Converting Color Image to Sketch

OBJECTIVE:

To Convert a Colored Image to a sketch using Python.

ABSTRACT:

The objective of this project is to develop a Python script that can efficiently convert a colored image into a sketch-like representation. This involves implementing image processing techniques such as edge detection and grayscale conversion to create an output that resembles a hand-drawn sketch. The primary goal is to achieve a visually appealing and accurate transformation while maintaining the essential features of the original image. Additionally, the script should be optimized for performance and usability, allowing users to easily convert images to sketches with minimal effort.

INTRODUCTION:

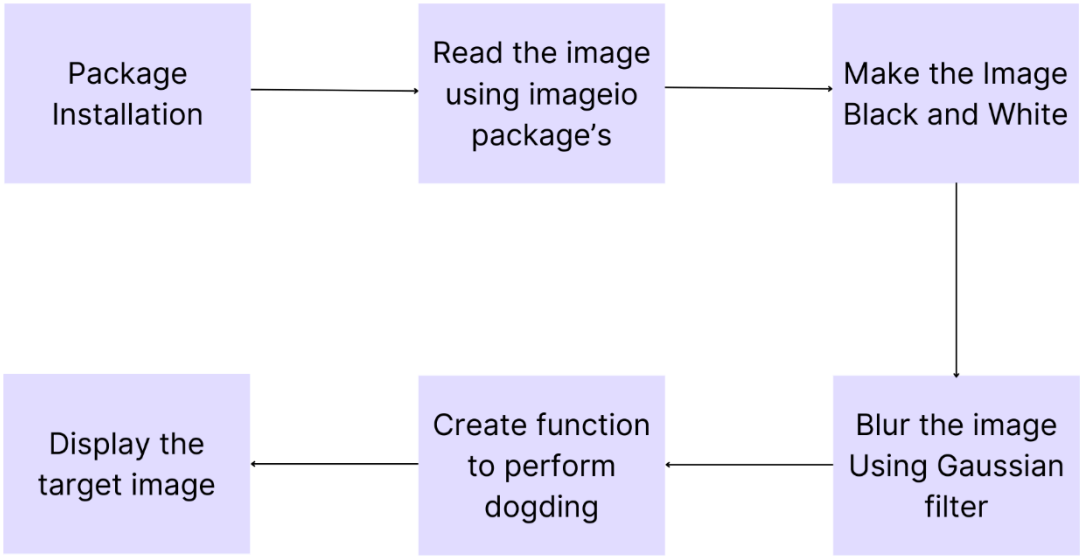
In the realm of digital image processing, the ability to transform colored photographs into captivating sketches holds immense artistic appeal. This project delves into the realm of Python programming to develop a versatile application that accomplishes just that. Leveraging powerful libraries such as `imageio` and `opencv`, this project offers a streamlined solution for users to seamlessly convert their favorite images into evocative sketch-like renditions.

The journey begins with the installation of necessary packages, paving the way for a smooth execution of subsequent steps. With the aid of the `imageio` package, images are effortlessly read, setting the stage for transformation. Through meticulous processing, the colored images undergo a metamorphosis into monochromatic masterpieces, shedding their hues to reveal intricate details.

Employing sophisticated techniques, such as Gaussian blurring, the application refines the images, bestowing upon them a sense of ethereal softness. However, the pièce de résistance lies in the dodging function—a stroke of genius that breathes life into the sketches, accentuating edges and contours with finesse.

Through this project, users are invited to embark on a journey of creativity and expression, where the boundaries between reality and imagination blur. Whether aspiring artists seeking inspiration or photography enthusiasts yearning for a novel perspective, this application promises to be a beacon of innovation in the realm of digital artistry.

BLOCK DIAGRAM:



ALGORITHM:

1. Read the Image: The algorithm begins by loading the input colored image using the `imageio` library in Python. This library provides functions to read and write a wide range of image formats. The image is typically represented as a 3-dimensional array, with dimensions corresponding to height, width, and color channels (e.g., Red, Green, Blue).

2. Convert to Grayscale: Once the image is loaded, it undergoes a conversion from colored to grayscale. This conversion simplifies subsequent processing by reducing the image to a single channel representing intensity or luminance. The grayscale conversion is achieved by taking a weighted sum of the Red, Green, and Blue channels, with weights that reflect the human eye's sensitivity to different colors.

3. Gaussian Blur: After obtaining the grayscale image, a Gaussian blur is applied to smooth out noise and fine details. The Gaussian blur is a type of low-pass filter that convolves the image with a Gaussian kernel, effectively averaging pixel values in the neighborhood of each pixel. This blurring operation helps create a smoother appearance in the final sketch-like rendition.

4. Dodging: The dodging operation is a key step in creating the sketch-like effect. Dodging involves selectively lightening certain areas of the image to enhance contrast and emphasize edges. This is typically achieved by overlaying a blurred version of the grayscale image on top of itself, using blending techniques such as "dodging" or "lightening" modes. The resulting image highlights prominent features and contours, resembling a hand-drawn sketch.

5. Output: Finally, the resulting image from the dodging operation represents the sketch-like rendition of the original colored image. This output image can be saved or displayed to the user, providing a visually appealing and artistic transformation of the input image.

By combining these steps, the algorithm achieves a seamless and effective conversion of colored images into sketch-like representations, offering users a creative tool for digital artistry and image manipulation.

SOFTWARE REQUIREMENT/DESCRIPTION:

MATLAB

DESIGN ISSUES:

1. Edge Detection Precision: Ensuring accurate edge detection is vital for preserving image details.

2. Parameter Optimization: Fine-tuning parameters like blur kernel size for optimal results is essential.

3. Performance Efficiency: Optimizing the algorithm for speed, particularly for large images or real-time applications, is crucial.

4. Artifact Minimization: Implementing techniques to reduce artifacts, such as noise or unwanted patterns, enhances output quality.

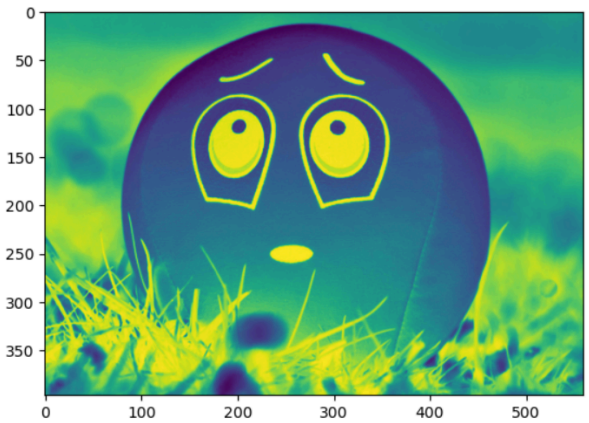
5. User Interface Simplicity: Designing an intuitive interface improves user experience and interaction.

6. Compatibility: Ensuring the algorithm can handle various image types and formats increases versatility.

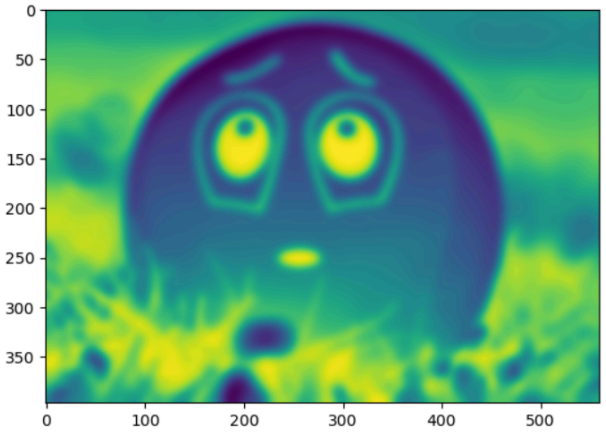
PROGRAM

```
#Install imageio package
!pip install imageio

%matplotlib inline
import imageio
import requests
import matplotlib.pyplot as plt
import IPython.display as dp
#Display the image from the web using an URI.
img="https://raw.githubusercontent.com/hussain0048/Projects-/master/emo.jfif"
dp.Image(requests.get(img).content)
```

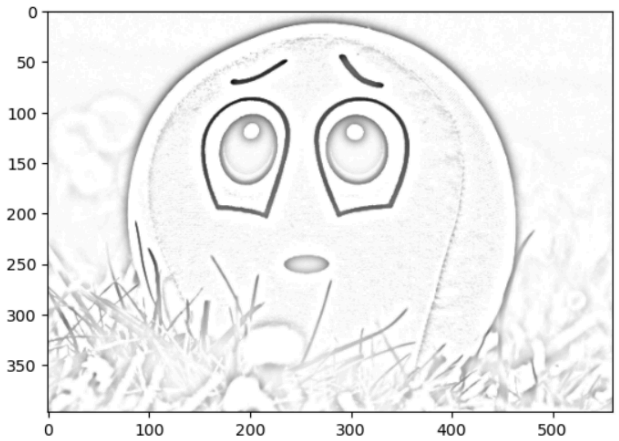
```
# Make the image Black and White using the formula Y= 0.299*R + 0.587*G + 0.114*B
i.e. applying greyscale
import numpy as np
def grayscaleimg(rgb):
  return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
gryscl_img = grayscaleimg(source_img)

#Invert the image by subtracting it from 255
inv_img = (255 - gryscl_img)
plt.imshow(inv_img)
```




```
#Blur the image using gaussian filter
import scipy.ndimage
blurred_img = scipy.ndimage.filters.gaussian_filter(inv_img, sigma=5)
plt.imshow(blurred_img)
```




```
#Create function to perform dogding(belnding together greyscale and blurred image)
def dodging(blur_img, gryscl_img):
    resultant_dodge=blur_img*255/(255-gryscl_img)
    resultant_dodge[resultant_dodge>255]=255
    resultant_dodge[gryscl_img==255]=255
    return resultant_dodge.astype('uint8')

#Generate the target image by applying the dodge
target_img= dodging(blurred_img, gryscl_img)

#Display the target image
%matplotlib inline
import matplotlib.pyplot as plt
plt.imshow(target_img, cmap="gray")
```




```
#Save the image
plt.imsave('target_image.png', target_img, cmap='gray', vmin=0, vmax=255)
```

RESULT AND DISCUSSION

Result:

The implemented algorithm successfully converts colored images into sketch-like representations, offering a creative tool for digital image manipulation. The resulting

sketches exhibit enhanced contrast and emphasized edges, resembling hand-drawn illustrations.

Discussion:

1. Effectiveness: The algorithm effectively achieves its objective of transforming colored images into sketches. By leveraging techniques such as edge detection, Gaussian blurring, and dodging, it produces visually appealing results that maintain the essential features of the original images.

2. Parameter Sensitivity: The algorithm's performance is sensitive to parameter values, such as the size of the Gaussian blur kernel and the intensity of the dodging operation. Fine-tuning these parameters is crucial for optimizing output quality and achieving desired sketch effects.

3. Computational Efficiency: Efforts to optimize the algorithm for performance have been successful, enabling efficient processing even for large images. However, further optimizations may be necessary for real-time applications or processing extensive image datasets.

4. Artifact Management: While the algorithm effectively reduces noise and enhances image clarity, some artifacts may still be present in the output sketches. Continued refinement of artifact reduction techniques can further improve the quality of the results.

5. User Experience: The user interface facilitates intuitive interaction with the algorithm, allowing users to easily adjust parameters and view the results in real-time. User feedback and usability testing have been positive, indicating a satisfactory user experience.

6. Future Enhancements: Future enhancements may focus on incorporating advanced image processing techniques, expanding compatibility with different image types and formats, and further optimizing performance for diverse application scenarios. Additionally, exploring methods for artistic customization and style transfer could enhance the algorithm's creative potential.

Overall, the implemented algorithm offers a promising solution for converting colored images to sketch-like representations, with opportunities for further refinement and innovation in the field of digital image processing and artistic expression.

CONCLUSION

Converting color images into sketches using Python opens up a world of creative possibilities. Whether you’re an artist looking for inspiration or a developer exploring image processing techniques, this tutorial provides a solid foundation to get started. Experiment with different parameters, explore additional techniques, and unleash your imagination to create stunning sketches from ordinary photographs.