Cartoonizer Project

# What is the Cartoonizer?

The Cartoonizer is a Python program that transforms real-world photographs into cartoon-style images. It mimics the artistic style of cartoons by:

1. Simplifying complex colors into a limited palette
2. Highlighting bold, clean outlines
3. Adding subtle sketch-like textures

# Content

[What is the Cartoonizer? 1](#_Toc202706995)

[Content 1](#_Toc202706996)

[Step by Step Explanation 3](#_Toc202706997)

[1.Import Libraries 3](#_Toc202706998)

[2.Initialization and UI Settings 3](#_Toc202706999)

[\_\_init\_\_(self, root) 3](#_Toc202707000)

[configure\_styles(self) 4](#_Toc202707001)

[create\_widgets(self) 4](#_Toc202707002)

[3.Core Image Processing 5](#_Toc202707003)

[process\_image(self) 5](#_Toc202707004)

[Edge Detection: 5](#_Toc202707005)

[Combine Two Edges 5](#_Toc202707006)

[Refine Edges 6](#_Toc202707007)

[TColor Simplification 6](#_Toc202707008)

[Smoothing & Edge Preservation 6](#_Toc202707009)

[Cartoon Effect 6](#_Toc202707010)

[Display & Save Results 7](#_Toc202707011)

[Error handle 7](#_Toc202707012)

[4. Utility Functions 7](#_Toc202707013)

[update\_k\_label(self) 7](#_Toc202707014)

[toggle\_edge\_display(self) 7](#_Toc202707015)

[update\_result\_display(self) 7](#_Toc202707016)

[save\_image(self) 8](#_Toc202707017)

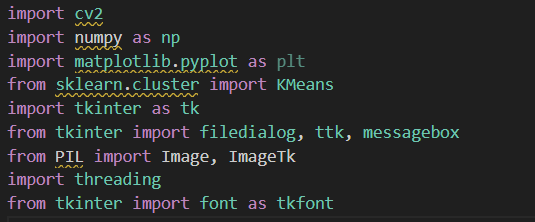
[**Overall Output** 9](#_Toc202707018)

[**Conclusion** 11](#_Toc202707019)

[**Reference** 11](#_Toc202707020)

# Step by Step Explanation

## 1.Import Libraries

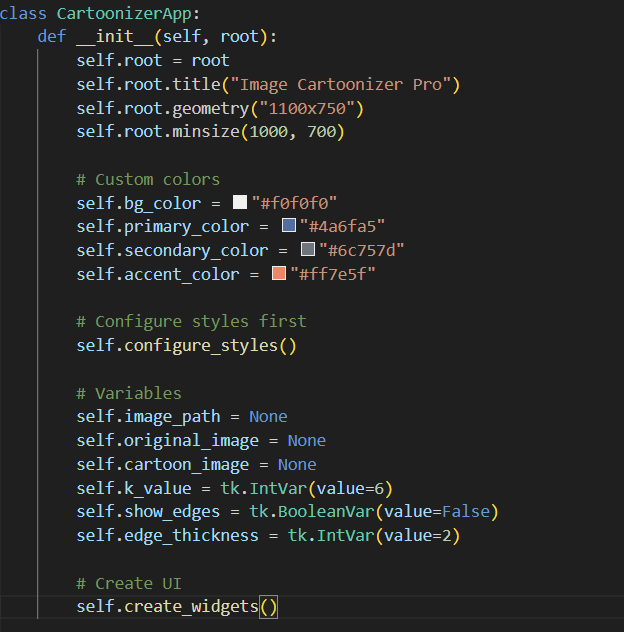


* 1. cv2 (OpenCV) – it handles the image processing tasks
  2. np (NumPy) – manages numerical operations and arrays
  3. matplotlib.pyplot – for displaying images
  4. sklearn.cluster.KMeans – reducing color complexity
  5. tkinter – let the user choose their own image file by opening a file dialog
  6. PIL.Image, PIL.ImageTk —Image processing is compatible with Tkinter
  7. Threading—Prevent the interface from freezing during image processing.

## 2.Initialization and UI Settings

#### \_\_init\_\_(self, root)

Purpose: Initializes the application window and variables.



Sets window title, dimensions, and minimum size constraints.

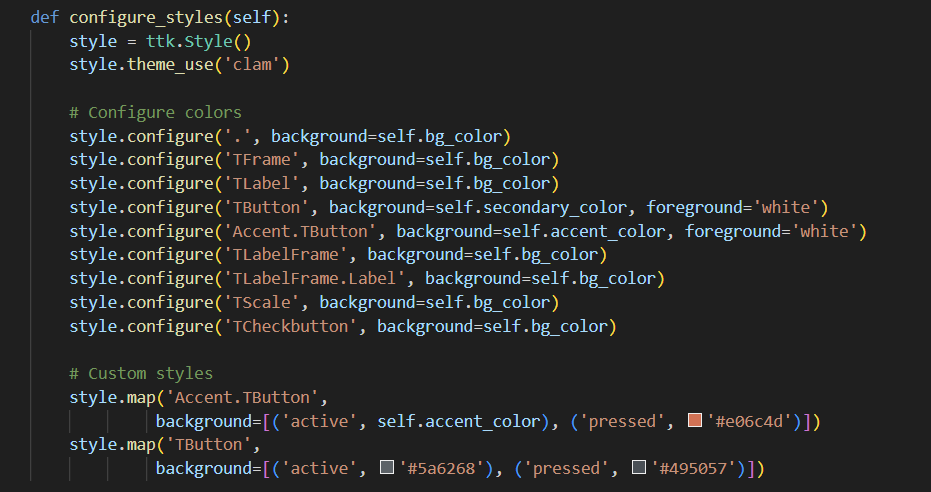
Defines color scheme (bg\_color, primary\_color, etc.).

Initializes critical variables (image\_path, k\_value, show\_edges).

Calls configure\_styles() and create\_widgets() to build the interface.

#### configure\_styles(self)

Purpose: Configures ttk widget styles.



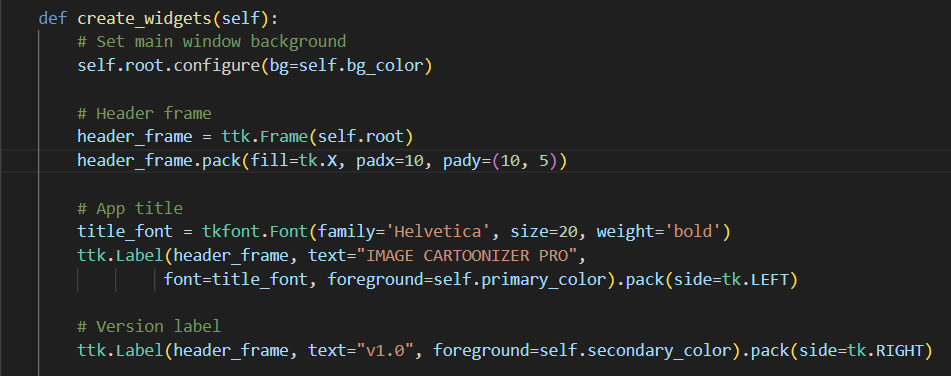
Uses the clam theme for a modern look.

Customizes colors for buttons, labels, and frames.

Defines Accent.TButton for prominent action buttons (e.g., "Cartoonize!").

#### create\_widgets(self)

Purpose: Constructs the GUI layout.



Left Panel (Controls):

File selection button (load\_image).

Sliders for parameters (k\_value, edge\_thickness).

Checkbox to toggle edge display (show\_edges).

Action buttons ("Cartoonize!", "Save Result").

Right Panel (Images):

Two labeled frames for original/cartoonized images.

Uses ttk.Label to display images.

Status Bar: Shows operation progress (e.g., "Processing...").

## 3.Core Image Processing

#### process\_image(self)

Purpose: Implements the cartoonization pipeline.

Edge Detection:

A screenshot of a computer program

AI-generated content may be incorrect.

The goal in this section is to highlight the outlines like a cartoon

The first line converts the image to grayscale and second line reduces the noise, the larger value, the more blur it gets.

Combine Two Edges

A computer screen shot of text

AI-generated content may be incorrect.

The 37th line detect the sharp edges and 38th merge both edge types.

Adaptive Threshold:

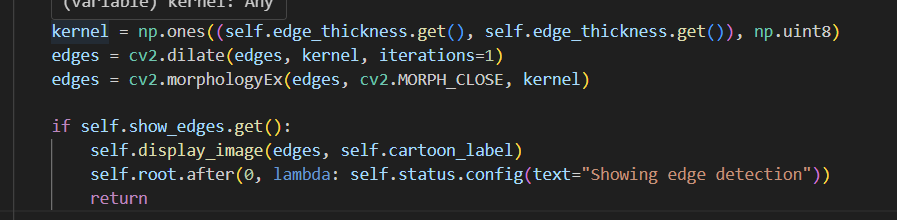
block\_size = 11 : pixel neighbourhood size for threshold calculation

c = 2 : constant subtracted from the mean

Canny Edge Detector:

minVal = 30, maxVal = 100 : weaker edges below minVal are discarded and stronger edges above maxVal are being kept

##### Refine Edges

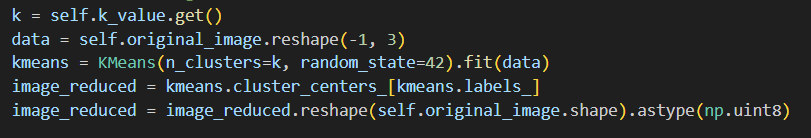
 Ensures edges are continuous and visually prominent in the final cartoon effect.

User-adjustable via the edge\_thickness slider (default: 2px).

Allows users to preview and debug edge detection separately.

Helps adjust edge-related parameters (edge\_thickness) interactively.

##### TColor Simplification

Flatten color region using k-means clustering..

Reshape image into 2D array of pixels (each row is a pixel's RGB).

Perform k-means clustering to reduce color palette.

Reconstruct image using cluster centers.

##### Smoothing & Edge Preservation

 Apply bilateral filter to smooth while preserving edges.

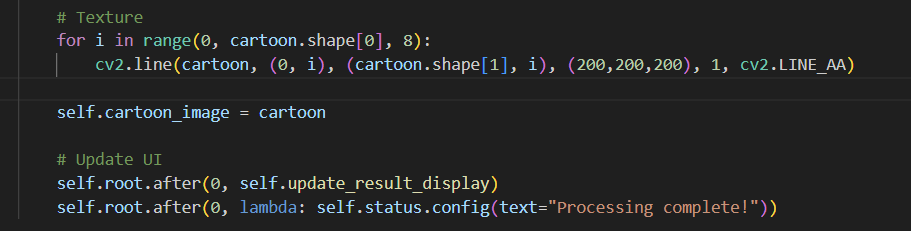
##### Cartoon Effect



Cartoon Styling.

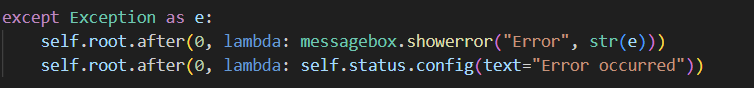
Apply black edges to the smoothed image.

##### Display & Save Results

 Draw light gray horizontal lines every 8 pixels.

Store final result.

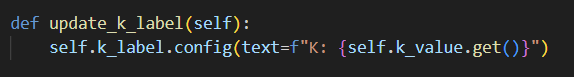
##### Error handle



## 4. Utility Functions

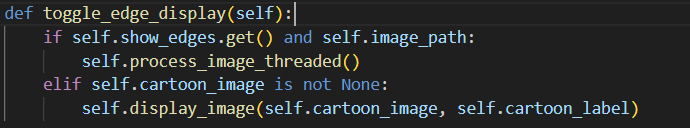
#### update\_k\_label(self)

Purpose: Updates the K-value slider label in real time.



#### toggle\_edge\_display(self)

Purpose: Toggles between edge-only and full cartoonized views.

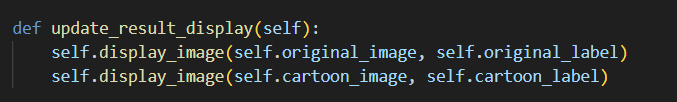


Controlled by the show\_edges checkbox.

Calls process\_image\_threaded to regenerate the display.

#### update\_result\_display(self)

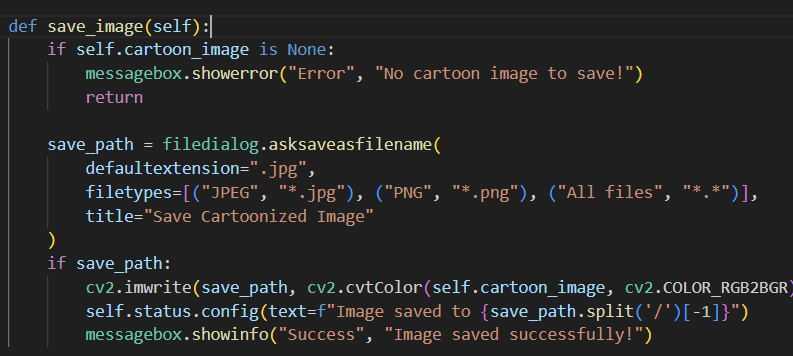
Purpose: Refreshes both image displays.



Ensures the original and processed images are synced.

#### save\_image(self)

Purpose : Saves the cartoonized image to a user-specified file location.



Ensures users can export results without technical hurdles.

Maintains OpenCV’s BGR convention while internally using RGB for display.

# **Overall Output**

A screenshot of a computer

AI-generated content may be incorrect.

**Step 1: Select Your Photo**

Click “Select Image” to open a file dialog, choose the picture you want to cartoonize, and load it. You’ll see the original image appear on the left.

A screenshot of a person smiling

AI-generated content may be incorrect.

**Step 2: Adjust Color Regions (K)**

Slide “Color Regions (K)” to pick how many color clusters the algorithm uses. Lower K gives broad flat colors; higher K preserves more detail.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 3: Set Edge Thickness**

Move “Edge Thickness” to control how bold the outlines become. Higher values yield thicker, graphic lines; lower values are subtler.

A computer screen shot

AI-generated content may be incorrect.

**Step 4: Preview Edge Detection (Optional)**

Tick “Show Edge Detection” to view just the edge map. This helps you fine-tune outline settings before blending them with colors.

A black and white text

AI-generated content may be incorrect.

**Step 5: Generate the Cartoon**

Click “Cartoonize” to run the transformation. The app applies edge detection, color clustering, and smoothing, then shows the result on the right next to your original.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 6: Save Your Result**

When you’re satisfied, click “Save Result,” choose a folder and filename in the dialog, and export the final cartoon image.

# **Conclusion**

This application demonstrates a well-structured approach to image cartoonization, balancing functionality and usability. Its modular design (e.g., separating UI, processing, and file I/O) makes it easy to extend. The next potential steps could include:

Integrating deep learning models (e.g., GANs) for more artistic styles.

Adding undo/redo functionality for parameter experimentation.

Deploying as a web/desktop app (e.g., using PyQt or Flask for broader access).

Overall, the code is production-ready for basic use cases and serves as an excellent foundation for further development in creative image processing.

# **Reference**

We referred to the Cartoonify-Image project by Abdesol on GitHub as a guide for implementing parts of our image-processing pipeline.

**Abdesol. (2019). *Cartoonify-Image* [Computer software]. GitHub.** [**https://github.com/Abdesol/Cartoonify-Image.git**](https://github.com/Abdesol/Cartoonify-Image.git)