear in the foreground as a new layer.

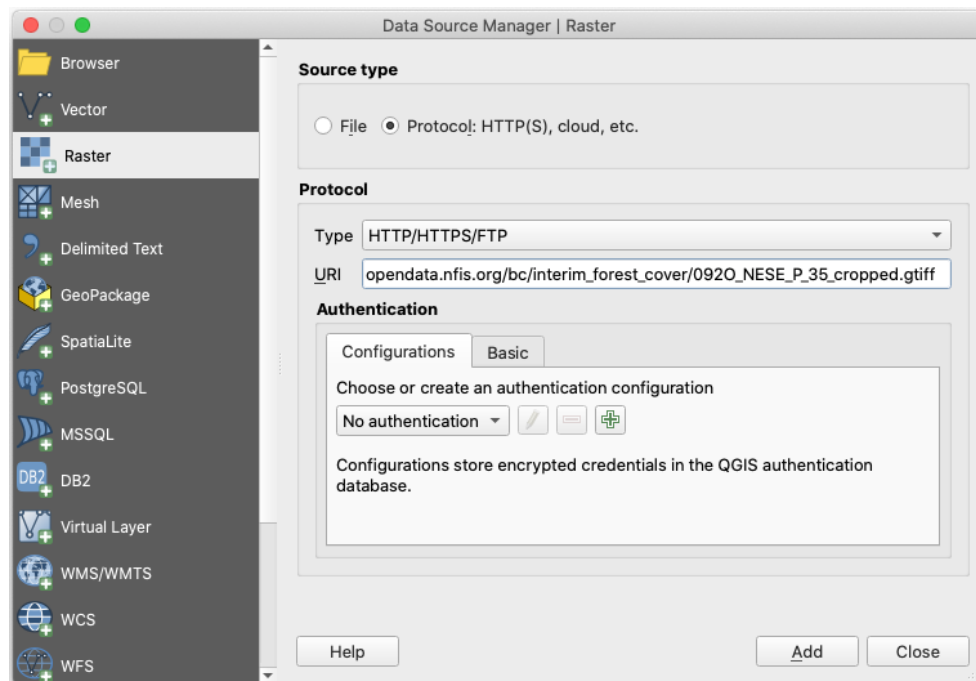


Figure 35 – Select a raster data source.

- Step 18. Right-click on the new layer from the *Layers* panel and select *Properties*, as shown in Figure 36.

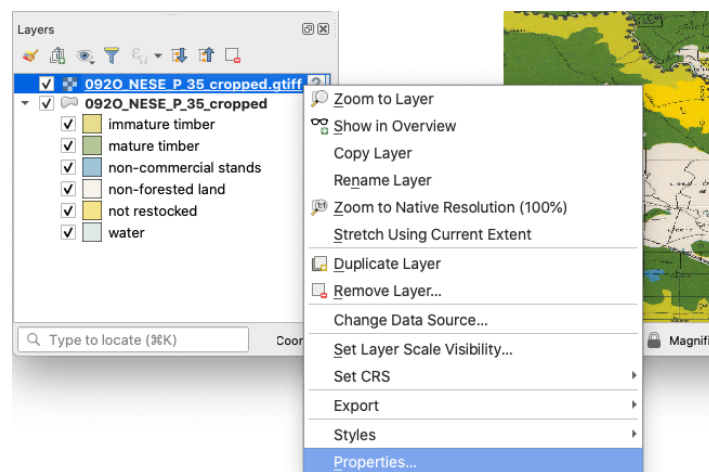


Figure 36 – Navigate to the raster layer properties.



Step 19. Set the *Global Opacity* to 50% in the *Transparency* tab, as shown in Figure 37.

Step 20. Click *OK*.

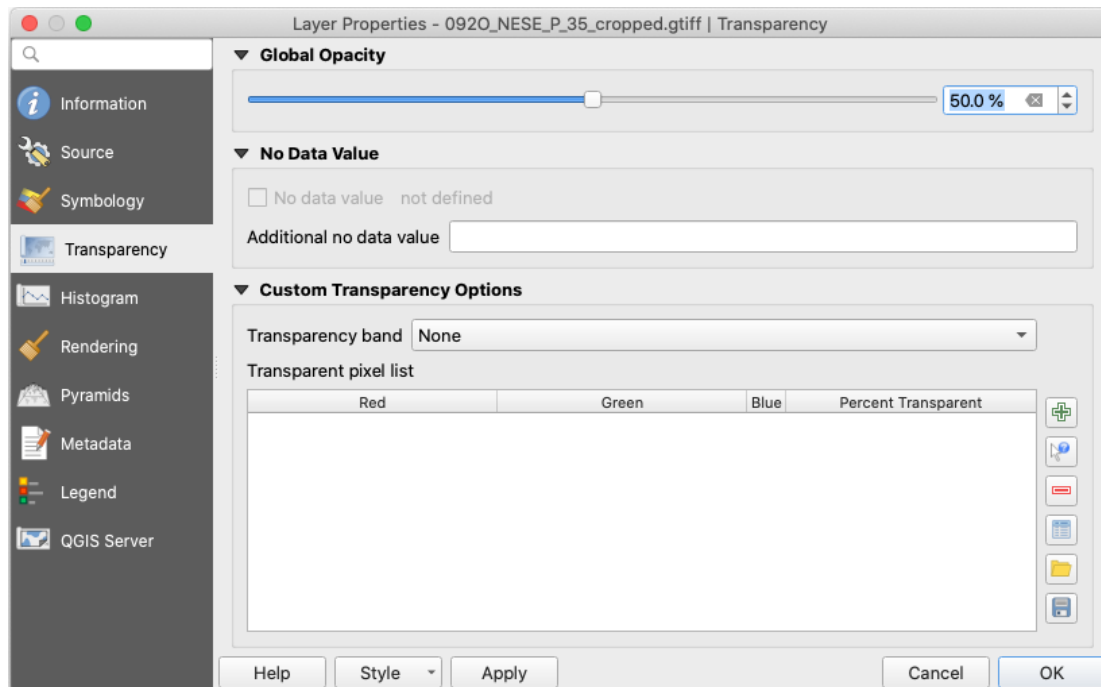


Figure 37 – Modify raster opacity.

The raster image colours should now appear paler and blended with the shapefile as shown in Figure 38.

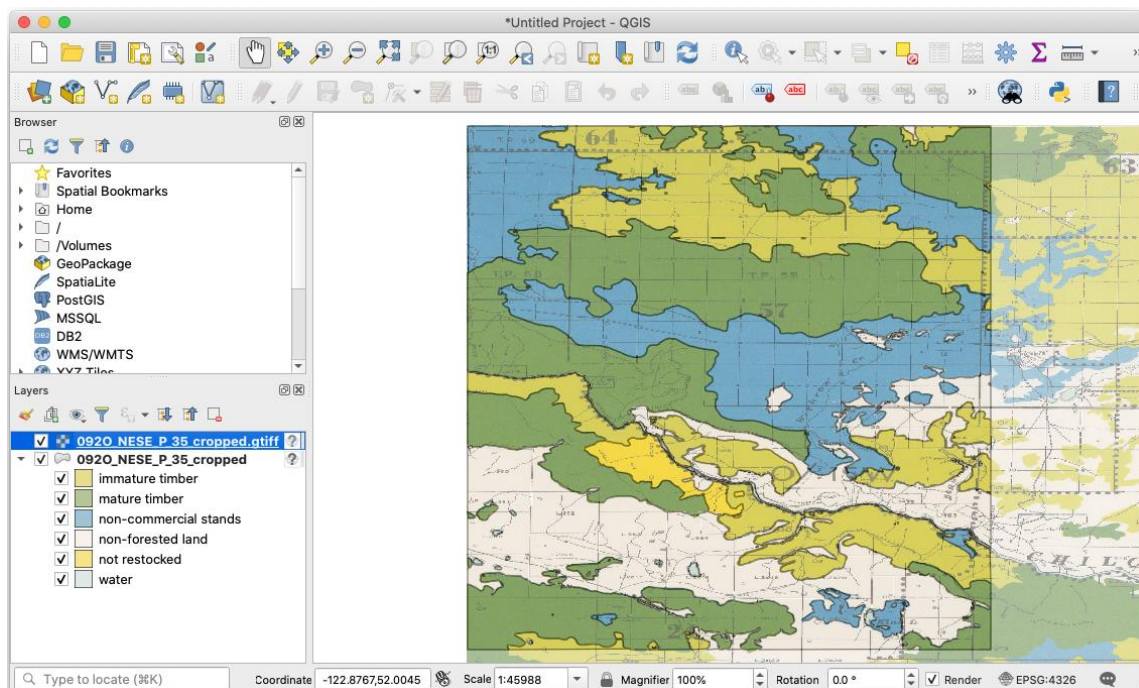


Figure 38 – Map raster and shapefile layers blended together.

Step 21. Zoom in to a small sub-section of the map, e.g. the section shown in Figure 39.

Step 22. Click the check-mark to the left of the vector layer in the *Layers* panel.

This will hide it from the view. The map will now appear similar to Figure 40.

Step 23. Click the check-mark to the left of the vector layer again to unhide it.

Step 24. Click the check-mark to the left of the raster layer in the *Layers* panel.

This will hide it from the view. The map will now appear similar to Figure 41.

Examining the blended view (Figure 39) as well as hiding and revealing the raster and vector layers separately (Figure 40, Figure 41) allows the user to make a detailed comparison between the original raster image and the shapefile output from the CMS Tool.

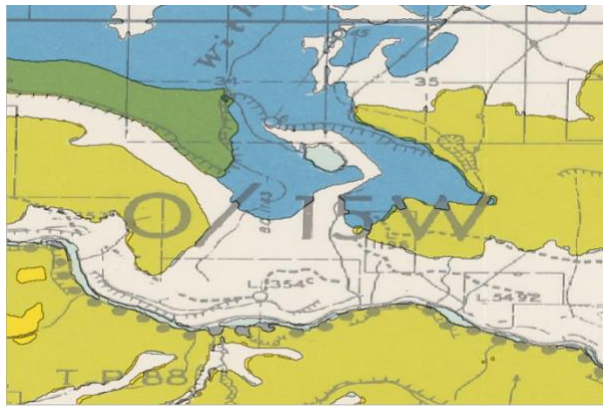


Figure 39 – Detailed map section with raster and vector layers blended together.



Figure 40 – Detailed map section of the raster layer.

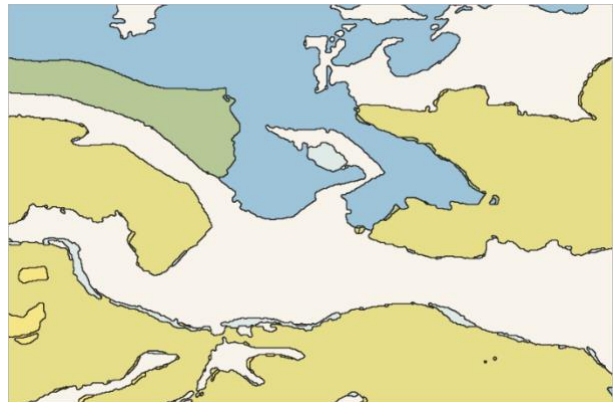


Figure 41 – Detailed map section of the vector layer.

## Summary

To use the CMS Tool, you can expect to follow the workflow shown in Figure 42:

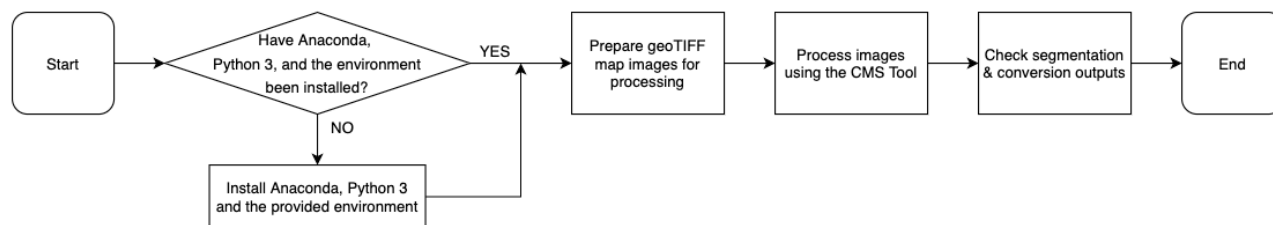


Figure 42 – CMS Tool workflow.

The key steps can be summarized as follows:

1. Prior to using the CMS Tool, install Anaconda and Python 3 onto your machine. Clone the CMS Tool source code from GitHub and activate the provided `conda` environment.
2. Collect map geoTIFFs for processing and place them in the `input` folder. These files will be read in by the CMS Tool.
3. Run the CMS Tool and provide user input when prompted to customize the process.
4. Check the `output` folder to review all image outputs and confirm that the map segments and shapefiles exported properly.
5. View the shapefile output using GIS software for further analysis.

# Conclusion

Thank you for using the CMS Tool.

We hope you found this user guide helpful in providing you with an overview of everything this tool can (currently) do.

Should you have any questions about the installation, operation, or functionality of the tool, please feel free to reach out to us at [teamfourtrees@protonmail.ch](mailto:teamfourtrees@protonmail.ch).

# Appendix

## Appendix A:

# K-Means vs HSV Segmentation Methods

This table summarizes the key benefits and drawbacks of using K-means or HSV segmentation methods on the provided set of 1950s historical forest cover choropleth maps.

*Table 1 – Comparison Summary between K-Means Clustering and HSV Segmentation Methods.*

	K-means	HSV
<b>Advantages</b>	<ul style="list-style-type: none"> <li>• Semi-automatic Unsupervised Machine Learning: The pixel colour classes are determined through comparison between the colour values of a central pixel and all the pixels surrounding it within a specific cluster.</li> <li>• Useful for experimentation and testing, and can be used with more than one type of choropleth map.</li> <li>• The user has the ability to set a custom threshold value and k-value to best fit their segmentation needs.</li> </ul>	<ul style="list-style-type: none"> <li>• Outperforms k-means on segmentation of large map geoTIFF images.</li> <li>• A good option if a legend or classification scheme is provided (i.e. the classes are known in advance).</li> <li>• Ability to fine-tune upper and lower bounds for pixel classes provides greater probability of accuracy.</li> <li>• Does not require input from the user which could be beneficial as it abstracts away complexity.</li> </ul>
<b>Disadvantages</b>	<ul style="list-style-type: none"> <li>• Map images have to be cropped for successful segmentation results since the k-means algorithm tends to work better on smaller images.</li> <li>• Segmentation is inaccurate for large historical map images, likely because of high pixel colour variation, which increases the risk of pixel misclassification.</li> <li>• Requires the user to have some knowledge of how k-means works to use it effectively.</li> </ul>	<ul style="list-style-type: none"> <li>• Colour boundaries have to be manually defined, which means the algorithm has to be modified with new colour data if a different map set is being processed.</li> <li>• Maps that are not compatible with the pre-defined legend may cause inaccuracies with segmentation.</li> </ul>

# **Appendix B:**

## **A Guide to Adapting the Choropleth Map Segmentation Tool**

### **Introduction**

The Choropleth Map Segmentation (CMS) Tool was designed to segment choropleth maps from a specific map set (i.e. the 1956 set of BC Interim Forest Cover Maps). However, it can be adapted to accommodate other choropleth maps with different legend colours by making modifications to its source code.

This document will explain how to prepare the input categories and modify the CMS Tool source code. It assumes that the reader is familiar with the CMS Tool, the Python programming language, and using the command line interface.

The workflow for adapting the CMS Tool is as follows:

- 1. Determine the Hue-Saturation-Value (HSV) Intensities of the Map Colours**
- 2. Modify the CMS Tool:**
  - A. Adjust the Colour Category Variables**
  - B. Adjust the HSV Ranges**

For a set of complete instructions on installing and using the CMS Tool, refer to the CMS Tool User Guide.

## 1. Determine the Hue-Saturation-Value (HSV) Intensities of the Map Colours

- Step 1. Delete the entire contents of the `legend` folder.
- Step 2. Create colour swatch files for each map colour or category and save them to the `legend` folder.
- This can be done by cropping colours from a legend strip (as shown in Figure 43) or by cropping small sections of a map to create a separate image file for each color.



Figure 43: Map legend (left) and colour swatch files (right)

- Step 3. Modify lines 55-61 of `source/legend_colours.py` to match the new colour swatches. There should be a function call to `printColours` for each colour swatch in the `legend` folder. The first argument of `printColours` should be the image file name (e.g. `mature timber.png`) and the second argument should be the colour name (e.g. `green`), as shown in Figure 44.

```

55 printColours("mature timber.png", "green")
56 printColours("immature timber.png", "dark yellow")
57 printColours("not restocked.png", "light yellow")
58 printColours("non-commercial stands.png", "dark blue")
59 printColours("non-forested land.png", "off-white")
60 printColours("water.png", "light blue")
61 printColours("boundary.png", "dark grey")

```

Figure 44: Code to be modified in `legend_colours.py`



- Step 4. Change directory to the `choropleth-map-segmentation` folder.
- Step 5. Activate the environment by running `conda activate drhmai`.
- Step 6. Run the command `python source/legend_colours.py` to analyze the colour swatches. A text file called `legendColours.txt` will be created with colour analysis data and stored in the `legend` folder. An example of the analysis output is shown in Figure 45.

```

1  mature timber.png
2      colour = green
3      HSV = [41 167 143]
4      RGB = [107 143 49]
5      GRAY = 99
6
7  immature timber.png
8      colour = dark yellow
9      HSV = [26 217 206]
10     RGB = [206 187 30]
11     GRAY = 141
12
13  not restocked.png
14     colour = light yellow
15     HSV = [25 229 237]
16     RGB = [237 202 23]
17     GRAY = 154
18
19  non-commercial stands.png
20     colour = dark blue
21     HSV = [89 92 154]
22     RGB = [98 153 151]
23     GRAY = 134
24
25  non-forested land.png
26     colour = off-white
27     HSV = [18 51 240]
28     RGB = [240 221 191]
29     GRAY = 217
30
31  water.png
32     colour = light blue
33     HSV = [27 34 203]
34     RGB = [203 200 175]
35     GRAY = 192
36
37  boundary.png
38     colour = dark grey
39     HSV = [16 52 77]
40     RGB = [77 70 61]
41     GRAY = 69

```

Figure 45: Example output appearing in `legendColours.txt`

## 2. Modify the CMS Tool

This section will outline a two-stage process:

- A. The first stage, Adjust the Colour Category Variables, deals with adjusting the colour category variables to allow the code to work with the new map colour categories
- B. The second stage, Adjust the HSV Ranges, deals with adjusting the HSV ranges to obtain an accurate segmentation result.

### A. Adjust the Colour Category Variables

Step 1. Modify lines 40-76 of `source/CMS_tool.py` (as shown in Figure 46) to match the contents of the `legend/legendColours.txt` file.

Lines 40-46 contain the HSV values, lines 48-60 contain the RGB values, lines 62-68 contain the grayscale value, and lines 70-76 contain the category names.

**Note** – The RGB variables in lines 48-60 are used to colour the segmented image outputs, and can be modified as needed to alter the appearance of the output segments. In the example shown in Figure 46, this modification was made by commenting out the legend colours from the analysis output (lines 51-53) and replacing them with new colours (lines 58-60).

```

38 # define legend pixel colour values (extracted using the legend_colours.py module)
39 # for referencing during the segmentation process
40 matureTimberHSV      = np.array([ 41, 167, 143], dtype='uint8')
41 immatureTimberHSV    = np.array([ 26, 217, 206], dtype='uint8')
42 notRestockedHSV      = np.array([ 25, 229, 237], dtype='uint8')
43 nonCommercialStandsHSV = np.array([ 89,  92, 154], dtype='uint8')
44 nonForestedLandHSV   = np.array([ 18,  51, 240], dtype='uint8')
45 waterHSV             = np.array([ 27,  34, 203], dtype='uint8')
46 boundaryHSV          = np.array([ 16,  52,  77], dtype='uint8')
47
48 matureTimberRGB      = np.array([107, 143,  49], dtype='uint8')
49 immatureTimberRGB    = np.array([206, 187,  30], dtype='uint8')
50 notRestockedRGB      = np.array([237, 202,  23], dtype='uint8')
51 # nonCommercialStandsRGB = np.array([ 98, 153, 151], dtype='uint8')
52 # nonForestedLandRGB    = np.array([240, 221, 191], dtype='uint8')
53 # waterRGB              = np.array([203, 200, 175], dtype='uint8')
54 boundaryRGB          = np.array([ 77,  70,  61], dtype='uint8')
55
56 # the colours below have not been extracted from the legend but are instead
57 # extracted from actual map regions and are more aesthetically pleasing
58 nonCommercialStandsRGB = np.array([ 64, 138, 177], dtype='uint8')
59 nonForestedLandRGB     = np.array([239, 231, 218], dtype='uint8')
60 waterRGB               = np.array([194, 211, 205], dtype='uint8')
61
62 matureTimberGray      = 99
63 immatureTimberGray    = 141
64 notRestockedGray      = 154
65 nonCommercialStandsGray = 134
66 nonForestedLandGray   = 217
67 waterGray            = 192
68 boundaryGray          = 69
69
70 matureTimberStr       = 'mature timber'
71 immatureTimberStr     = 'immature timber'
72 notRestockedStr       = 'not restocked'
73 nonCommercialStandsStr = 'non-commercial stands'
74 nonForestedLandStr    = 'non-forested land'
75 waterStr              = 'water'
76 boundaryStr           = 'boundary'

```

Figure 46: Colour analysis variables to be modified

- Step 2. Modify the variable names in lines 1108-1178 of `source/CMS_tool.py` so that each map category makes direct reference to the variables defined in Figure 46.

Each map category segment should have lines of code similar to those shown in Figure 47. The variables `seg1` and `mask1` are incremented by one for each map category.

**Note** – The code in lines 1108-1178 generates coloured segments (i.e. `seg1`, `seg2`, etc.) and binary masks (i.e. `mask1`, `mask2`, etc.) for each map category.

Any pixel with HSV intensities which fall outside of the map category colour bounds (e.g. `matureTimberLBound` and `matureTimberUBound`) are classified as black pixels. Consequently, the code in lines 1108-1178 does not include code for the `boundary` section as it is automatically accounted for when creating the coloured segments. Although not defined as such in the code, the dark pixels can be thought of as `seg0`.

```

1108     matureTimberLBound = (35, 91, threshold)
1109     matureTimberUBound = (82, 255, 255)
1110     seg1, mask1 = segmentByHsvRange(imageHSV,
1111                                     matureTimberLBound,
1112                                     matureTimberUBound,
1113                                     matureTimberRGB,
1114                                     matureTimberStr)
1115     printToLog(imageFolder, matureTimberStr
1116               + ' LB: {0}'.format(matureTimberLBound))
1117     printToLog(imageFolder, matureTimberStr
1118               + ' UB: {0}'.format(matureTimberUBound))

```

Figure 47: Colour segmentation variables to be modified

- Step 3. Initialize the lower bound and upper bound HSV threshold values (i.e. the first two lines of code shown in Figure 47) for each map category.

**Note** – The Hue and Saturation variables are primarily used to segment the colours whereas the Value variable is used to segment very dark or black pixels. Any pixel with a Value below a defined `threshold` variable is considered to be a dark or black pixel.

The Hue and Saturation bounds for each category colour can be initialized using the category's intensities generated from the swatch analysis (i.e. from `legend/legendColours.txt`). The bounds can be adjusted as needed.

For example, the `matureTimber` category has an HSV intensity of `(41,167,143)` and can be initialized to have a lower bound `matureTimberLBound = (31,157,threshold)` and upper bound `matureTimberUBound = (61,177,255)`.

- Step 4. Modify line 1180 so that all image segments are summed together. In this case, the six colour segments are added as shown in Figure 48.

```
1180     imageSegmented = seg1 + seg2 + seg3 + seg4 + seg5 + seg6
```

*Figure 48: imageSegmented variable to be modified*

- Step 5. Modify the `numSegments` variable in line 1185 to reflect the correct number of segments. In this case, there are six colour segments and one dark segment for a total of 7 segments, as shown in Figure 49.
- Step 6. Modify line 1186 so that all `mask` variables are added together to create the `imageLabels` variable. Each `mask` should be multiplied by an incremented integer value, as shown in Figure 49.

```
1185     numSegments = 7 # including black segment
1186     imageLabels = mask1 + mask2*2 + mask3*3 + mask4*4 + mask5*5 + mask6*6
```

*Figure 49: numSegments and imageLabels variables to be modified*

- Step 7. Modify lines 1192-1198 so that the RGB colours defined in lines 48-60 (Figure 46) are combined into a single array, as shown in Figure 50.

**Note** – The first entry should be the `blackRGB` colour, and then each map category colour should be listed in the same order that the `seg` and `mask` variables were defined in lines 1108-1178.

```
1192     colours = np.vstack((blackRGB,
1193         matureTimberRGB,
1194         immatureTimberRGB,
1195         notRestockedRGB,
1196         nonCommercialStandsRGB,
1197         nonForestedLandRGB,
1198         waterRGB))
```

*Figure 50: colours variable to be modified*

- Step 8. Modify lines 1201-1207 so that the map category strings defined in lines 70-76 (Figure 46) are combined into a single list, as shown in Figure 51.

```
1201     classNames = [boundaryStr,
1202         matureTimberStr,
1203         immatureTimberStr,
1204         notRestockedStr,
1205         nonCommercialStandsStr,
1206         nonForestedLandStr,
1207         waterStr]
```

*Figure 51: classNames variable to be modified*

## B. Adjust the HSV Ranges

- Step 1. Add a map with the desired map categories to the `input` folder for processing.
- Step 2. Run the command `python source/CMS_tool.py run` and use the HSV segmentation method to segment the map using the newly-defined HSV bounds.
- Step 3. Review the images `05 Binarized Image.png` and `Pixel Intensity Histogram - Image Brightness.png` in the map's output `processing` folder.  
These images illustrate how the tool segments dark/black pixels. An example of successful segmentation of dark pixels is shown in Figure 52 and Figure 53.

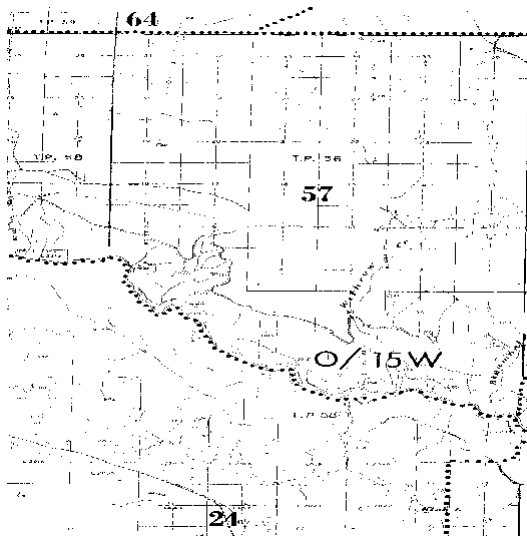


Figure 52: Example of binarized image

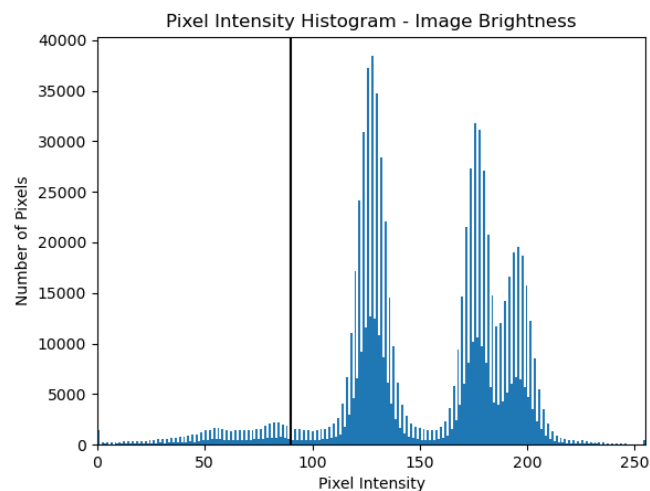


Figure 53: Example of pixel brightness histogram

- Step 4. Adjust the `threshold` variable on line 1059 (as shown in Figure 54) as needed.  
If the threshold is increased, more pixels will be categorized as dark/black pixels.  
If the threshold is decreased, fewer pixels will be categorized as dark/black pixels.

```
1059      threshold = 90
```

Figure 54: threshold variable to be modified as needed

- Step 5. Review the images `06 Segmented Image.png` and `Scatter Plot - HSV Pixel Intensities with RGB Colour Labels.png` in the map's output `processing` folder. These images illustrate how the tool segments colour pixels. An example of successful segmentation of colour pixels is shown in Figure 55.

Step 6. Adjust the HSV bounds (e.g. `matureTimberLBound` and `matureTimberUBound`) in lines 1108-1178 for each of the map categories as needed.

**Important** – Ensure that the bounds of HSV regions do not overlap with each other. If there is overlap, the segmented image will be incorrectly coloured.

**Note** – If there are large black areas in `06 Segmented Image.png` where a colour is missing, the HSV range for that colour should be expanded based on the numerical ranges of the colours observed in `Scatter Plot - HSV Pixel Intensities with RGB Colour Labels.png`.

Similarly, if there are large areas of misclassified colours, then the HSV ranges for the misclassified colours should also be adjusted.

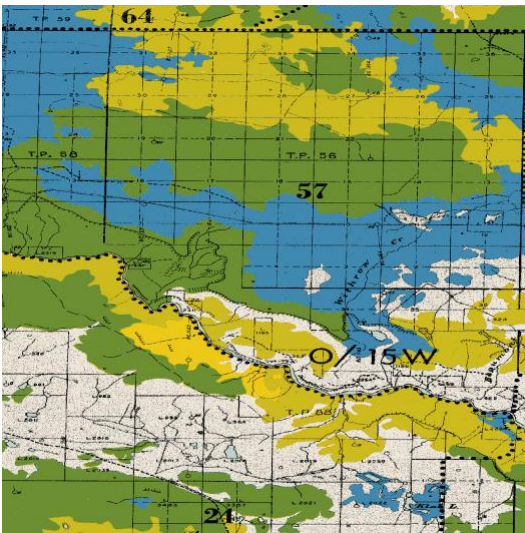


Figure 55: Example of segmented image

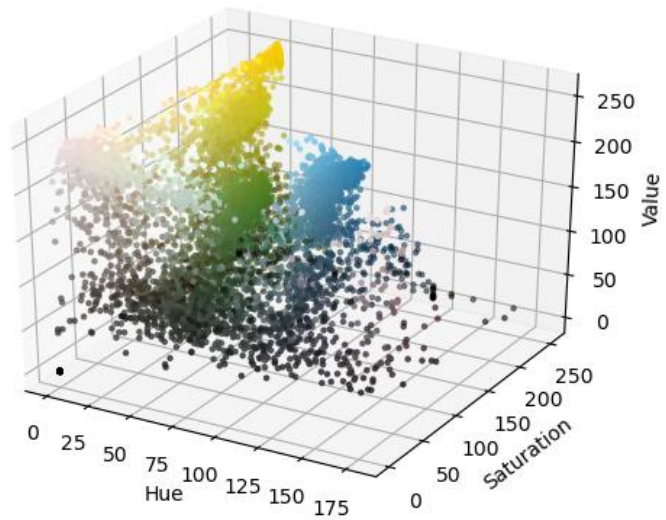


Figure 56: Example of HSV pixel intensity scatter plot

Step 7. Repeat Step 1 to Step 6 using different input maps and adjust each colour's HSV values as needed each time. This ensures that the specified HSV ranges are compatible across the dataset as there is likely some variation in category colours from image to image.

As an example, the final adjusted HSV bounds of the map category colours shown in Figure 56 are presented in Table 2.

*Table 2: Comparison of initial HSV intensity with final adjusted HSV bounds for all colour categories*

<b>Forest Cover Category</b>	<b>Initial HSV Intensity</b>	<b>Final Adjusted HSV Lower Bound</b>	<b>Final Adjusted HSV Upper Bound</b>
Mature timber	(41, 167, 143)	(35, 91, 90)	(82, 255, 255)
Immature timber	(26, 217, 206)	(22, 101, 90)	(34, 255, 220)
Not restocked	(25, 229, 237)	(22, 101, 225)	(34, 255, 255)
Non-commercial stands	(89, 92, 154)	(83, 101, 90)	(120, 255, 255)
Non-forested lands	(18, 51, 240)	(10, 0, 90)	(28, 95, 255)
Water	(27, 34, 203)	(30, 0, 90)	(90, 90, 255)