**Intensity Transformations and Neighborhood Filtering**

**GitHub Link :-** [**https://github.com/Manimohan05/Image-Processing-and-Machine-Vision--A01-Intensity-Transformations-and-Neighborhood-Filtering.git**](https://github.com/Manimohan05/Image-Processing-and-Machine-Vision--A01-Intensity-Transformations-and-Neighborhood-Filtering.git)

1. **Question – 01**

In this question, we must implement an intensity transformation on the input image. Through the adjustment of pixel values using this transformation, our objective is to enhance the overall visual quality and appearance of the image.

Important part of the code

c = np.array([(100, 50), (150, 200)])

arr1 = np.linspace(0, c[0, 1], c[0, 1] + 1 - 0).astype('uint8')

arr2 = np.linspace(c[0, 0] + 1, 255, c[1, 1] - c[0, 0]).astype('uint8')

arr3 = np.linspace(c[1, 0] + 1, 255, 255 - c[1, 0]).astype('uint8')

transform = np.concatenate((arr1, arr2), axis=0).astype('uint8')

transform = np.concatenate((transform, arr3), axis=0).astype('uint8')

image\_transformed = cv.LUT(img\_orig, transform)

A graph with a line

Description automatically generated

figure 1:- Intensity transformation

**A collage of women with white paint on their faces

Description automatically generated**

figure 2:- Original and Transformed

1. **Question – 02**

Our task involves carrying out a comparable procedure to the one mentioned earlier, with the goal of highlighting specific brain tissue features in proton density images. More precisely, our objective is to improve the perceptibility of both white matter and gray matter separately. This requires developing personalized intensity adjustments specific to each type of tissue. To record this procedure, we will generate plots of intensity transformations that visually depict the modifications applied to enhance the visibility of white matter and gray matter within the image.

Important code for accentuating gray matter.

c= np.array([(0,50),(150,100),(180,255)])

arr1=np.linspace(0,c[1,1],c[0,1]+1).astype("uint8")

arr2 = np.linspace(c[2,0]+1,c[2,1],c[2,0]-c[0,1]).astype("uint8")

arr3 = np.linspace(c[0,1]-20,c[2,0]-c[0,1],255-c[2,0]).astype("uint8")

gray\_matter\_transformation = np.concatenate((arr1,arr2),axis=0).astype("uint8")

**A graph with a line

Description automatically generated**gray\_matter\_transformation = np.concatenate((gray\_matter\_transformation,arr3),axis=0).astype("uint8")

figure 3:- Intensity transformation(gray)

Important code for accentuating white matter.

c=np.array([(50,25),(50,90),(180,220),(255,255)])

arr1=np.linspace(0,c[0,1],c[0,0]+1).astype("uint8")

arr2 = np.linspace(c[0,1]+1,c[1,1],c[2,0]-c[1,0]).astype("uint8")

arr3 = np.linspace(c[2,1]+1,255,255-c[2,0]).astype("uint8")

white\_matter\_transformation = np.concatenate((arr1,arr2),axis=0).astype("uint8")

**A graph with a line

Description automatically generated**white\_matter\_transformation= np.concatenate((white\_matter\_transformation,arr3),axis=0).astype("uint8")

figure 4:- Intensity transformation(white)

**A close-up of a brain scan

Description automatically generated**

figure 5:- Original and Transformed

1. **Question – 03**

In this task, we are utilizing gamma correction on the L plane within the L\*a\*b color space. Gamma correction is a method employed to fine-tune the intensity levels of an image, and in this instance, it will be employed on the luminance channel (L) of the color space. We will be given a particular value that determines the degree of correction. Furthermore, we will create histograms to visually contrast the intensity distribution of the original L channel with that of the adjusted version. These histograms provide a means to assess the level of improvement accomplished through the gamma correction procedure.

Important part of code

# Load the original image

img\_orig = cv.imread('images\highlights\_and\_shadows.jpg', cv.IMREAD\_COLOR)

# Convert the image to LAB color space

img\_lab = cv.cvtColor(img\_orig, cv.COLOR\_BGR2LAB)

# Apply gamma correction to the L channel

gamma = 1.4

img\_lab[:,:,0] = (((img\_orig[:,:,0]/255)\*\*gamma)\*255).astype(np.uint8)

# Convert the image back to BGR color space

img\_gamma = cv.cvtColor(img\_lab, cv.COLOR\_LAB2BGR)

A comparison of a person's legs

Description automatically generated

figure 6:- Original and L plane corrected Images

A close-up of a graph

Description automatically generated

figure 7:- Histograms

1. **Question – 04**

To enhance the vibrancy of a photograph, we employ a particular saturation plane transformation utilizing the function f(x). This method entails breaking down the image into its hue, saturation, and value components. The transformation is then applied to the saturation plane, with the parameter 'a' adjusted to achieve the best outcomes. Following this adjustment, the modified components are combined back together, resulting in an enhanced version of the original image with increased vibrance. To provide a comprehensive visual comparison, we display both the enhanced image and the transformation graph alongside the original.

# Load the image in BGR color space

image\_bgr = cv.imread('images/spider.png', cv.IMREAD\_COLOR)

# Convert the BGR image to HSV color space

image\_hsv = cv.cvtColor(image\_bgr, cv.COLOR\_BGR2HSV)

# Extract the saturation (S) plane

saturation\_plane = image\_hsv[:, :, 1]

# Define the intensity transformation function

def intensity\_transform(x, a, sigma=70):

    # Apply an intensity transformation to enhance vibrance

    transformed\_x = np.clip(x + a \* 128 \* np.exp(-(x - 128) \*\* 2 / (2 \* sigma \*\* 2)), 0, 255)

    return transformed\_x

# Apply the intensity transform function to enhance saturation

transformed\_saturation\_plane = intensity\_transform(saturation\_plane, 0.5)

# Create a copy of the HSV image for modification

image\_copy = image\_hsv.copy()

# Recombine the planes with the transformed saturation

image\_copy[:, :, 1] = transformed\_saturation\_plane

# Convert the HSV image back to BGR color space

original\_image = cv.cvtColor(image\_hsv, cv.COLOR\_HSV2RGB)

transformed\_image = cv.cvtColor(image\_copy, cv.COLOR\_HSV2RGB)

**A collage of a couple of people in clothing

Description automatically generated**

**A graph with a line

Description automatically generated**

figure 8:- Original and Enhanced

figure 9:- Intensity transformation

figure 8:- Original and Enhanced

1. **Question – 05**

This assignment requires the development of a custom Python function for histogram equalization applied to a given image. This process is designed to exclude the use of built-in functions like 'cv2.equalizeHist()'. The function begins by loading the image as a NumPy array and subsequently computes its histogram and cumulative distribution function (CDF). Following this, the CDF is normalized to ensure it falls within the desired range. The core step involves redistributing pixel intensities according to the CDF. Finally, the histograms before and after equalization are displayed, providing insight into how the transformation impacts the image's intensity distribution.

important code related to histogram equalization:

def compute\_histogram(image):

    # Compute the histogram of the input image

    hist, bins = np.histogram(image.ravel(),256,[0, 256])

    return hist, bins

def compute\_cumulative\_histogram(hist):

    # Compute the cumulative histogram

    cumulative\_hist = hist.cumsum()

    normalized\_cumulative\_hist = cumulative\_hist \* hist.max() / cumulative\_hist.max()

    return cumulative\_hist, normalized\_cumulative\_hist

def compute\_probabilities(hist):

    # Compute probabilities for each intensity level

    probabilities = hist / np.sum(hist)

    return probabilities

def compute\_cumulative\_sum(hist):

    # Compute the cumulative sum of pixels

    cumulative\_sum = np.zeros(256)

    for i in range(len(cumulative\_sum)):

        cumulative\_sum[i] = np.sum(hist[:i])

    return cumulative\_sum

def equalize\_histogram(image, cumulative\_sum):

    # Equalize the image using the cumulative sum

    equalized\_cumulative\_sum = np.zeros(256)

    for x in range(len(equalized\_cumulative\_sum)):

        equalized\_cumulative\_sum[x] = (cumulative\_sum[x] \* 255) / image.size

    equalized\_cumulative\_sum = equalized\_cumulative\_sum.astype('uint8')

    equalized\_image = np.zeros(image.shape)

    for i in range(len(image)):

        for j in range(len(image[i])):

            equalized\_image[i][j] = equalized\_cumulative\_sum[image[i][j]]

    equalized\_image = equalized\_image.astype('uint8')

    return equalized\_image

A comparison of images of coffee beans

Description automatically generated

figure 10:- Original and Equalized

**A graph of a diagram and a graph of a diagram

Description automatically generated**

figure 11:- Original and Equalized Histograms

1. **Question – 06**This task involves improving the histogram exclusively for the foreground of an image, resulting in an equalized histogram for the image's foreground. The process consists of several steps, which include segmenting the image into its hue, saturation, and value components, creating a foreground mask through thresholding, and using operations such as 'cv2.bitwise and' to isolate the foreground region. Next, the histogram for the foreground is calculated, and its cumulative sum is determined. The application of histogram equalization formulas enhances histogram for the foreground. Finally, the background information is reintegrated to achieve the desired result. The key components of this task involve the hue, saturation, and value planes, the foreground mask, the original image, and the resulting image displaying the histogram-equalized foreground.

This code implementation of this task.

# Convert the image to the HSV color space (Hue, Saturation, Value)

hsv\_image = cv.cvtColor(image, cv.COLOR\_BGR2HSV)

# Split the HSV image into its individual channels: Hue, Saturation, and Value

hue\_channel, saturation\_channel, value\_channel = cv.split(hsv\_image)

# Display the individual channels in grayscale

plt.figure(figsize=(12, 4))

# Manually set a threshold value (adjust as needed)

threshold\_value = 14

# Choose the saturation channel for thresholding

threshold\_channel = saturation\_channel

# Perform thresholding to extract the foreground mask

\_, foreground\_mask = cv.threshold(threshold\_channel, threshold\_value, 255, cv.THRESH\_BINARY)

# Extract the foreground region using cv2.bitwise\_and

foreground = cv.bitwise\_and(image, image, mask=foreground\_mask)

# Compute the histogram of the foreground region

foreground\_gray = cv.cvtColor(foreground, cv.COLOR\_BGR2GRAY)

hist, bins = np.histogram(foreground\_gray.ravel(), bins=256, range=[0, 255])

# Calculate the cumulative sum of the histogram

cdf = hist.cumsum()

# Perform histogram equalization on the foreground

equalized\_foreground\_gray = cv.equalizeHist(foreground\_gray).astype(np.uint8)

# Convert the equalized foreground back to BGR format

equalized\_foreground = cv.cvtColor(equalized\_foreground\_gray, cv.COLOR\_GRAY2BGR)

# Extract the background by inverting the foreground mask

background\_mask = cv.bitwise\_not(foreground\_mask)

background = cv.bitwise\_and(image, image, mask=background\_mask)

# Combine the equalized foreground with the background

result = cv.add(equalized\_foreground, background)

# Display the original image and the result image side by side

image\_rgb = cv.cvtColor(image, cv.COLOR\_BGR2RGB)

result\_rgb = cv.cvtColor(result, cv.COLOR\_BGR2RGB)

**A person with long hair

Description automatically generated**

**A person with long blonde hair

Description automatically generatedA white face with black background

Description automatically generated**

**A collage of a person

Description automatically generated**

figure 12:- h, s, v planes and Original, Foreground and Equalized Images

1. **Question – 07**

In this section, we explore various techniques for applying Sobel filtering to an image:

* In the initial approach, we employ the existing 'filter2D' function to execute Sobel filtering.
* In the second method, we craft our own custom code for Sobel filtering, granting us greater control over the process.
* Finally, we leverage a convolution property using a specific matrix configuration. By applying this matrix convolution to the image, we achieve the Sobel filtering effect, which effectively enhances and emphasizes the edges within the image.

figure 14:- Sobel filtering without using "filter2D" function.

Here is the code snippet that corresponds to the first approach:(for two different kernels)

# Define two convolution kernels for image filtering

ker1=np.array([[-1,0,1],[-2,0,2],[-1,0,1]])

ker2=np.array([[-1,-2,-1],[0,0,0],[1,2,1]])

# Apply the convolution operation with the first kernel

img1 = cv.filter2D(img, -1, ker1)

# Apply the convolution operation with the second kernel

A collage of different men's faces

Description automatically generatedimg2 = cv.filter2D(img, -1, ker2)

figure 13:- Sobel Filter Using "filter2D" function

Here is the code snippet that corresponds to the second approach:(for two different kernels)

# Define Sobel kernels

ker1 = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

ker2 = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

# Function to apply convolution filter

def filter(image, kernel):

    assert kernel.shape[0] % 2 == 1 and kernel.shape[1] % 2 == 1

    k\_hh, k\_hw = math.floor(kernel.shape[0] / 2), math.floor(kernel.shape[1] / 2)

    h, w = image.shape

    image\_float = cv.normalize(image.astype(float), None, 0.0, 1.0, cv.NORM\_MINMAX)

    result = np.zeros(image.shape, float)

    for m in range(k\_hh, h - k\_hh):

        for n in range(k\_hw, w - k\_hw):

            result[m, n] = np.dot(image\_float[m - k\_hh:m + k\_hh + 1, n - k\_hw:n + k\_hw + 1].flatten(), kernel.flatten())

    return result

# Apply Sobel filters

img\_1 = filter(img, ker1)

img\_2 = filter(img, ker2)

A collage of a person's face

Description automatically generated

Here is the code snippet that corresponds to the third approach:

kernel1 = np.array([[1], [2], [1]])

kernel2 = np.array([[1, 0, -1]])

kernel = kernel1\*kernel2

# Apply the 1D convolutions

conv1 = cv.filter2D(img, -1, kernel)

A person with white hair and a mustache

Description automatically generated

figure 15:- Sobel filtering using third approach.

1. **Question – 08**

This code is designed for image magnification with a specified factor 's' within the (0, 10) range. It offers two distinct zooming techniques: nearest-neighbor and bilinear interpolation. When utilizing the nearest-neighbor method, the program enlarges the image by replicating the nearest pixel values, preserving a blocky appearance. In contrast, the bilinear interpolation method calculates pixel values by considering the weighted average of neighboring pixels, resulting in smoother enlarged images. To assess the accuracy of the zooming implementation, it calculates the normalized sum of squared differences (SSD) when scaling up small images by a factor of 4 and compares them to the original images.

Important code for nearest-neighbor method

def zoom\_nearest\_neighbor(image, factor):

    h, w, \_ = image.shape

    new\_h = int(h \* factor)

    new\_w = int(w \* factor)

    zoomed\_image = np.zeros((new\_h, new\_w, 3), dtype=np.uint8)

    for i in range(new\_h):

        for j in range(new\_w):

            orig\_i = int(i / factor)

            orig\_j = int(j / factor)

            zoomed\_image[i, j] = image[orig\_i, orig\_j]

    return zoomed\_image

Important code for Bilinear method

def zoom\_bilinear(image, factor):

    h, w, \_ = image.shape

    new\_h = int(h \* factor)

    new\_w = int(w \* factor)

    zoomed\_image = np.zeros((new\_h, new\_w, 3), dtype=np.uint8)

    for i in range(new\_h):

        for j in range(new\_w):

            orig\_i = i / factor

            orig\_j = j / factor

            i1, i2 = int(np.floor(orig\_i)), int(np.ceil(orig\_i))

            j1, j2 = int(np.floor(orig\_j)), int(np.ceil(orig\_j))

            i1 = max(0, min(i1, h - 1))  # Ensure indices stay within image boundaries

            i2 = max(0, min(i2, h - 1))

            j1 = max(0, min(j1, w - 1))

            j2 = max(0, min(j2, w - 1))

            # Bilinear interpolation

            value = (1 - (orig\_i - i1)) \* (1 - (orig\_j - j1)) \* image[i1, j1] + \

                    (1 - (orig\_i - i1)) \* (orig\_j - j1) \* image[i1, j2] + \

                    (orig\_i - i1) \* (1 - (orig\_j - j1)) \* image[i2, j1] + \

                    (orig\_i - i1) \* (orig\_j - j1) \* image[i2, j2]

            zoomed\_image[i, j] = value.astype(np.uint8)

    return zoomed\_image

Important code for Calculate SSD

def compute\_normalized\_ssd(image1, image2):

    pixel\_difference = image1 - image2

    squared\_pixel\_difference = pixel\_difference \*\* 2

    sum\_of\_squared\_differences = np.sum(squared\_pixel\_difference)

    normalized\_ssd = sum\_of\_squared\_differences / (image1.size \* 255 \*\* 2)

    return normalized\_ssd

|  |  |  |
| --- | --- | --- |
| Image | Nearest Neighbor | Bilinear |
| 01 | 0.000481 | 0.000603 |
| 02 | 0.000183 | 0.000249 |
| 03 | - | - |
| 04 | 0.001210 | 0.001255 |
| 05 | 0.000777 | 0.000825 |
| 06 | 0.000469 | 0.000546 |
| 07 | 0.000430 | 0.000464 |
| 08 | - | - |
| 09 | 0.000325 | 0.000410 |
| 10 | - | - |
| 11 | - | - |

1. **Question – 09**

In the provided image of a flower, where both the foreground and background are in sharp focus, the task comprises two main objectives: 1. In the first part, the "grab-Cut" algorithm is utilized for image segmentation. This results in two primary outcomes: the creation of a segmentation mask that distinguishes the foreground from the background and the isolation of the foreground and background images.

2. In the second part, the objective is to enhance the image by applying significant background blur. This action aims to create a visually striking separation between the subject and its background. By presenting both the original and enhanced images side by side, viewers can clearly observe the improvements made.

Important code for this task:

# Create a mask initialized with zeros

mask = np.zeros(img.shape[:2], np.uint8)

# Define a rectangle that contains the object of interest (flower)

rectangle = (50, 50, 505, 505)  # (x, y, width, height)

# Apply GrabCut algorithm with rectangle initialization

background\_model = np.zeros((1, 65), np.float64)

foreground\_model = np.zeros((1, 65), np.float64)

cv.grabCut(img, mask, rectangle, background\_model, foreground\_model, 5, cv.GC\_INIT\_WITH\_RECT)

# Create a binary mask where the probable background and definite background are set to 0

mask1 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

foreground\_img = img \* mask1[:, :, np.newaxis]

# Apply GrabCut algorithm with mask initialization

cv.grabCut(img, mask, rectangle, background\_model, foreground\_model, 5, cv.GC\_INIT\_WITH\_MASK)

# Create a binary mask where the probable foreground and definite foreground are set to 0

mask2 = np.where((mask == 3) | (mask == 1), 0, 1).astype('uint8')

background\_img = img \* mask2[:, :, np.newaxis]

A collage of different colors of flowers

Description automatically generated

figure 16:- 6 Segmented, Foreground and Background images

# Create a mask initialized with zeros

mask = np.zeros(image.shape[:2], np.uint8)

# Define a rectangle that contains the object of interest (flower)

rect = (50, 50, image.shape[1] - 50, image.shape[0] - 50)

# Apply GrabCut algorithm with rectangle initialization

bgdModel = np.zeros((1, 65), np.float64)

fgdModel = np.zeros((1, 65), np.float64)

cv.grabCut(image, mask, rect, bgdModel, fgdModel, 5, cv.GC\_INIT\_WITH\_RECT)

# Modify the mask to get a binary mask

mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

mask3 = 1 - mask2

# Apply the mask to get segmented images

foreground\_img = image \* mask2[:, :, np.newaxis]

background\_img = image \* mask3[:, :, np.newaxis]

# Create a blurred background

blurred\_background = cv.GaussianBlur(background\_img, (0, 0), sigmaX=10, sigmaY=10)

blurred\_background = blurred\_background \* mask3[:, :, np.newaxis]

# Replace the background in the segmented image

enhanced\_img = foreground\_img + blurred\_background

A close-up of a yellow flower

Description automatically generated

figure 17:- Original and Enhanced images

Regarding Part C:

The darkening of the background near the flower's edge in an enhanced image is due to intentional blurring, a technique used to separate the subject from the background. This blurring, often used for creating a bokeh effect, reduces sharpness in distant elements, making the background intentionally blurry and darker. This enhances the flower's prominence as the focal point and creates a visually pleasing contrast between the subject and its surroundings.

\*\*\*