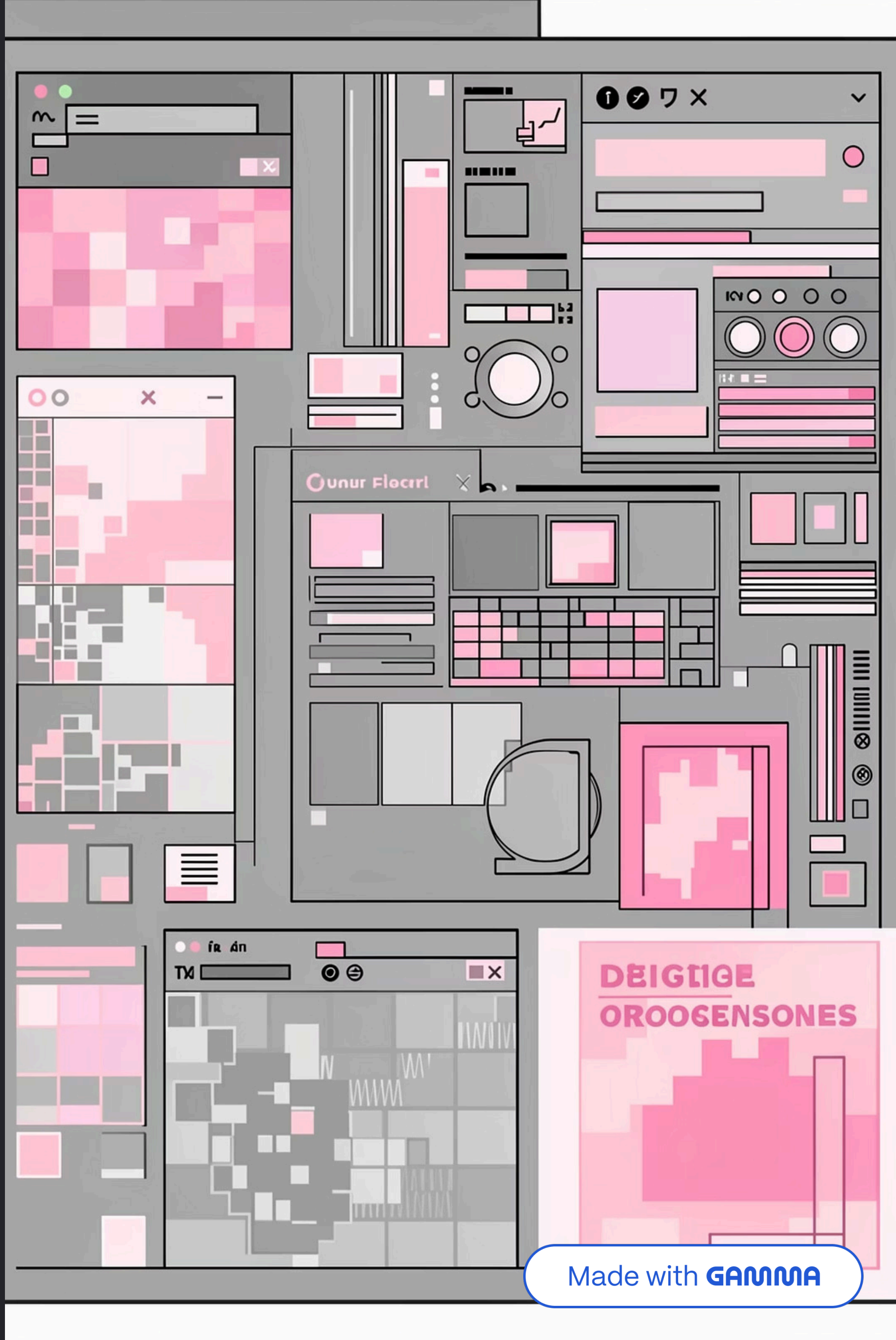
Image Filtering Using OpenCV in Python

A practical guide to computer vision fundamentals

AMINA GULL(100043)

NIMRA KANWAL(100081)



INTRODUCTION

What is Image Filtering?

Image filtering is a fundamental technique in computer vision that modifies or enhances images by applying mathematical operations to pixel values. Think of it as applying different "lenses" to see different aspects of an image.

Filters help us extract features, reduce noise, detect edges, and prepare images for further analysis in machine learning and computer vision applications.

Project Objectives

Learn OpenCV Basics

Understand how to load, display, and manipulate images using Python's most popular computer vision library.

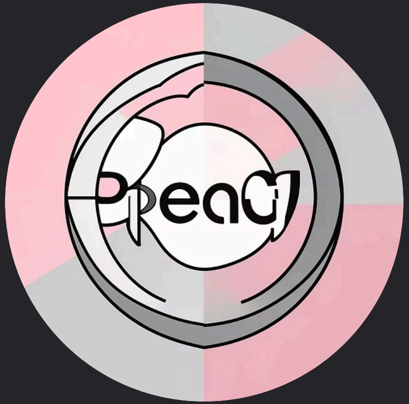
Apply Common Filters

Implement grayscale conversion, blur effects, and edge detection to see how filters transform images.

Compare Results

Analyze the differences between filtered outputs and understand when to use each technique.

Essential Libraries for Image Processing



OpenCV

The core library for computer vision tasks. Provides powerful functions for reading, processing, and analyzing images with optimized performance.

NumPy

Handles numerical operations and array manipulation. Images are stored as multi-dimensional arrays, making NumPy essential for pixel-level operations.

Matplotlib

Creates visualizations and displays images in notebooks. Perfect for comparing before-and-after results side by side.

SETUP

Installing Required Libraries

Before we begin, we need to install the necessary Python packages. In Google Colab, OpenCV and NumPy usually come pre-installed, but it's good practice to verify.

Installation Commands

```
pip install opencv-python
pip install numpy
pip install matplotlib
```

After installation, import the libraries at the top of your notebook to make their functions available throughout your code.

Import Statements

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

Pro tip: Always run your imports first to catch any installation issues early!

Working with Images in Google Colab

01

Upload Your Image

Click the folder icon in the left sidebar, then click the upload button to select an image from your computer.

02

Locate the File Path

Right-click on the uploaded file and select "Copy path" to get the exact location of your image.

03

Read the Image

Use cv2.imread() with your file path to load the image into memory as a NumPy array.

04

Display and Verify

Use Matplotlib to display the image and confirm it loaded correctly before applying filters.

Reading and Displaying the Original Image

Sample Code

```
# Read the image
img = cv2.imread('your_image.jpg')

# Convert BGR to RGB for proper display
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Display the image
plt.figure(figsize=(8, 6))
plt.imshow(img_rgb)
plt.title('Original Image')
plt.axis('off')
plt.show()
```

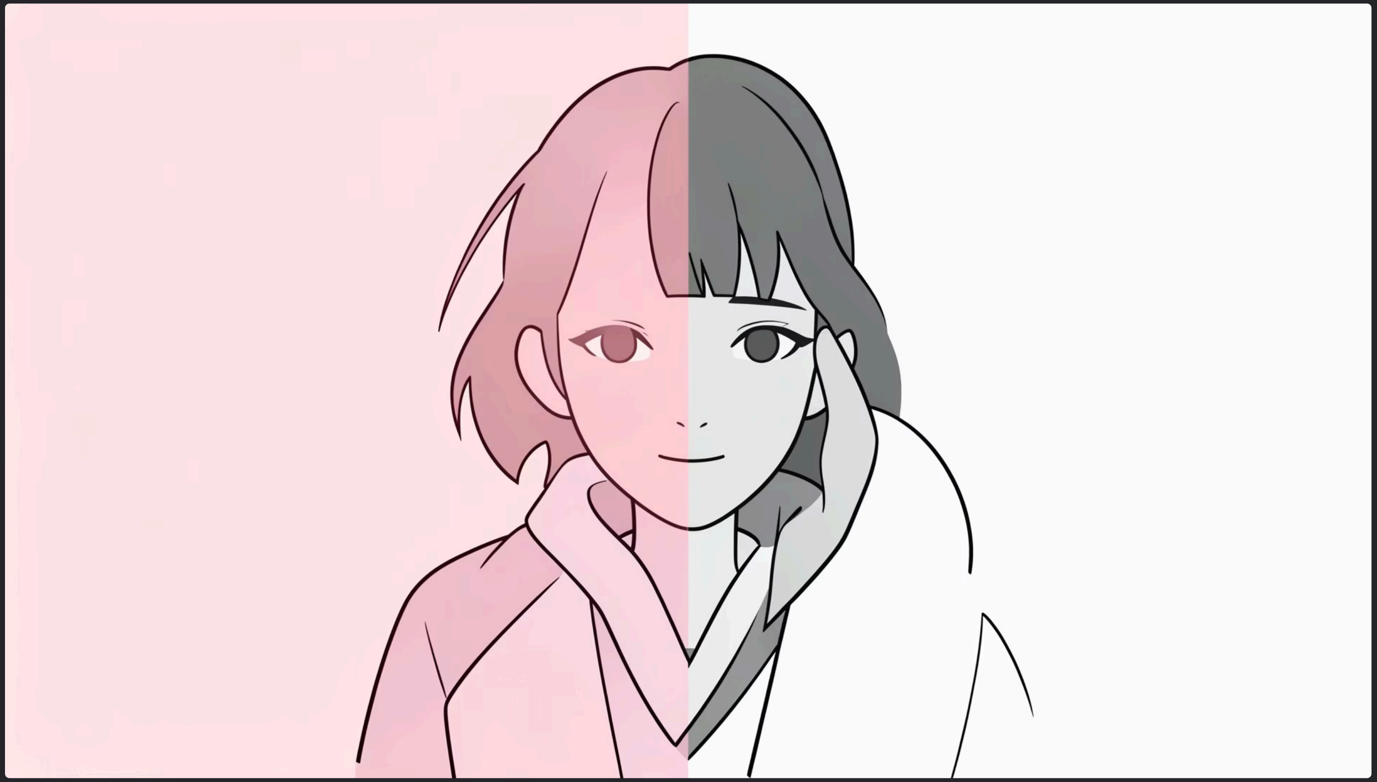
Important Note

OpenCV reads images in BGR (Blue-Green-Red) format, not RGB. Always convert to RGB before displaying with Matplotlib to see the correct colors!

The axis('off') command removes the x and y coordinate labels for a cleaner presentation. The figure size can be adjusted to fit your needs.

FILTER #1

Grayscale Filter



What It Does

Converts a color image to shades of gray by combining the RGB channels into a single intensity value. Each pixel goes from three color values to just one brightness value.

Why Use It?

- Simplifies image data and reduces processing time
- Many algorithms work better on grayscale images
- Essential preprocessing step for edge detection

Code Example

```
gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY)
```

FILTER #2

Gaussian Blur Filter

Original Sharp Image

Contains noise and fine details that might interfere with analysis

Apply Gaussian Kernel

Weighted average of surrounding pixels smooths the image

Smoothed Result

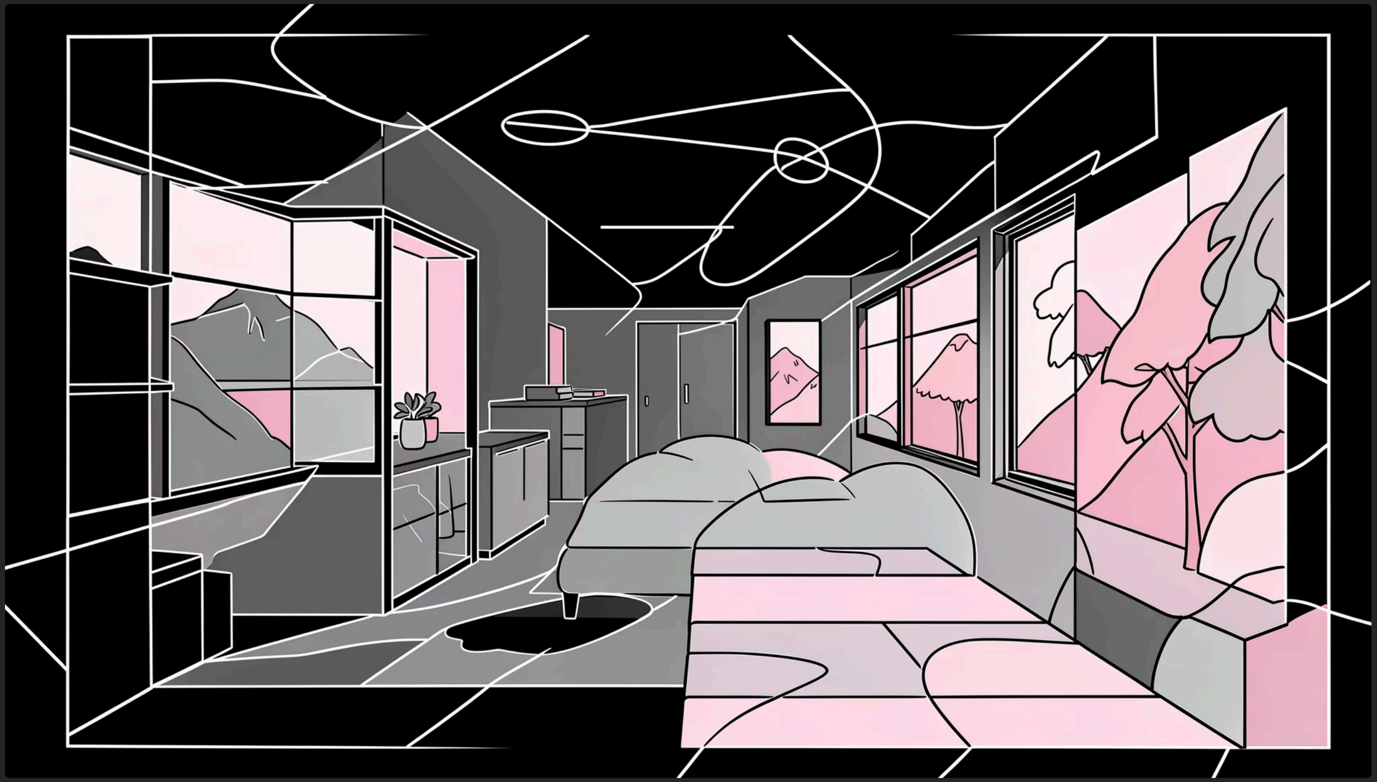
Reduced noise with preserved important features

Gaussian blur reduces image noise and detail by averaging each pixel with its neighbors using a Gaussian distribution. The kernel size (must be odd numbers like 5x5 or 7x7) controls how much blur is applied—larger kernels create stronger blur effects.

```
blurred = cv2.GaussianBlur(img, (5, 5), 0)
```

FILTER #3

Edge Detection Using Canny Algorithm



How It Works

The Canny algorithm identifies sharp changes in pixel intensity to find object boundaries. It uses gradient calculations and double thresholding to detect strong edges while filtering out noise.

Key Parameters

- **Lower threshold (100):** Minimum intensity gradient to consider
- **Upper threshold (200):** Strong edge confirmation value

```
edges = cv2.Canny(gray, 100, 200)
```

Adjusting these values changes sensitivity—lower values detect more edges but may include noise.

Summary:

- This project focuses on **image filtering using Python and OpenCV**
- An image is uploaded and displayed in its **original form**
- The image is converted to **grayscale** to remove color information
- **Gaussian blur** is applied to reduce noise and smooth the image
- **Edge detection** is used to highlight object boundaries
- **Matplotlib** is used to display and compare all images
- The project helps understand **basic image processing techniques**
- Image filtering is widely used in **computer vision applications**