Multi-Style Image Cartoonification Using OpenCV and KMeans Clustering

Joseph Nikhil Reddy Maramreddy
yue15@txstate.edu

Abstract

This project presents a user-interactive application that transforms standard photographs into cartoon-styled images using a variety of image processing techniques. Implemented using Python, the application leverages OpenCV, scikit-learn, and tkinter to provide a graphical user interface where users can upload images, choose from multiple cartoonifying styles, adjust parameters, and save the output. The system integrates classical computer vision techniques such as bilateral filtering, adaptive thresholding, and KMeans clustering to produce diverse artistic effects such as anime, oil painting, pencil sketch, and edge-preserving transformations. The goal of this project is to reproduce and enhance existing open-source techniques on novel image datasets in an accessible desktop application form.

1. Introduction

The process of transforming real-world photographs into stylized, cartoon-like imagery, has become increasingly popular in digital art, entertainment, and social media. These stylized images often involve simplified color regions, bold edges, and artistic abstraction that mimic hand-drawn cartoons or paintings.

This project approaches cartoonification using classical computer vision techniques, avoiding deep learning to ensure transparency, interactivity, and ease of experimentation. The goal is to build a user-friendly desktop application that lets users upload images, apply various cartoon styles, and adjust processing parameters, offering both a creative tool and a practical demonstration of how edge detection, color quantization, and smoothing can be combined for visual stylization.

2. Methodology and Image Processing Pipeline

The Cartoonify Image App employs a modular image processing pipeline to transform user-supplied photographs into cartoon-like images using classical computer vision techniques. The system is implemented in Python using OpenCV for image transformations and tkinter for a lightweight, interactive graphical user interface. It supports six distinct cartoon styles, each achieved through a tailored combination of the following core techniques:

2.1. Bilateral Filtering

Used as the primary smoothing method, bilateral filtering reduces noise and texture while preserving edges. This technique enhances the stylized look of flat color regions, mimicking the brush-stroke feel of hand-drawn art.

2.2. Color Quantization with KMeans Clustering

KMeans clustering is applied on the pixel color data to reduce the image to a fixed number of representative colors (palette reduction). This simplifies regions of color, mimicking the flat shading used in cartoons.

2.3. Edge Detection with Adaptive Thresholding

Adaptive thresholding on a grayscale, blurred version of the image helps extract sharp edge maps. These edges are used to highlight object boundaries and structure in the final cartoon.

2.4. Image Fusion

Once simplified color regions and edge maps are generated, the two are combined using bitwise operations or multiplication to produce a cartoonified output that preserves both stylistic shading and clear outlines.

2.5. Style-specific Logic

Each cartoon style tweaks these basic techniques:

- **Enhanced:** Combines bilateral filtering, KMeans quantization, and adaptive thresholding with fine-tuned parameters.
- **Advanced Filter:** Uses aggressive color smoothing and high-contrast edge enhancement.
- **Anime Style:** Applies multiple rounds of bilateral filtering for ultra-smooth shading, emphasizing clean, bold outlines.
- **Oil Painting:** Uses cv2.xphoto.oilPainting when available, falling back to cartoon logic otherwise.
- **Pencil + Color:** Merges a pencil sketch (inverted blurred grayscale) with color-quantized regions and edge masks.
- **Edge Preserving:** Applies OpenCV's edgePreservingFilter and then quantizes the result using KMeans.

2.6. Interactive GUI and Parameter Control

The app features a tkinter-based GUI where users can:

- Upload images and view original vs. processed outputs side-by-side.
- Choose from available styles via dropdown.
- Adjust key parameters such as:
    - Number of color clusters (3–20)
    - Median blur value
    - Line thickness for edge detection

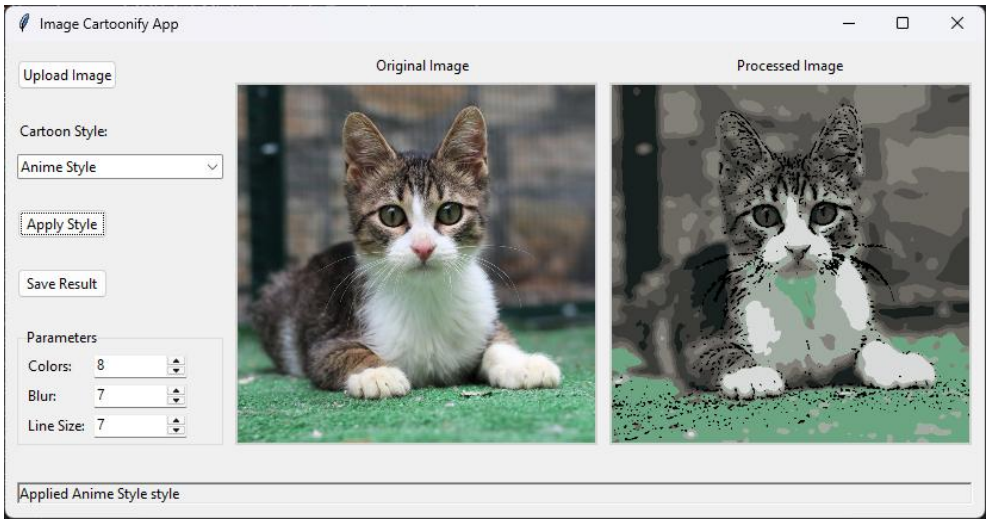

Fig 1: The GUI of the Cartoonify Image App, built using Python's tkinter.

2.7. Supporting Functionalities

- **Upload/Display/Save Image:** Uses filedialog and ImageTk for handling user images.
- **Parameter Adjustment:** Uses Spinbox and IntVar bindings to capture user-defined processing parameters.
- **Dynamic Updating:** Upon selecting a style or adjusting parameters, the image is automatically re-processed and updated in the display area.

2.8. Implementation Architecture

The entire application is encapsulated in a class named CartoonifyApp, structured around:

- UI Layer: Built with ttk.Frame, Label, and Combobox widgets.
- Image I/O Handlers: Enable upload and save via filedialog.
- Dynamic Update Mechanism: Automatically triggers processing when styles or parameters change.
- Robust Fallbacks: The app defaults to internal methods if specific OpenCV modules (like xphoto) are unavailable, ensuring broad compatibility.

3. Findings and Observations

Key observations from style and parameter experiments: \

3.1. Visual Quality

- The Anime Style method produced the cleanest and most visually striking results for portraits and close-ups, thanks to aggressive smoothing and edge retention.
- Enhanced and Advanced Filter worked well across a broad range of images, offering good edge clarity and balanced color abstraction.

- The Oil Painting effect, when supported, gave a textured and artistic look resembling brush strokes, although fallback results (using standard quantization) were still aesthetically pleasing.

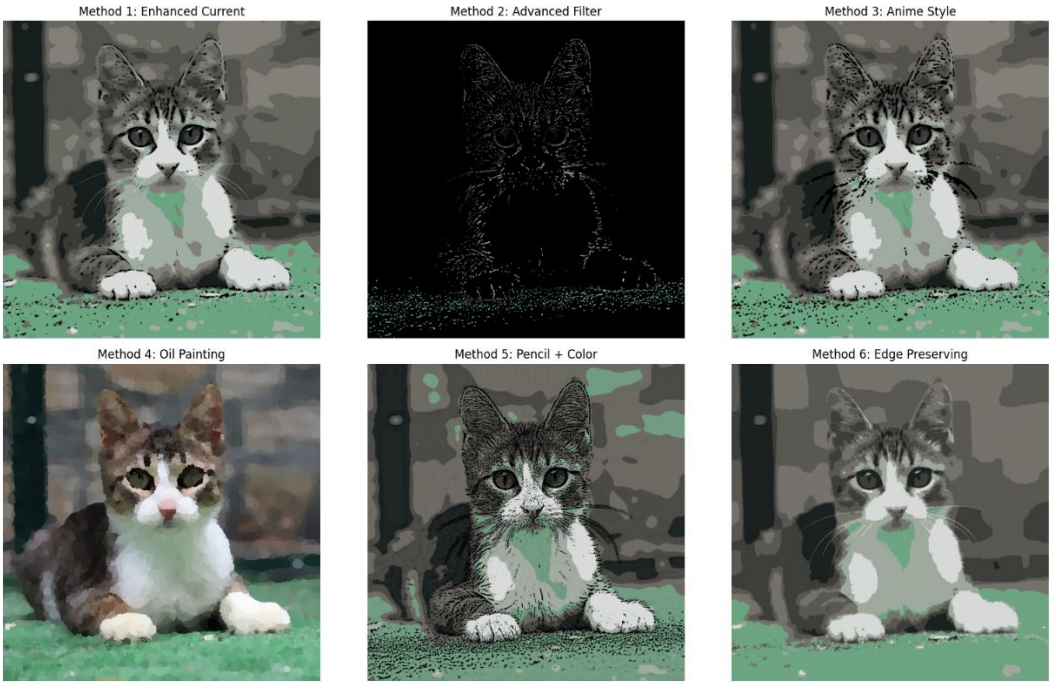

Fig 2: A collage showcasing different cartoon styles applied to a single input image.

3.2. Responsiveness

- The app processed and updated images quickly for resolutions under 1080p. KMeans quantization was the most computationally expensive part, but performance remained acceptable even with 20 clusters.
- Increasing blur and line size parameters directly affected edge detection results, allowing users to control the sharpness and intensity of outlines.

3.3. Parameter Sensitivity

- The number of color clusters (num_colors) had a significant visual impact. Lower values produced more abstract, posterized effects; higher values retained more photo-realistic features.
- Blur and line size parameters influenced the balance between detail preservation and stylization. A moderate blur (e.g., 7) with sharp edge detection gave the most cartoon-like results.

3.4. Usability

- The GUI was intuitive and stable across test sessions. Users could easily load, stylize, and save images without requiring technical knowledge.

- The real-time update feature helped in immediate visual feedback while adjusting parameters or switching styles.

3.5. Limitations

- Since no GPU acceleration or parallel processing was used, performance may degrade with high-resolution images (>4K).
- The app is currently limited to static images, videos are unsupported.

4. Future Work

Despite its effectiveness, there are areas where improvements could enhance the experience and extend the system's capabilities.

- **Video Support:** Extending the same stylization techniques to video frames to produce animated cartoon videos.
- **Batch Processing:** Adding a feature to process multiple images in one go.
- **Performance Optimization:** Implementing multiprocessing or GPU acceleration for faster processing on large images.
- **Deep Learning Integration:** Exploring pre-trained models from HuggingFace (e.g., cartoon GANs or diffusion-based stylizers) for comparison or hybrid approaches.
- **Custom Style Uploads:** Allowing users to define their own filters or style presets via configuration files or plugins.

5. Conclusion

This project shows that classical computer vision techniques can effectively replicate artistic cartoon styles without deep learning. By combining filtering, clustering, and thresholding within an interactive GUI, the application offers both a creative tool and an educational platform for exploring image processing workflows.

Overall, the project reinforces the value of interpretable, modular systems in creative technology and lays a strong foundation for future enhancements, including video support, performance scaling, and potential integration with learning-based methods.