Introduction:

The Image Filtering Code is crafted as a dynamic and accessible Python script, employing the OpenCV library for powerful image processing. Designed with simplicity and flexibility in mind, this code allows users to seamlessly apply various filters to images, unlocking a realm of creative possibilities. Python's readability and the extensive capabilities of OpenCV make this code an ideal platform for both novice and experienced users to delve into the fascinating world of image manipulation.

Why Python and OpenCV:

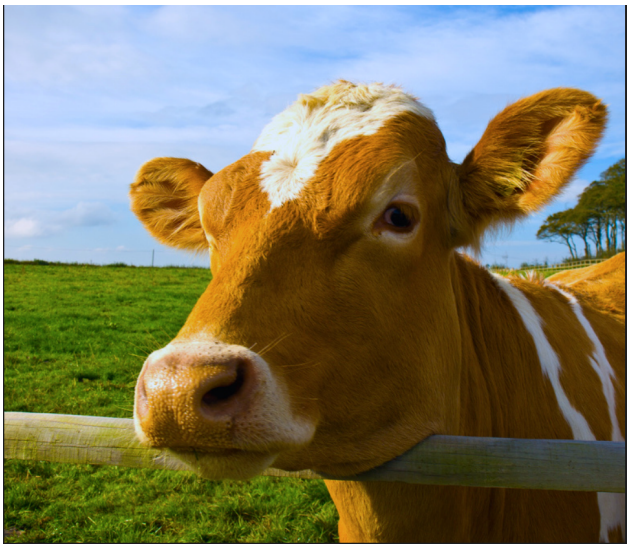
Python's ubiquity in the programming landscape, coupled with its ease of use, positions it as an optimal language for rapid development. OpenCV, a computer vision library for Python, provides a rich set of tools for image processing. The synergy between Python and OpenCV makes this code an intuitive and powerful resource for image filtering.

Understanding Filters:

In the context of image processing, filters are mathematical operations that act on pixel values, shaping the visual appearance of an image. These operations, applied through convolution or other techniques, enable the enhancement of edges, details, and overall aesthetics of an image.

Implementation Details:

This is the unfiltered image used as an example:

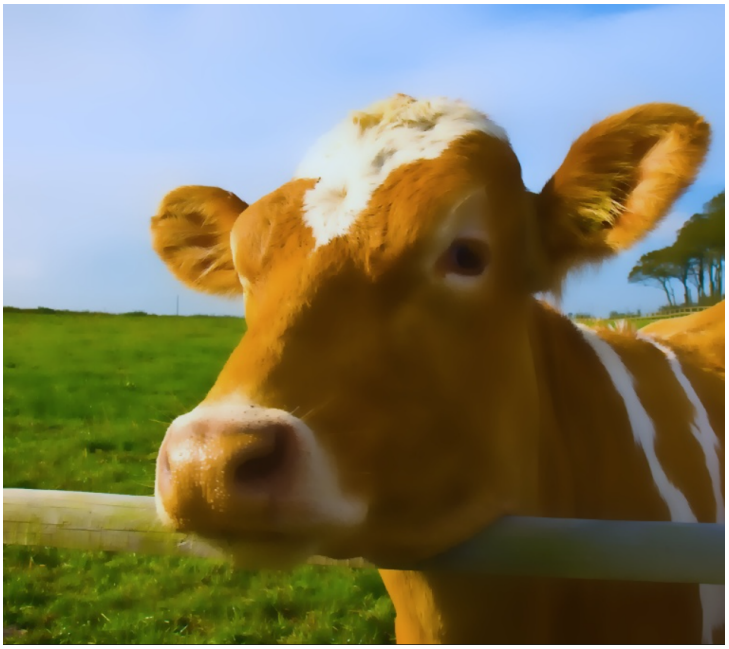


Diving into the specifics of the code, a series of functions has been meticulously crafted to encapsulate the essence of various filters:

1. Edge-Preserving Recursive Filter (`apply_edge_preserving_recursive_filter`):

- This filter enhances edges while preserving intricate details within the image.
- By employing a convolution operation with a carefully designed kernel, the filter brings forth a pronounced edge-preserving effect.
-This matrix is designed to enhance edges while preserving details. The central pixel has higher weight (5), and the surrounding pixels contribute to edge enhancement by having negative weights.

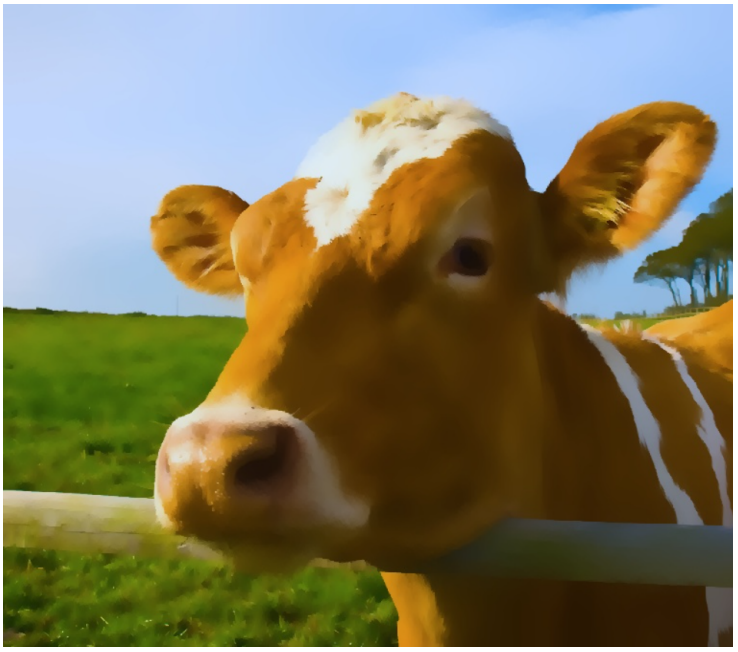
Here's applied filter:



2. Edge-Preserving Normalized Convolution Filter (`apply_edge_preserving_normalized_convolution_filter`):

- Designed to accentuate edges and maintain image structure, this filter employs normalized convolution.
- The convolution operation, orchestrated with a specific kernel, imparts a distinct emphasis on edges while preserving the overall image.
-Similar to the recursive filter, but with a different configuration. It uses normalized convolution, emphasizing the central pixel more strongly (weight 9) and maintaining the structure of the image.

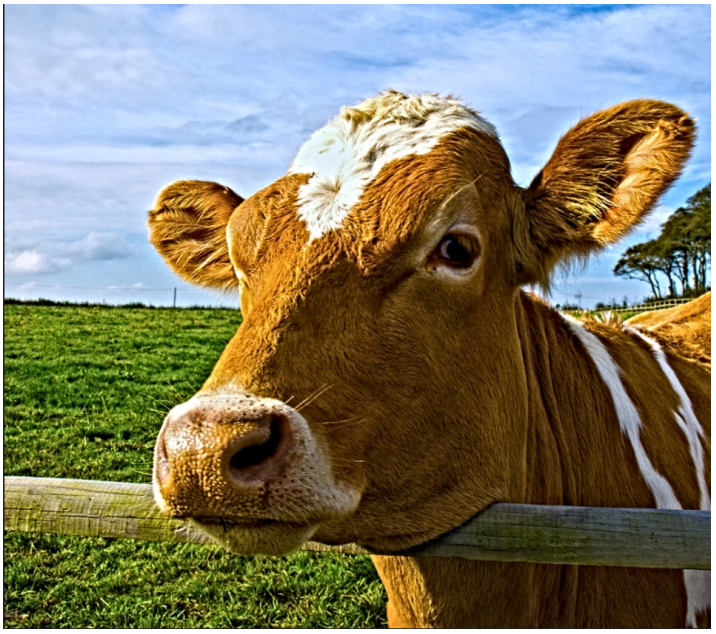
Here’s applied filter:



3. Detail Enhance Filter (`apply_detail_enhance_filter`):

- Tailored to boost fine details, this filter carefully enhances details while upholding the image's structural integrity.
- The filter achieves its goal by combining the original image with a version that has undergone detail enhancement.
- This matrix is applied to enhance fine details in an image. It contributes to sharpening and highlighting intricate features while preserving the overall structure.

Here’s applied filter:



4. Pencil Sketch Filter (`apply_pencil_sketch_filter`):

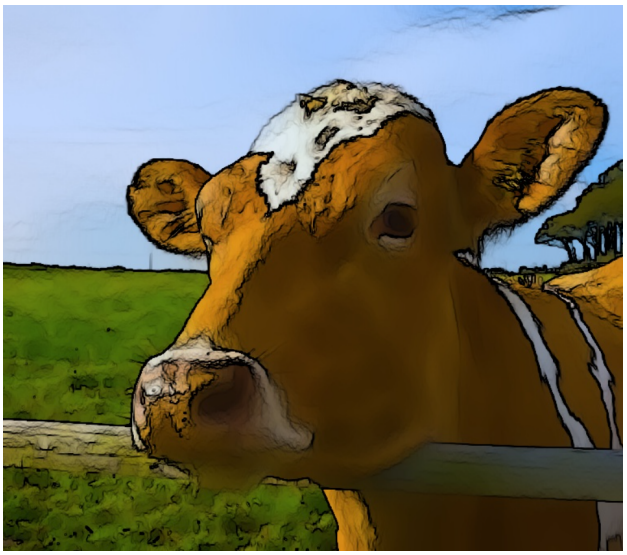
- This filter creates both grayscale and color pencil sketch effects.
- The process involves converting the image to grayscale, inverting it, and skillfully blending it with the original image.

Here's applied filter:

5. Stylization Filter (`apply_stylization_filter`):

- The stylization filter introduces an artistic flair, transforming the image into a stylized masterpiece.
- By employing a weighted blending technique, the filter harmoniously combines the original image with a stylized version.
- Similar to the detail enhance filter, it employs a kernel that enhances certain features. The stylization filter aims to create an artistic effect, and this kernel contributes to the visual style.

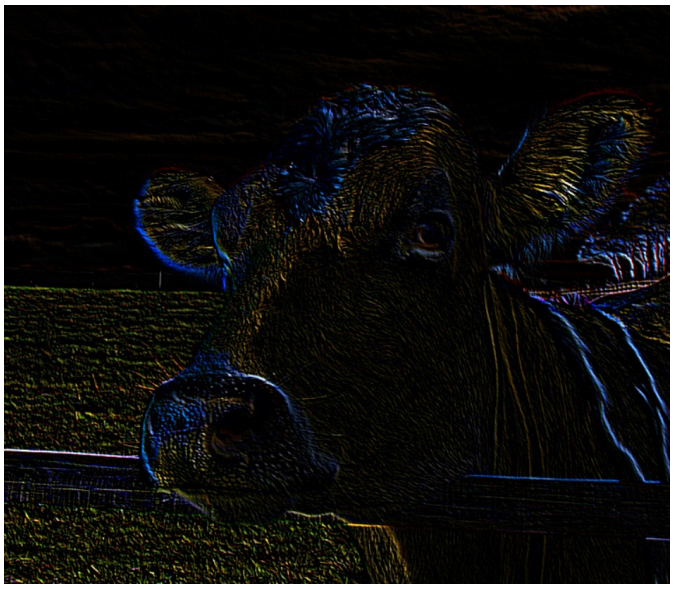
Here's applied filter:



6. Embossing Filter (`apply_embossing_filter`):

- This filter adds a 3D-like quality to the image by emphasizing edges through custom embossing.
- The user-defined embossing kernel is applied, imparting a distinct and visually striking effect.
-This custom kernel is used for embossing, creating a 3D-like effect by emphasizing edges in a specific direction. The arrangement of positive and negative weights produces a directional lighting effect.

Here's applied filter:



How to Use the Code:

1. Reading an Input Image:

- Before diving into the filters, ensure you have an input image. Use the `cv2.imread()` function to read your chosen image into the script.

2. Choosing a Filter:

- Presented with a menu, users can choose from a variety of filters by entering the corresponding number.

3. Applying the Selected Filter:

- Once the user makes a selection, the corresponding function is invoked to apply the chosen filter to the input image.

4. Saving the Result:

- Finally, the result is saved to an output file using the `cv2.imwrite()` function.

Conclusion:

In conclusion, the Image Filtering Code offers an immersive and interactive experience in image processing. From enhancing details to creating artistic stylizations, the code provides a platform for creative experimentation and exploration of the diverse realms of image manipulation. With a user-friendly design and a rich set of functions, this code invites users to unlock the full potential of image filtering and digital artistry.

```
import cv2
import numpy as np

# Function to apply edge-preserving filter and save the result
def apply_edge_preserving_recursive_filter(input_image):
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]])
    return cv2.filter2D(input_image, -1, kernel)

# Function to apply edge-preserving filter with normalized
convolution and save the result
def
apply_edge_preserving_normalized_convolution_filter(input_image):
    kernel = np.array([[0, -1, 0],
                       [-1, 9, -1],
                       [0, -1, 0]])
    return cv2.filter2D(input_image, -1, kernel)

# Function to apply detail enhance filter and save the result
def apply_detail_enhance_filter(input_image):
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]])
    detail_enhanced_image = cv2.filter2D(input_image, -1, kernel)
    return cv2.addWeighted(input_image, 1.5,
detail_enhanced_image, -0.5, 0)

# Function to apply pencil sketch filter and save the result in
both grayscale and color
def apply_pencil_sketch_filter(input_image):
    gray_image = cv2.cvtColor(input_image, cv2.COLOR_BGR2GRAY)
    inverted_image = cv2.bitwise_not(gray_image)
    pencil_sketch = cv2.divide(255 - inverted_image, 255,
scale=256.0)
    return gray_image, cv2.cvtColor(pencil_sketch,
cv2.COLOR_GRAY2BGR)

# Function to apply stylization filter and save the result
def apply_stylization_filter(input_image):
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]])
```

```
    stylized_image = cv2.filter2D(input_image, -1, kernel)
    return cv2.addWeighted(input_image, 0.5, stylized_image, 0.5,
0)

# Function to apply a custom filter and save the result
def apply_custom_filter(input_image, kernel):
    return cv2.filter2D(input_image, -1, kernel)

# Function to apply custom embossing filter and save the result
def apply_embossing_filter(input_image):
    # Custom embossing kernel
    custom_emboss_kernel = np.array([[0, -1, -1],
                                     [1,  0, -1],
                                     [1,  1,  0]])
    return apply_custom_filter(input_image, custom_emboss_kernel)

# Read the input image
image_path = "cow.jpg"
input_image = cv2.imread(image_path)

# Menu
print("Choose a filter to apply:")
print("1. Edge-Preserving Recursive Filter")
print("2. Edge-Preserving Normalized Convolution Filter")
print("3. Detail Enhance Filter")
print("4. Pencil Sketch Filter")
print("5. Stylization Filter")
print("6. Embossing Filter")

choice = int(input("Enter the number of your choice: "))

# Apply selected filter
if choice == 1:
    result = apply_edge_preserving_recursive_filter(input_image)
elif choice == 2:
    result =
apply_edge_preserving_normalized_convolution_filter(input_image)
elif choice == 3:
    result = apply_detail_enhance_filter(input_image)
elif choice == 4:
    _, result = apply_pencil_sketch_filter(input_image)
elif choice == 5:
    result = apply_stylization_filter(input_image)
```

```
elif choice == 6:
    result = apply_embossing_filter(input_image)
else:
    print("Invalid choice. Please choose a number from 1 to 6.")
    # Display the original image in case of an invalid choice
    result = input_image

# Save the result
output_filename = "filtered_result.jpg"
cv2.imwrite(output_filename, result)
print(f"Result saved as {output_filename}")
```