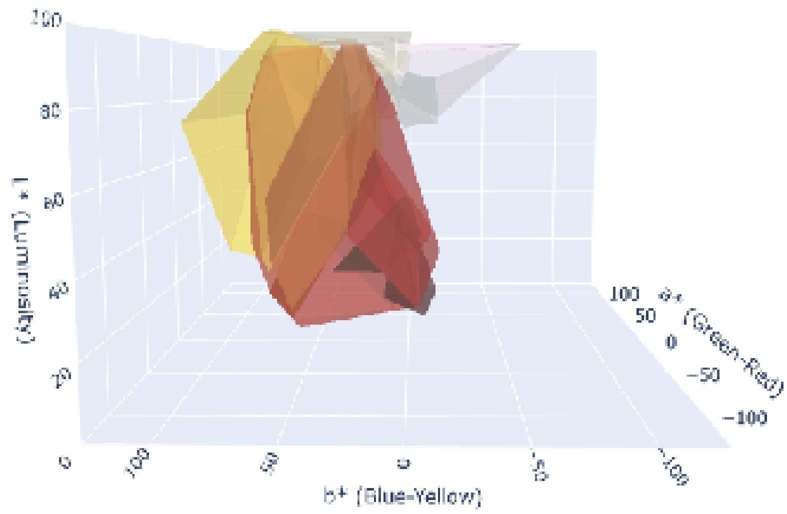
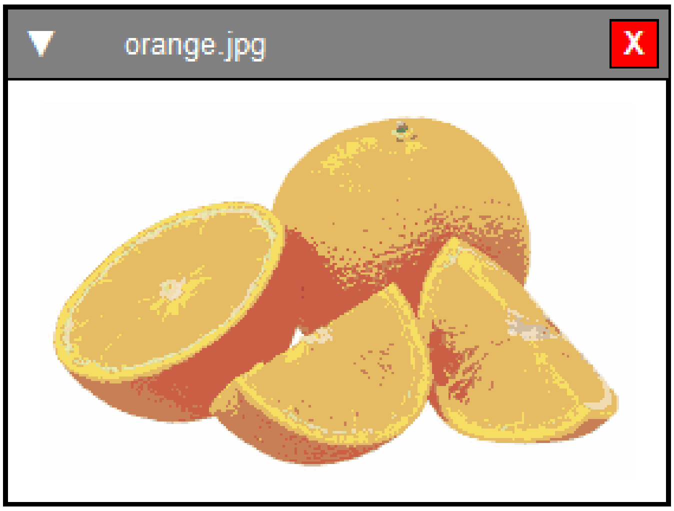
PyFCS GUI: User Manual and Technical Guide

Rafael Vázquez Conejo

May 22, 2025





Contents

1 Introduction

PyFCS GUI is a graphical user interface developed as an extension of the open-source PyFCS library, designed for the creation and analysis of fuzzy color spaces. This GUI enhances usability and accessibility, enabling the generation of fuzzy color spaces from palettes or images, along with interactive 3D visualization and advanced color mapping tools.

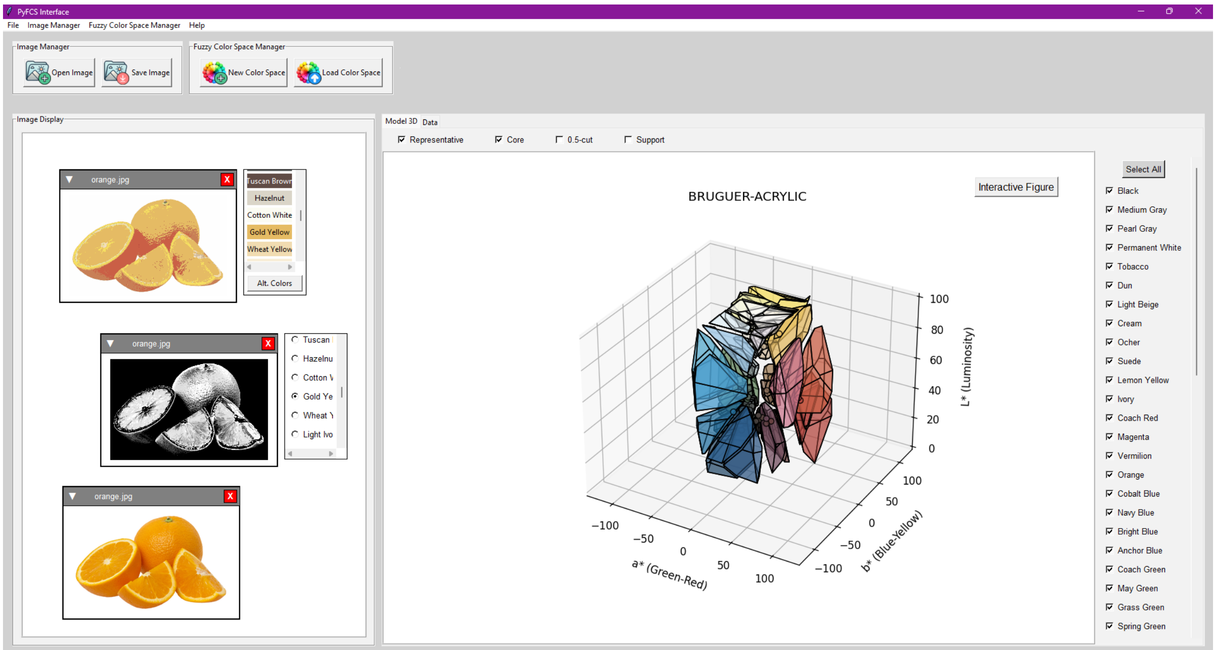


Figure 1: Main interface of PyFCS GUI.

Through parametric controls, it is possible to explore various levels of generalization and specificity in color representation. The generated color spaces can be exported in reusable formats to support reproducibility.

This extension facilitates the practical application of fuzzy color models, providing a versatile tool suitable for scientific research as well as artistic or perceptual analysis.

1.1 System Requirements

The following requirements must be met to ensure proper operation of the application:

- A runtime environment for executing Python scripts.
- Python version 3.9 or higher (version 3.10 recommended).
- The `pip` package manager installed and accessible from the command line.
- Operating system: currently available for Windows only.

2 Installation

The project is publicly available on GitHub at:

`https://github.com/RafaelConejo/PyFCS`

2.1 Installation Steps on Windows

Currently, this tool is available exclusively for Windows systems. Before proceeding, ensure that a Python environment is properly configured. This can be a standard Python installation (version 3.9 or higher is required; 3.10 is recommended) or a virtual environment manager such as Anaconda.

Important: The commands `python` and `pip` must be accessible from the command line. If not, ensure that Python is correctly added to your system's PATH.

Note: If the `pip` command is not recognized, it may not be installed or not included in your system's PATH. You can install it by running the following command:

```
python -m ensurepip --upgrade
```

Alternatively, make sure that Python and pip are correctly installed and accessible from the command line. For more details, refer to: `https://pip.pypa.io/en/stable/installation/`

If no modifications to the source code are needed, follow these steps for a quick installation:

1. Access the project repository on GitHub and download the library using the **"Clone or Download"** option, or from the releases section by downloading the `.zip` file.

2. Extract the contents of the `.zip` file to a preferred local folder.

3. Open a terminal (CMD or PowerShell), navigate to the project's root directory, and run the following command to install the required dependencies:

   ```
   pip install -r PyFCS\external\requirements.txt
   ```

4. Once the dependencies are installed, the main structure of the interface can be launched by executing:

   ```
   python PyFCS\visualization\basic_structure.py
   ```

3 Theoretical Foundations

3.1 Fuzzy Colors and Fuzzy Color Spaces

A *fuzzy color* is a flexible categorization of color that captures the subjective and imprecise nature of human color perception. Mathematically, a fuzzy color is defined within a color space (such as RGB or CIELAB [1]) where each color is associated with a membership degree that quantifies how representative it is of a particular fuzzy category.

Formally, given a color space C, a fuzzy color $\widetilde{C}$ is defined by a membership function $\mu_{\widetilde{C}} : C \to [0, 1]$. A value close to 1 indicates a strong representation of the fuzzy color, while a value near 0 implies low representativeness.

A *fuzzy color space* is a collection of such fuzzy colors, each represented by its own membership function [2]. It provides a conceptual framework where color categories can have overlapping boundaries, reflecting the way humans perceive color. Formally, we define a fuzzy color space as $\widetilde{\Gamma} = \{\widetilde{C}_1, \ldots, \widetilde{C}_m\}$, where each fuzzy color $\widetilde{C}_i$ is defined on a crisp color space Γ.

3.2 Methodology for Developing Fuzzy Color Spaces

This subsection presents the methodology used for representing color terms as fuzzy colors. It combines the theory of conceptual spaces [2, 3] with specialized fuzzification techniques as described in [4]. While the approach can construct a complete fuzzy color space $\widetilde{\Gamma} = \{\widetilde{C}_1, \ldots, \widetilde{C}_m\}$, it does so by modeling each fuzzy color $\widetilde{C}_i$ individually.

Conceptual spaces typically define distinct, crisp categories. However, due to the fuzzy nature of color language, this methodology defines fuzzy boundaries by determining multiple α-cuts through a sequence of increasingly inclusive volumes $\mathcal{V}_{\widetilde{C}_i}$ in the color space. These volumes are derived by scaling a central volume, representing the crisp category, to generate different levels of inclusion. According to the fuzzy set representation theorem, a fuzzy set can be reconstructed from its α-cuts. However, it is not practical to define all $\alpha \in (0, 1]$, so our approach defines key cuts (e.g., core and support) and interpolates the remaining membership values.

The fuzzification methodology from [4] follows these steps:

- **Prototype selection:** Identify a representative crisp color $\mathbf{r}^i$ (positive prototype) for the color term C_i, and a set R_i^- of negative prototypes that are distinct from C_i.
- **Voronoi tessellation:** Apply a Voronoi partition on the crisp color space Γ using $\mathbf{r}^i$ and the negative prototypes. The Voronoi cell V^i associated with $\mathbf{r}^i$ represents the 0.5-cut of the fuzzy color $\widetilde{C}_i$.
- **Core and support scaling:** Create the core volume V_1^i and support volume V_q^i by scaling V^i around $\mathbf{r}^i$ using two scale factors: $\lambda \in [0, 1]$ to shrink for the core, and $\lambda' \in [1, 2]$ to expand for the support. These ensure separation from the cores of negative prototypes.
- **Membership interpolation:** Construct the membership function of $\widetilde{C}_i$ by linearly interpolating between the boundary surfaces of V_1^i, V^i, and V_q^i.

This method enables the software to represent human-like color categories with fuzzy boundaries and is used internally to classify, visualize, and interpret fuzzy color inputs.

4 PyFCS: Design and Features

Developed in Python, **PyFCS** is an open-source library available on GitHub[1]. It provides a versatile and user-friendly framework for constructing and manipulating fuzzy color spaces. Its modular architecture, designed with Python's broad adoption and scientific ecosystem in mind, ensures high accessibility and seamless integration with tools for scientific computing and image processing.

4.1 General Design and Modules

The architecture of PyFCS is carefully modular, with key components working together to support the construction and analysis of fuzzy color spaces. The main modules include:

- **Color Space Module:** Handles different color representations and conversions, essential for ensuring accurate processing across RGB, HSV, LAB, and other standards.
- **Geometry Module:** Manages geometric constructs such as points, vectors, planes, and faces. These are foundational for building and manipulating volumes in color modeling.
- **Fuzzy Logic Module:** Contains core fuzzy logic operations and algorithms for handling fuzzy color modeling and membership function computation.
- **Visualization Module:** Provides visualization tools for fuzzy colors and spaces, including support, core, and α-cut boundaries, improving interpretability.
- **Input/Output Module:** Supports file import/export in various formats to facilitate interoperability and persistence.
- **External Libraries Module:** Integrates external packages such as `skimage.color` and `scipy.spatial.Voronoi`, extending PyFCS capabilities.
- **Test Module:** Includes example scripts and test cases for validating and demonstrating the functionality of each module.

4.2 Core Classes and Responsibilities

PyFCS defines several key classes, each implementing specific functionality within the fuzzy color modeling pipeline:

- **ColorSpace.py:** Manages color space conversions (e.g., RGB, HSV, CIELAB), ensuring consistent and accurate color representation.
- **Point.py:** Represents points in a multidimensional color space; the fundamental building block for all geometric entities.
- **Vector.py:** Builds on `Point.py` to define vectors, providing direction and magnitude for geometric operations.

[1] `https://www.github.com/RafaelConejo/PyFCS`

- **Plane.py:** Constructs planes from vectors to represent boundaries and spatial partitions in color space.
- **Face.py:** Assembles faces from planes, forming surface elements of volumes used in fuzzy color definitions.
- **Volume.py:** Manages the definition and manipulation of volumes (core, support, α-cuts), integrating with `scipy.spatial.Voronoi` for region generation.
- **FuzzyColor.py:** Defines fuzzy colors via their volume representations. Includes a label and membership function for proximity-based membership computation.
- **FuzzyColorSpace.py:** Orchestrates multiple `FuzzyColor` instances to construct and manage a full fuzzy color space.
- **Prototype.py:** Stores and manages color prototypes (positive and negative), critical for Voronoi-based fuzzification.
- **MembershipFunction.py:** Performs interpolation between α-cut surfaces to compute fuzzy membership degrees based on spatial relationships.
- **Input.py:** Facilitates parsing and adapting input data formats (e.g., `.fcs`, `.cns`) into PyFCS-compatible objects.

4.3 External Integration and Extensibility

PyFCS is built to integrate with well-established libraries like `skimage.color`, which ensures accurate color transformations, and `scipy.spatial`, which powers robust Voronoi tessellation and geometric computation. This enables a reliable and extensible platform suitable for both academic research and practical applications in fuzzy logic, color science, and human-centered computing.

4.4 Connection to PyFCS GUI

The **PyFCS GUI** is a graphical interface built on top of this library. All GUI functionalities—such as defining prototypes, visualizing color categories, and exporting fuzzy color spaces—are powered directly by PyFCS. Understanding the core PyFCS design is essential for developers or advanced users who wish to extend, debug, or interact with the system beyond the graphical layer.

5 Basic Usage

Upon launching the application, three main modules are available to create, visualize, and apply fuzzy color spaces. Their key functionalities are outlined below:

5.1 Fuzzy Color Space Manager

This module allows for the creation, loading, and management of fuzzy color spaces using two primary methods:

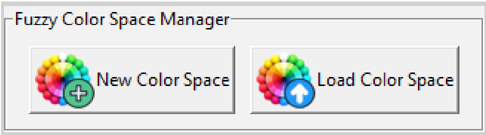


Figure 2: Fuzzy Color Space Manager module interface, showing available actions for creating and loading color spaces.

5.1.1 Creating New Color Spaces

The `New Color Space` button initiates the construction of a new fuzzy color space, which can be done through two main approaches:

- **Palette-based creation**: Colors can be entered in CIELAB space, labeled with linguistic tags, and used to automatically generate the corresponding fuzzy sets (Figure 3).

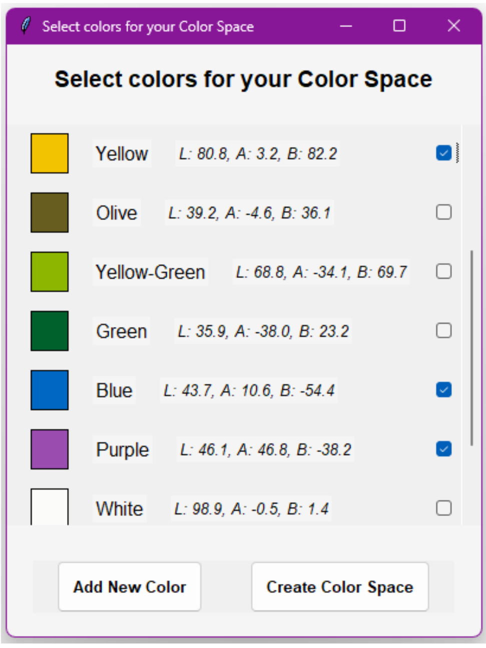


Figure 3: Palette-based creation interface.

Colors can be added using the `Add New Color` button. LAB values can be entered manually or selected visually using the `Browse Color` option, which opens a color

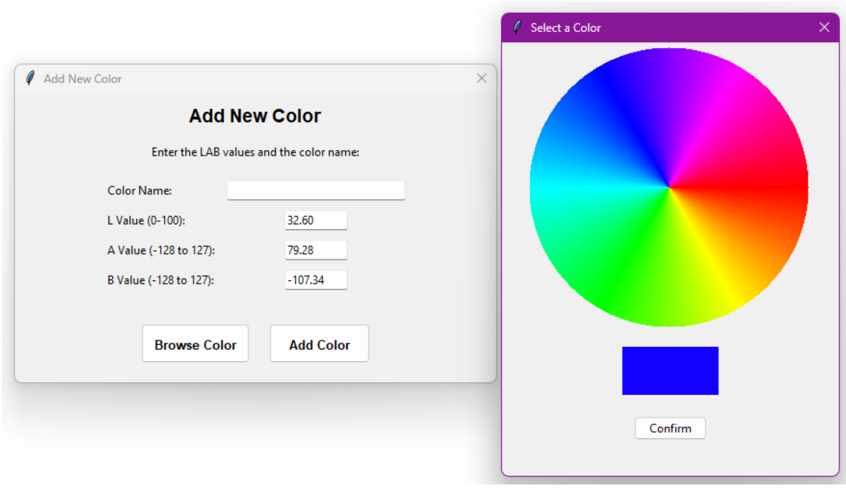


Figure 4: Color input interface showing the fields to define new color and linguistic label.

wheel (Figure 4). Each color must be named and confirmed using the `Add Color` button.

After adding at least two colors (minimum required to create a space), a dialog appears to name the fuzzy color space. The space is saved in the `PYFCS/fuzzy_color_spaces` directory with a `.fcs` extension.

- **Image-based creation**: A fuzzy color space can also be generated from dominant colors extracted from one or more images using the DBSCAN clustering algorithm [5] (Figure 5).

  The threshold parameter (between 0 and 1) controls the generalization level. Lower values result in fewer, more generalized colors, while higher values yield more detailed segmentation. The `Recalculate` button must be pressed after modifying the threshold.

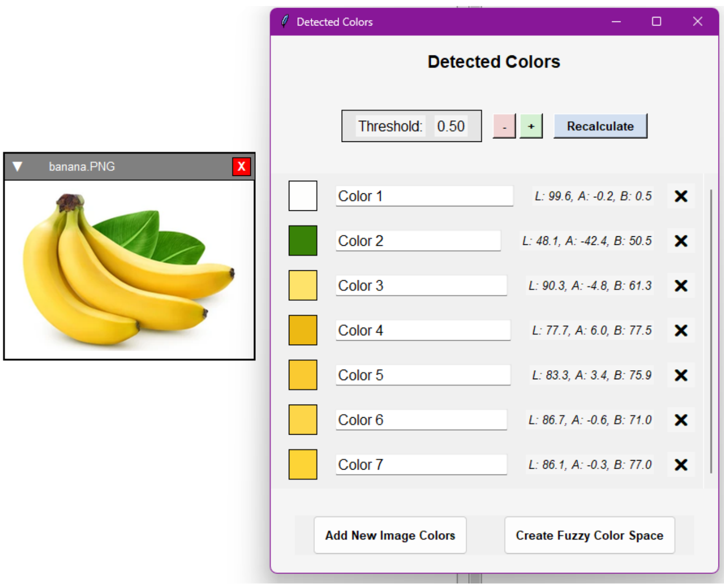


Figure 5: Example of color extraction from an image using DBSCAN.

At least one image must be loaded using the Image Manager Module for this function to operate. The `Add New Image Colors` button allows the inclusion of additional colors from other loaded images.

Before finalizing the color space, detected color names can be edited, and undesired ones can be removed. The `Create Fuzzy Color Space` button triggers the naming dialog and saves the file in `PYFCS/fuzzy_color_spaces` with a `.fcs` extension.

5.1.2 Loading Color Spaces

The `Load Color Space` button allows for importing previously saved fuzzy color spaces. Files with `.fcs` and `.cns` extensions are supported. By default, the file browser opens in the `PYFCS/fuzzy_color_spaces` directory.

5.2 Fuzzy Color Space Visualization Module

This module offers interactive tools for visually inspecting and editing the structure of fuzzy color spaces:

- **3D Visualization**: Fuzzy colors are displayed in the CIELAB space using a 3D representation, showing crisp representatives, cores, α-cuts (e.g., $\alpha = 0.5$), and support regions (Figure 6).

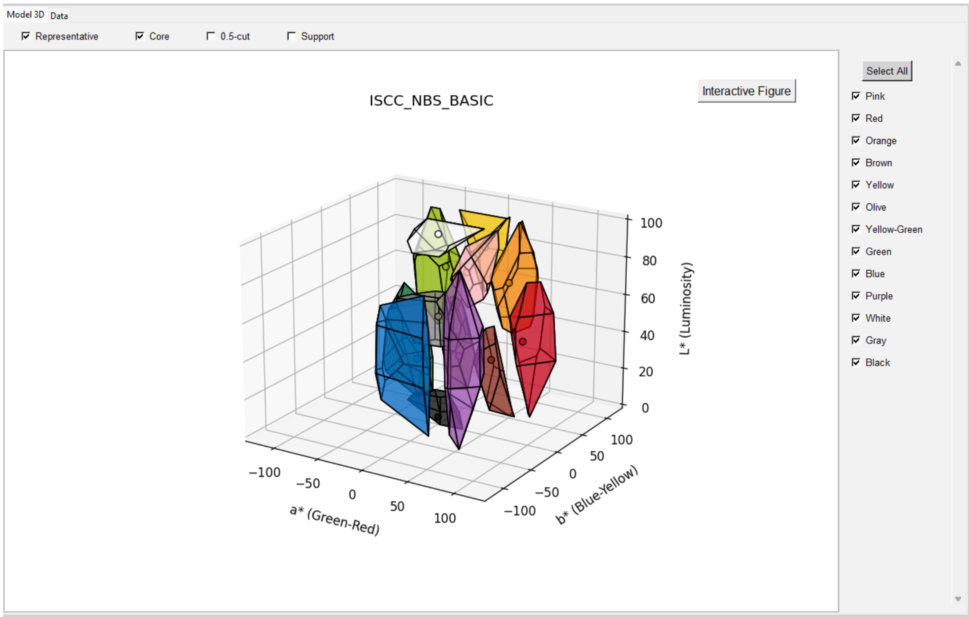


Figure 6: 3D visualization of a Fuzzy Color Space core regions.

Colors can be selected individually or all at once using the `Select All` button. The `Interactive Figure` button opens an auxiliary window with enhanced features such as zoom, pan, screenshot, and free rotation (Figure 7).

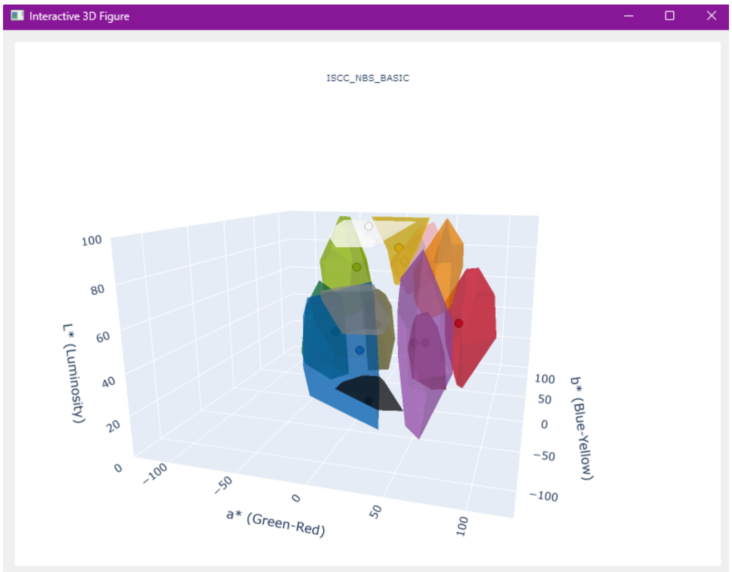
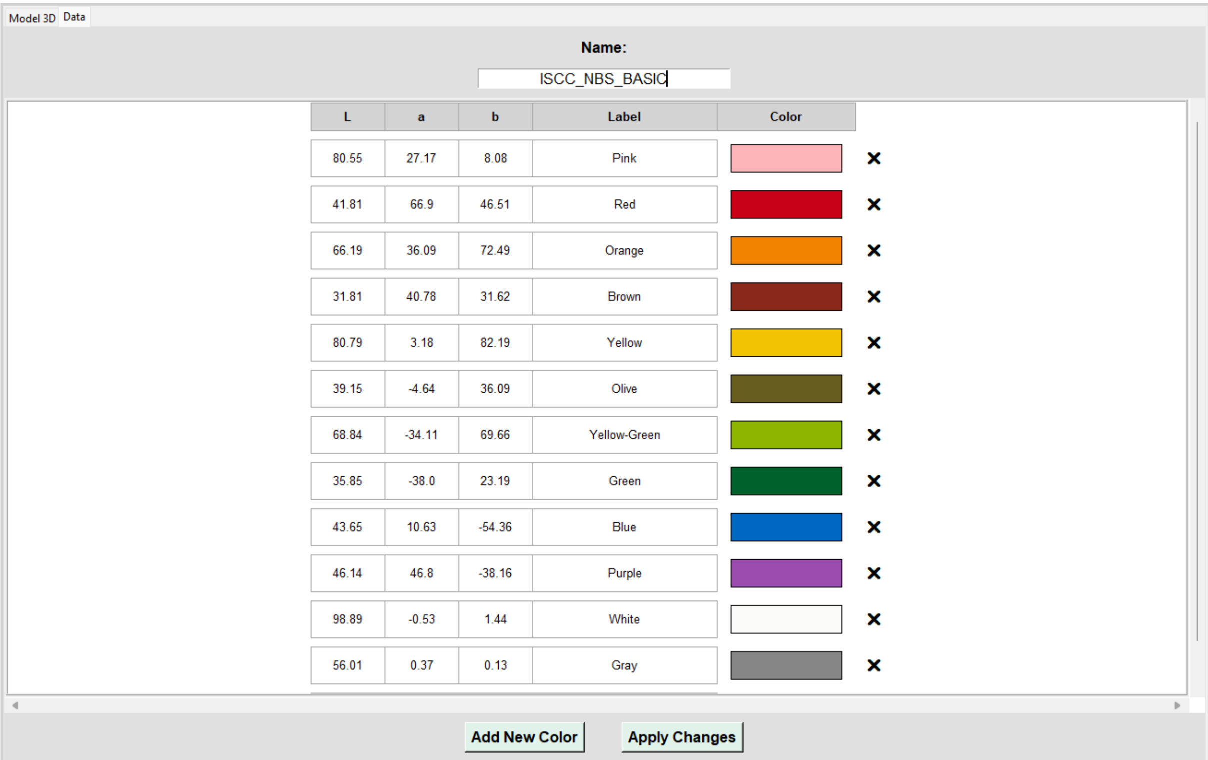


Figure 7: Interactive 3D view with navigation and inspection tools.

- **Data view and editing**: Alongside the graphical representation, each fuzzy color's linguistic label and crisp representative are shown, following the conceptual spaces framework [4] (Figure 8).



L	a	b	Label	Color
80.55	27.17	8.08	Pink	
41.81	66.9	46.51	Red	
66.19	36.09	72.49	Orange	
31.81	40.78	31.62	Brown	
80.79	3.18	82.19	Yellow	
39.15	-4.64	36.09	Olive	
68.84	-34.11	69.66	Yellow-Green	
35.85	-38.0	23.19	Green	
43.65	10.63	-54.36	Blue	
46.14	46.8	-38.16	Purple	
98.89	-0.53	1.44	White	
56.01	0.37	0.13	Gray	

Figure 8: Detail view of fuzzy color Data window.

The space can be edited—new colors added with `Add New Color` or existing ones deleted. Changes must be confirmed with `Apply Changes` to update both the file and visualization.

5.3 Image Manager Module

This module enables the application of fuzzy color spaces to image analysis and supports two main operations:

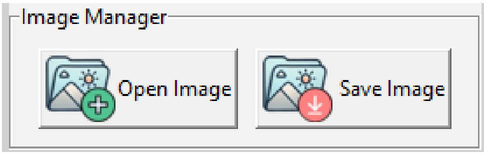


Figure 9: Image Manager module, showing the panel for loading and saving images.

The `Open Image` button allows loading images (`jpg`, `jpeg`, `png`, `bmp`). Multiple images can be opened and moved freely within the interface. Images can be closed via the red `X` in their top-right corner (Figure 10).

Figure 10: Loaded image with UI controls for closing and applying fuzzy color mappings.

By clicking the arrow on the left edge of each image frame, color space application options become available (after loading a fuzzy color space):

- **Color Mapping**: Displays the degree of membership of a specific fuzzy color across the image (Figure 11).

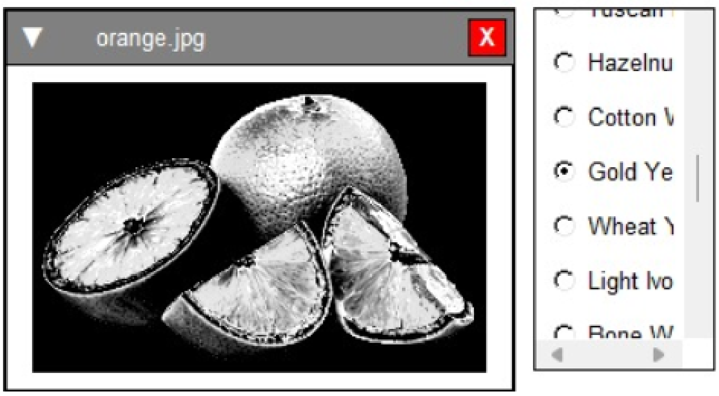


Figure 11: Fuzzy color mapping showing pixel-wise degrees of membership to a selected color of the *bruguer acrylic* color space.

- **Color Mapping All**: Reconstructs the entire image by assigning each pixel the fuzzy color with the highest membership. Two palettes are available: a custom-generated one and the original palette (toggle via `Alt. Colors`) (Figure 12).

Figure 12: Full image color mapping using *bruguer acrylic* colors space and alternate color palettes.

- **Original View**: Restores the original image for comparison or further processing.

If any modifications are made, the `Save Image` button enables exporting the processed image to a user-selected location.

5.4 Typical Workflow

The following is a representative workflow demonstrating how the system is used to analyze image colors with fuzzy color spaces:

1. **Load Image**: Open the *Image Manager Module* and use `Open Image` to select the image for analysis.
2. **Create Color Space**: In the *Fuzzy Color Space Manager*, click `New Color Space` and choose the image-based method. Use DBSCAN to extract dominant colors, adjusting the threshold as needed. Add more images with `Add New Image Colors` if desired.
3. **Edit Colors**: Review and optionally rename or remove colors. Click `Create Fuzzy Color Space` to name and save the space.
4. **Visualize**: Load the saved space in the *Visualization Module* using `Load Color Space`. Explore its structure in 3D with `Interactive Figure`.
5. **Analyze Image**: Return to the *Image Manager Module* and apply:
   - **Color Mapping** to view how a selected fuzzy color maps onto the image.
   - **Color Mapping All** to segment the image by fuzzy color membership.
6. **Save Results**: Use `Save Image` to export the processed image.

This workflow enables a complete analysis pipeline from raw image input to perceptually structured representation using fuzzy color spaces.

6 Contact and Support

- For technical support, please contact: `rafaconejo@ugr.es`

References

[1] R. Kuehni, Color Space and Its Divisions: Color Order from Antiquity to the Present, Wiley, 2003.

[2] P. Gärdenfors, Conceptual Spaces: The Geometry of Thought, A Bradford book, MIT Press, 2004.

[3] P. Gärdenfors, The Geometry of Meaning: Semantics Based on Conceptual Spaces, MIT Press, 2014.

[4] J. Chamorro-Martínez, J. Soto-Hidalgo, P. Martínez-Jiménez, D. Sánchez, Fuzzy color spaces: A conceptual approach to color vision, IEEE Transactions on Fuzzy Systems (2016).

[5] D. Deng, DBSCAN clustering algorithm based on density, in: 2020 7th International Forum on Electrical Engineering and Automation (IFEEA), 2020, pp. 949–953. `doi:10.1109/IFEEA51475.2020.00199`.