PAK AUSTRIA FACHHOCHSCHULE: INSTITUTE OF APPLIED SCIENCES AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEEING

Digital Image Processing Lab

COMP-342L

Project Report

School of Computing Sciences

Name: Syed Hasnain Ali

Ubaid Ullah

Reg: B22F0263AI039

B22F0761AI048

AI-Blue-22

Submitted To:

Mr. Rizwan Shah

Image Cartoonization Project Report

Introduction & Problem Statement:

Digital Image Processing (DIP) is a field that involves the manipulation of digital images using various algorithms and techniques. One interesting application of DIP is image cartoonization, which transforms regular photographs into cartoon-like images. This project implements a comprehensive image cartoonization system using classical computer vision techniques without relying on deep learning methods.

The goal of this project is to develop an application that can convert standard photographs into cartoon-style sketches by applying edge detection and bilateral filtering techniques. The application includes a graphical user interface (GUI) that allows users to adjust various parameters in real-time to customize the cartoonization effect according to their preferences.

Image cartoonization has various applications, including:

- Creating artistic renditions of photographs
- Developing visual effects for media and entertainment
- Educational purposes to demonstrate classical image processing techniques
- Preprocessing for certain computer vision tasks

Techniques Used and Justification:

This project implements several classical DIP techniques to achieve the cartoonization effect. The main techniques used are:

Bilateral Filtering:

Bilateral filtering is a non-linear, edge-preserving smoothing filter that is particularly effective for the cartoonization process. Unlike conventional filters that only consider spatial proximity, bilateral filters take into account both spatial proximity and intensity similarity. This means that the filter can smooth regions while preserving strong edges, which is essential for creating the cartoon effect.

Implementation Details:

```
def applybilateralfilter(self, img):
    """
    Apply bilateral filter to smooth the image while preserving edges.
```

```
        Args:
            img: Input image

        Returns:
            Filtered image
        """
        # Apply bilateral filter
        filtered = cv2.bilateralFilter(img, self.bilateral_filter_d,
                                       self.bilateral_sigma_color,
                                       self.bilateral_sigma_space)
        return filtered
```

Justification:

Bilateral filtering was chosen because it effectively reduces noise and texture details while preserving the strong edges that define object boundaries. This creates the smooth, flat-colored regions characteristic of cartoon images while maintaining the overall structure of the original image.

2. Edge Detection

Edge detection is a fundamental technique in image processing that identifies points in an image where brightness changes sharply. For cartoonization, we use the Canny edge detector, which is known for its ability to detect a wide range of edges in images.

Implementation Details:

```
def edge_detection(self, img):
    """
    Detect edges in the image using Canny edge detector.

        Args:
            img: Input image

        Returns:
            Edge mask
        """
        # Convert to grayscale
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
# Apply median blur to reduce noise
gray_blur = cv2.medianBlur(gray, self.line_size)

# Apply Canny edge detector
edges = cv2.Canny(gray_blur, self.edge_threshold1,
self.edge_threshold2)

# Dilate edges to make them more visible
kernel = np.ones((2, 2), np.uint8)
edges = cv2.dilate(edges, kernel, iterations=1)

# Invert edges to get black lines on white background
edges = cv2.bitwise_not(edges)

# Convert back to 3-channel image
edges = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

return edges
```

Justification:

The Canny edge detector was selected for its ability to produce clean, well-defined edges with minimal noise. The edge detection process is crucial for creating the black outlines characteristic of cartoon images. The additional steps of median blurring before edge detection and dilation after edge detection help to create more consistent and prominent edges.

3. Color Quantization:

Color quantization reduces the number of distinct colors in an image. This technique is essential for creating the flat, simplified color regions typical of cartoon images.

Implementation Details:

```
def color_quantization(self, img):
    """
    Reduce the number of colors in the image.
```

```
Args:
    img: Input image

Returns:
    Image with reduced colors
"""
# Convert to float32 for processing
data = np.float32(img).reshape((-1, 3))

# Define criteria for K-means
criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 20, 0.001)

# Apply K-means clustering
ret, label, center = cv2.kmeans(data, self.total_color_levels, None,
criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
center = np.uint8(center)

# Map back to original image dimensions
result = center[label.flatten()]
result = result.reshape(img.shape)

return result
```

Justification:

K-means clustering was used for color quantization because it effectively groups similar colors together, reducing the color palette while maintaining the overall color distribution of the image. This creates the simplified color regions characteristic of cartoon images.

4. Combining Techniques for Cartoonization:

The final cartoonization effect is achieved by combining the above techniques in a specific sequence:

Implementation Details:

```
def cartoonize(self, img):
    """
    Apply cartoonization effect to the image.
```

```
Args:
    img: Input image

Returns:
    Cartoonized image
"""
# Apply bilateral filter for smoothing
filtered = self.apply_bilateral_filter(img)

# Apply color quantization
color_quantized = self.color_quantization(filtered)

# Detect edges
edges = self.edge_detection(img)

# Combine edges with color quantized image
cartoon = cv2.bitwise_and(color_quantized, edges)

return cartoon
```

Justification:

This sequence of operations was chosen after experimentation to produce the most visually appealing cartoon effect. The bilateral filter first smooths the image while preserving edges, then color quantization reduces the number of colors to create flat regions, and finally, edge detection adds the characteristic black outlines. The bitwise AND operation combines the color regions with the edge mask to produce the final cartoon image.

Graphical User Interface (GUI):

A comprehensive GUI was developed using Tkinter to allow users to interact with the cartoonization process. The GUI includes:

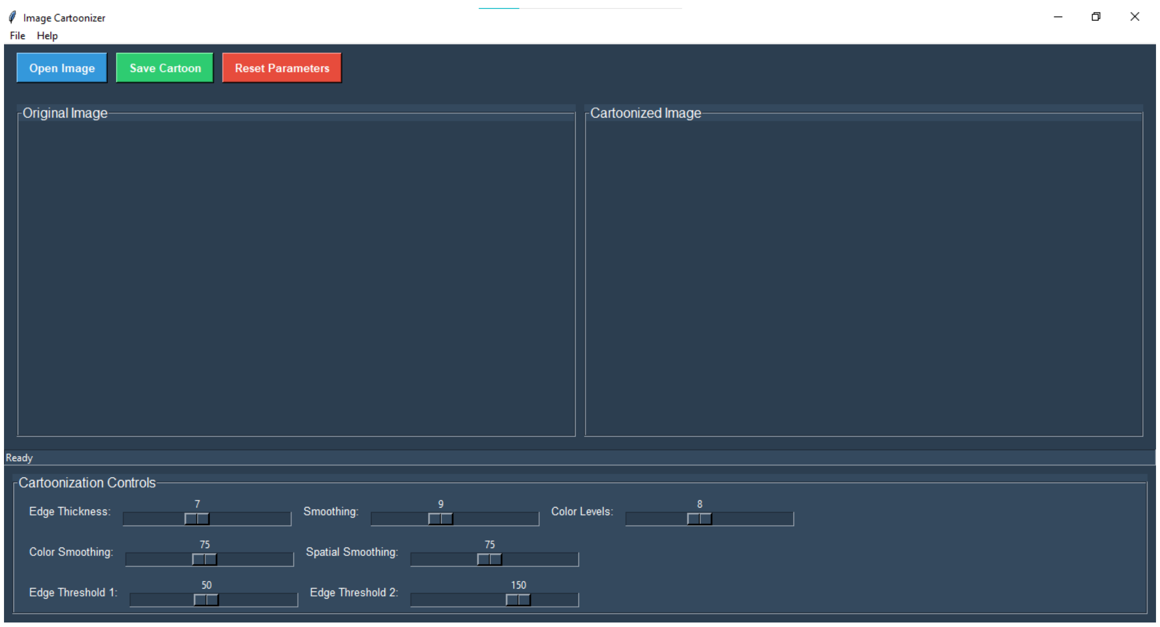
1. **Image Loading and Saving:** Users can open images from their local storage and save the cartoonized results.
2. **Parameter Adjustment:** Sliders allow users to adjust various parameters in real-time, including:

- Edge thickness
- Smoothing level
- Color smoothing

- Spatial smoothing
- Edge detection thresholds
- Number of color levels

3. **Before/After Comparison:** The GUI displays both the original and cartoonized images side by side for easy comparison.
4. **Reset Functionality:** Users can reset all parameters to their default values.

The GUI was designed with a modern, user-friendly interface using a dark color scheme for better visibility of the images.

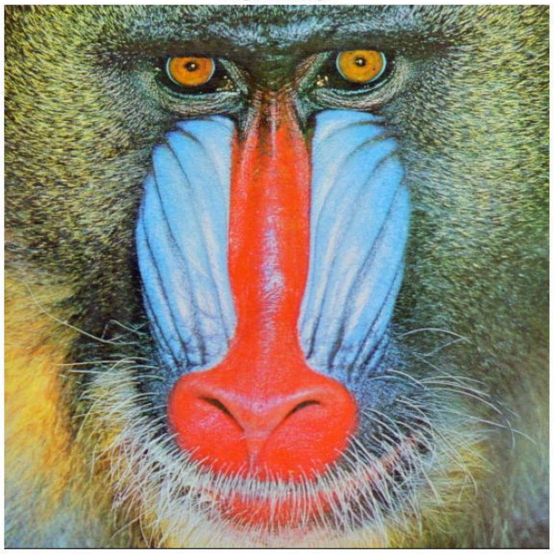


Edge Thickness: 7
Smoothing: 9
Color Levels: 8
Color Smoothing: 75
Spatial Smoothing: 75
Edge Threshold 1: 50
Edge Threshold 2: 150

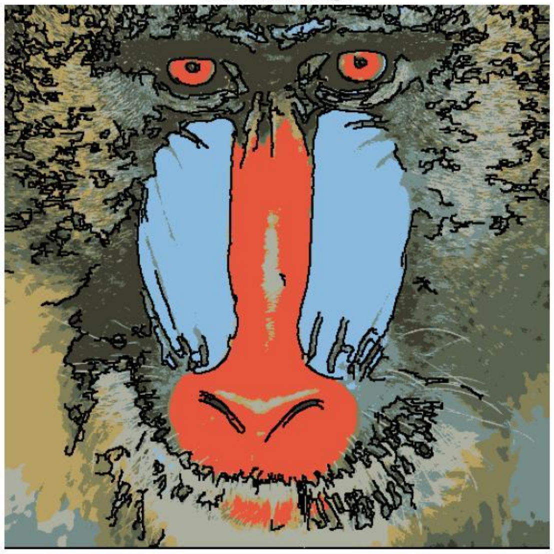
Results & Analysis:

The cartoonization algorithm was tested on a variety of images, including portraits, landscapes, city scenes, nature images, and animal photos. The results demonstrate the effectiveness of the implemented techniques in creating cartoon-like images.

Original Image

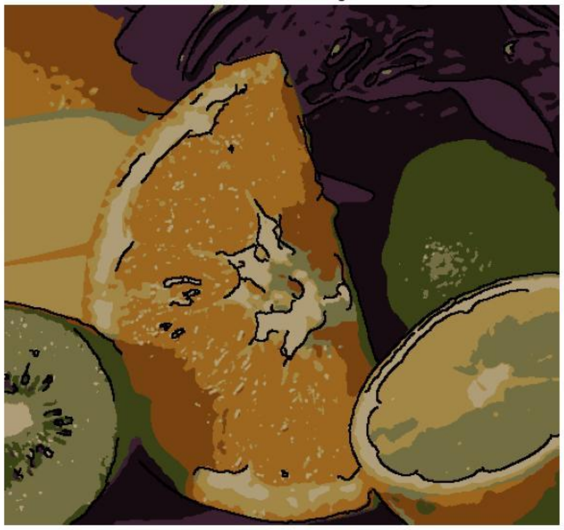


Cartoonized Image



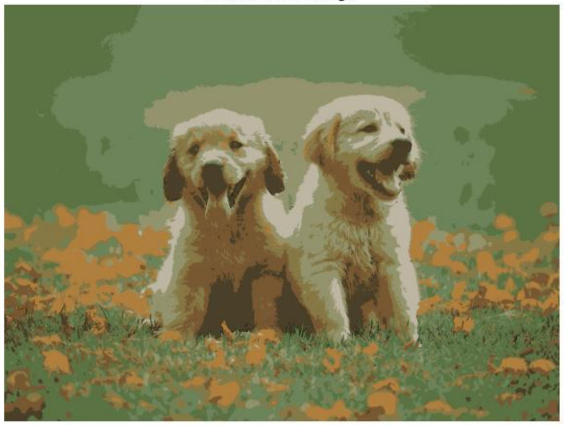
Original Image

Cartoonized Image



Original Image

Cartoonized Image



Parameter Optimization:

Various parameter combinations were tested to find the optimal settings for different types of images. The key findings include:

Bilateral Filter Parameters:

- A filter diameter (d) of 9 provides a good balance between smoothing and detail preservation.
- Sigma values around 75 for both color and space domains work well for most images.

Edge Detection Parameters:

- Thresholds of 50 and 150 for the Canny edge detector produce clean, well-defined edges for most images.
- A line size (median blur kernel size) of 7 effectively reduces noise while preserving important edges.

Color Quantization:

- 8 color levels provide a good balance between simplification and maintaining the original color distribution.

Visual Analysis:

- The cartoonization effect is particularly effective for:
- Images with clear, distinct objects and strong contrasts
- Portraits with well-defined facial features
- Landscapes with clear boundaries between elements (sky, land, water)

The effect is less effective for:

- Images with very fine details or textures
- Low-contrast images
- Images with complex patterns or backgrounds

Performance Analysis:

The cartoonization process is computationally intensive, particularly the bilateral filtering and color quantization steps. However, the implementation is efficient enough for real-time parameter adjustments on moderately sized images. For very large images, there may be a noticeable delay when adjusting parameters.

Challenges Faced & Conclusions:

Challenges:

1. **Parameter Tuning:** Finding the optimal parameters for different types of images was challenging. What works well for one image may not work as well for another. The solution was to provide user-adjustable parameters through the GUI.
2. **Edge Detection Quality:** Getting clean, continuous edges without excessive noise was difficult. This was addressed by applying median blur before edge detection and dilating the edges afterward.
3. **Balancing Simplification and Detail:** Finding the right balance between simplifying the image (for the cartoon effect) and preserving important details was challenging. This was addressed by carefully tuning the bilateral filter and color quantization parameters.
4. **GUI Responsiveness:** Ensuring the GUI remained responsive while processing images in real-time required efficient implementation of the cartoonization algorithms.

Conclusion:

This project successfully implemented an image cartoonization system using classical DIP techniques. The combination of bilateral filtering, edge detection, and color quantization effectively transforms photographs into cartoon-like images. The GUI provides an intuitive interface for users to customize the cartoonization effect according to their preferences.

The project demonstrates the power of classical computer vision techniques in creating visually appealing effects without relying on deep learning methods. It also highlights the importance of parameter tuning and the trade-offs involved in image processing.

Future improvements could include:

- Adaptive parameter selection based on image content
- Additional cartoonization styles or effects
- Batch processing capabilities for multiple images
- Performance optimizations for faster processing of large images

References:

OpenCV Documentation: https://docs.opencv.org/

Bilateral Filtering for Gray and Color Images:
https://ieeexplore.ieee.org/document/710815

Canny Edge Detection: https://ieeexplore.ieee.org/document/4767851

Color Quantization using K-Means:
https://scikitlearn.org/stable/modules/clustering.html#k-means

Tkinter Documentation: https://docs.python.org/3/library/tkinter.html