PRACTICAL FILE

**BE (CSE) 7th Semester**

**DIGITAL IMAGE PROCESSING (CS 751)**

**July 2023 – December 2023**

**Submitted By**

# Rohit Kumar Singh

**Roll Number: UE203093**

**Submitted To**

# Ms. Surbhi

**Faculty, Computer Science and Engineering**



**Computer Science and Engineering University Institute of Engineering and Technology Panjab University, Chandigarh – 160014, INDIA**

**2023**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Name of the Practical/Program** | **Page No.** | **Remarks** |
| 1 | Display images in different formats using different color models. | 3 - 5 |  |
| 2 | Perform gray level transformations on a grayscale image. | 6 - 10 |  |
| 3 | Perform power law transformations for different values of gamma on a grayscale image. | 11 - 14 |  |
| 4 | Perform histogram equalization for   1. *Grayscale image* 2. *RGB image* | 15 - 21 |  |
| 5 | Perform correlation on an image using filters of size 3x3, 5x5, 7x7 & 9x9 and compare their outputs. | 22 - 25 |  |
| 6 | Perform correlation and convolution on an image using a given filter and compare their outputs. | 26 - 28 |  |

# Practical 1: Display images in different formats using different color models.

**THEORETICAL DISCUSSION:**

1. **Computer Vision using OpenCV**

Computer Vision with OpenCV is a field of computer science that enables machines to interpret and analyze visual data from images and videos. OpenCV, an open-source computer vision library, provides a comprehensive set of tools for tasks like image processing, object detection, face recognition, and more. It offers pre-built functions and algorithms to simplify the development of computer vision applications.

1. **Color Models**

Color models are essential in image processing and computer vision, representing how colors are encoded and manipulated. The most common models are RGB (Red, Green, Blue) and BGR (Blue, Green, Red), with RGB widely used in computer graphics and displays, while BGR is typical in OpenCV.

* + *RGB (Red, Green, Blue):* In RGB, colors are defined by mixing different intensities of red, green, and blue light. This model is additive, where combining maximum intensity of all three channels results in white, while no intensity results in black. It is widely used in displays and digital imaging.
  + *BGR (Blue, Green, Red):* BGR is essentially the same as RGB but with the order of channels reversed. This model is commonly used in OpenCV due to how it stores pixel values.
  + *Grayscale:* Grayscale images have a single channel, representing intensity or brightness, with black as the lowest value and white as the highest. Grayscale images are useful for simplifying computations in various image processing tasks and reducing memory requirements.
  + *Binary Images:* Binary images have only two values, typically 0 and 255, representing black and white, respectively. They are used in image segmentation, object detection, and feature extraction. Converting an image to binary is often based on a threshold, where values above it become white, and those below become black.

# IMPLEMENTATION DETAILS:

**# 1. Importing Libraries**

import **cv2** as **cv**

import **matplotlib**.**pyplot** as **plt**

**# 2. Reading image and resizing**

img = **cv**.**imread**("fruits.png") h = **int**(img.shape[0]\*0.5)

w = **int**(img.shape[1]\*0.5) img = **cv**.**resize**(img, (w, h))

**# 3. Converting image to various colour models**

rgb\_img = **cv**.**cvtColor**(img, **cv**.COLOR\_BGR2RGB) gray\_img = **cv**.**cvtColor**(img, **cv**.COLOR\_BGR2GRAY)

ret, binary\_img = **cv**.**threshold**(gray\_img, 127, 255, **cv**.THRESH\_BINARY)

*# RGB image*

**cv**.**imshow**("BGR Image", rgb\_img)

**cv**.**imwrite**("RGB.png", rgb\_img)

*# BGR image* **cv**.**imshow**("RGB Image", img) **cv**.**imwrite**("BGR.png", img)

*# GRAY image*

**cv**.**imshow**("Gray Image", gray\_img)

**cv**.**imwrite**("GRAY.png", gray\_img)

*# BINARY image*

**cv**.**imshow**("Binary Image", binary\_img)

**cv**.**imwrite**("BINARY.png", binary\_img)

**cv**.**waitKey**(0) **cv**.**destroyAllWindows**()

**# 4. Plotting all the processed images**

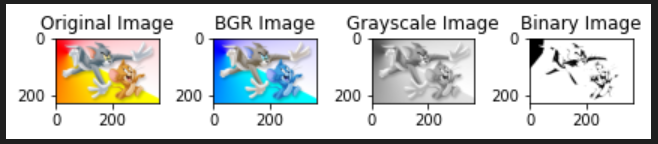
**plt**.**subplot**(2, 2, 1), **plt**.**imshow**(rgb\_img), **plt**.**title**("Original Image")

**plt**.**subplot**(2, 2, 2), **plt**.**imshow**(img), **plt**.**title**("BGR Image")

**plt**.**subplot**(2, 2, 3), **plt**.**imshow**(gray\_img, cmap="gray"), **plt**.**title**("Grayscale Image") **plt**.**subplot**(2, 2, 4), **plt**.**imshow**(binary\_img, cmap="gray"), **plt**.**title**("Binary Image") **plt**.**tight\_layout**()

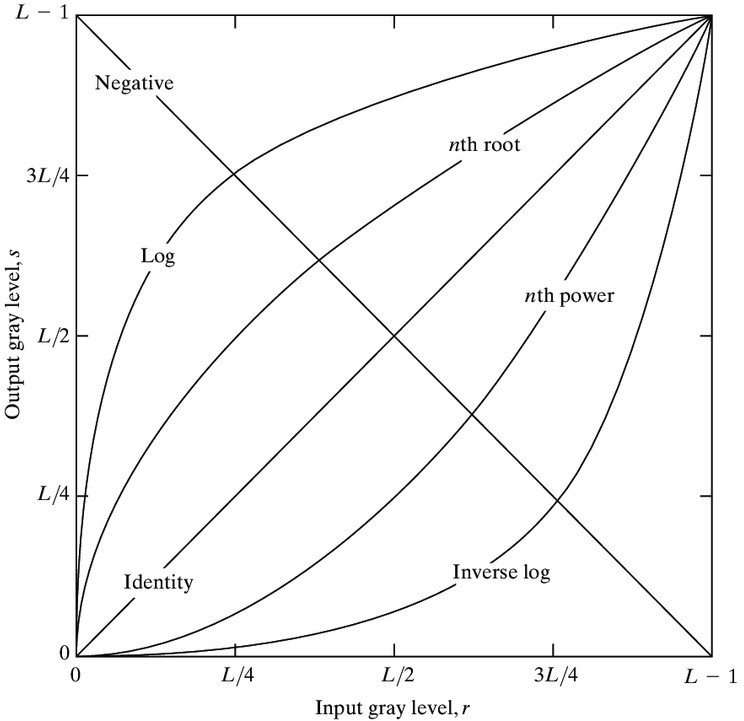
**plt**.**show**()

# OUTPUT:



**Practical 2: Perform gray level transformations on a grayscale image.**

# THEORETICAL DISCUSSION:



1. **Linear Transformations**:
   * *Identity Transformation*: This transformation leaves the pixel values unchanged, resulting in no change to the image appearance.
     + Mathematical Formula: (G (x, y) = F (x, y))
   * *Negative Transformation*: The negative transformation inverts the gray levels, making dark regions lighter and vice versa.
     + Mathematical Formula: G (x, y) = L - 1 - F (x, y)
2. **Logarithmic Transformations**:
   * *Log Transformation*: The log transformation is used to expand the dynamic range of dark pixels while compressing the dynamic range of brighter pixels.
     + Mathematical Formula: c × log (1 + F (x, y))
   * *Inverse Log Transformation*: The inverse log transformation is used to compress the dynamic range of dark pixels while expanding the dynamic range of brighter pixels.
     + Mathematical Formula: c × (10F (x, y) / c - 1)
3. **Power Law Transformations**:
   * *Nth Power Transformation*: It enhances the contrast for values close to zero (dark pixels for n > 1) or values close to the maximum (bright pixels for 0 < n < 1).
     + Mathematical Formula: G (x, y) = c × [F (x, y)] n
   * *Nth Root Transformation*: It is used to decrease the contrast of dark pixels (for n > 1) or bright pixels (for 0 < n < 1).
     + Mathematical Formula: G (x, y) = c × [F (x, y)] 1/n

# IMPLEMENTATION DETAILS:

**# 1. Importing Libraries**

import **cv2** as **cv**

import **numpy** as **np**

import **matplotlib**.**pyplot** as **plt**

**# 2. Reading image in grayscale and resizing**

img = **cv**.**imread**("fruits.png", **cv**.IMREAD\_GRAYSCALE) h = **int**(img.shape[0]\*0.5)

w = **int**(img.shape[1]\*0.5) img = **cv**.**resize**(img, (w, h))

**# 3. Display Original image and save**

*# Original GRAY image* **cv**.**imshow**("Original Image", img) **cv**.**imwrite**("Gray Image.png", img)

**cv**.**waitKey**(0) **cv**.**destroyAllWindows**()

**# 4. Identity transformation**

identity\_img = img

**cv**.**imshow**("Identity Image", identity\_img)

**cv**.**imwrite**("Identity Image.png", identity\_img)

**cv**.**waitKey**(0) **cv**.**destroyAllWindows**()

**# 5. Negative transformation**

negative\_img = 255 - img

**cv**.**imshow**("Negative Image", negative\_img)

**cv**.**imwrite**("Negative Image.png", negative\_img)

**cv**.**waitKey**(0) **cv**.**destroyAllWindows**()

**# 6. Log transformation**

log\_img = (30 \* **np**.log1p(1 + img)).clip(0, 255).astype(**np**.uint8)

**cv**.**imshow**("Log Image", log\_img)

**cv**.**imwrite**("Log Image.png", log\_img)

**cv**.**waitKey**(0) **cv**.**destroyAllWindows**()

**# 7. Inverse-Log transformation**

inverse\_log\_img = **np**.exp(img / 30.0).clip(0, 255).astype(**np**.uint8) - 1

**cv**.**imshow**("Inverse-Log Image", inverse\_log\_img)

**cv**.**imwrite**("Inverse-Log Image.png", inverse\_log\_img)

**cv**.**waitKey**(0) **cv**.**destroyAllWindows**()

**# 8. Nth power transformation**

n = 1.2

power\_law\_img = **np**.power(img, n).**clip**(0, 255).**astype**(**np**.uint8)

**cv**.**imshow**("Nth Power Image", power\_law\_img)

**cv**.**imwrite**("Nth Power Image.png", power\_law\_img)

**cv**.**waitKey**(0) **cv**.**destroyAllWindows**()

**# 9. Nth root transformation**

n = 1.5

root\_law\_img = **np**.power(img, 1/n).**clip**(0, 255).**astype**(**np**.uint8)

**cv**.**imshow**("Nth Root Image", root\_law\_img)

**cv**.**imwrite**("Nth Root Image.png", root\_law\_img)

**cv**.**waitKey**(0) **cv**.**destroyAllWindows**()

**# 10. Plotting the transformed images**

*# Linear transformations*

**plt**.**figure**(figsize=(5, 7))

**plt**.**suptitle**("Linear Transformations", fontsize=16)

**plt**.**subplot**(3, 2, 1) **plt**.**imshow**(identity\_img, cmap="gray") **plt**.**title**("Identity Image")

**plt**.**subplot**(3, 2, 2) **plt**.**imshow**(negative\_img, cmap="gray") **plt**.**title**("Negative Image")

**plt**.**tight\_layout**()

*# Logarithmic transformations*

**plt**.**figure**(figsize=(5, 7))

**plt**.**suptitle**("Logarithmic Transformations", fontsize=16)

**plt**.**subplot**(3, 2, 1) **plt**.**imshow**(log\_img, cmap="gray") **plt**.**title**("Log Image")

**plt**.**subplot**(3, 2, 2) **plt**.**imshow**(inverse\_log\_img, cmap="gray") **plt**.**title**("Inverse-Log Image")

**plt**.**tight\_layout**()

*# Power transformations*

**plt**.**figure**(figsize=(5, 7))

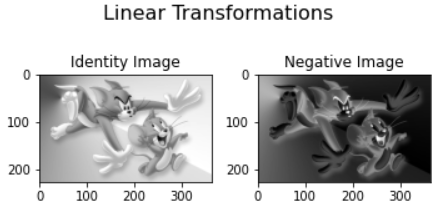
**plt**.**suptitle**("Power Transformations", fontsize=16)

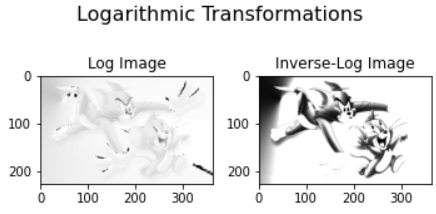
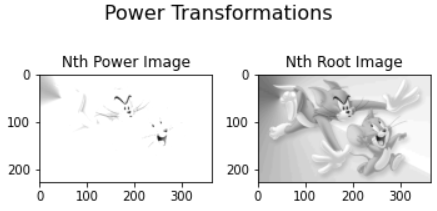
**plt**.**subplot**(3, 2, 1) **plt**.**imshow**(power\_law\_img, cmap="gray") **plt**.**title**("Nth Power Image")

**plt**.**subplot**(3, 2, 2) **plt**.**imshow**(root\_law\_img, cmap="gray") **plt**.**title**("Nth Root Image")

**plt**.**tight\_layout**() **plt**.**show**()

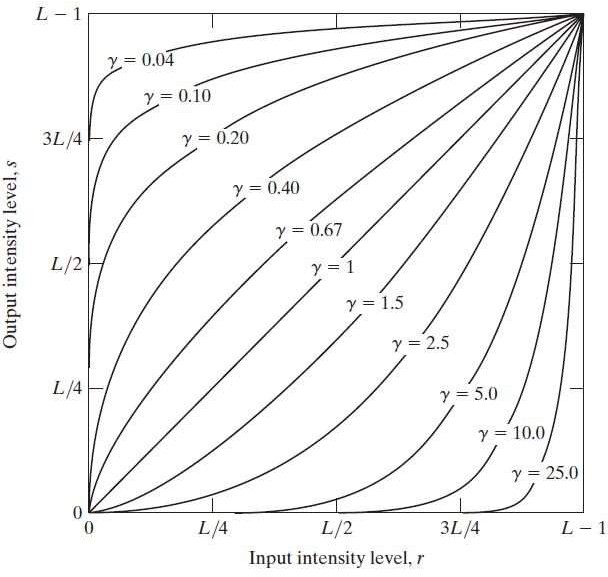
# OUTPUT:





**Practical 3: Perform power law transformations for different values of gamma on a grayscale image.**

# THEORETICAL DISCUSSION:



* *Gamma values from 0 to 1*:
  + As gamma approaches 0, the transformation becomes highly nonlinear, with a focus on enhancing the contrast in darker areas of the image.
  + A gamma of 0 effectively maps all pixel values to 1, resulting in an extreme high-key effect.
  + As gamma increases from 0 towards 1, the transformation becomes less extreme, providing smoother contrast adjustments.
* *Gamma values from 1 to infinity*:
  + A gamma of 1 has no effect on the image, as it maintains the original pixel values.
  + As gamma increases beyond 1, the transformation becomes increasingly nonlinear, with a focus on enhancing the contrast in brighter areas of the image.
  + Higher gamma values (much greater than 1) create a more pronounced S-curve in the transformation, enhancing the mid-tones and highlights while compressing the shadows.
  + Extremely high gamma values result in a drastic reduction in the overall contrast, causing the image to become almost monochromatic and washed out.

# IMPLEMENTATION DETAILS:

**# 1. Importing Libraries**

import **cv2** as **cv**

import **numpy** as **np**

import **matplotlib**.**pyplot** as **plt**

**# 2. Reading image in grayscale and resizing**

img = **cv**.**imread**("fruits.png", **cv**.IMREAD\_GRAYSCALE) h = **int**(img.shape[0]\*0.5)

w = **int**(img.shape[1]\*0.5) img = **cv**.**resize**(img, (w, h))

**# 3. Display Original image and save**

*# Original GRAY image* **cv**.**imshow**("Original Image", img) **cv**.**imwrite**("Gray Image.png", img)

**cv**.**waitKey**(0) **cv**.**destroyAllWindows**()

**# 4. Power law transformations for various values of gamma**

gamma1 = 0.34

power\_law\_img\_1 = **np**.power(img, gamma1).**clip**(0, 255).**astype**(**np**.uint8) gamma2 = 0.4

power\_law\_img\_2 = **np**.power(img, gamma2).**clip**(0, 255).**astype**(**np**.uint8) gamma3 = 0.5

power\_law\_img\_3 = **np**.power(img, gamma3).**clip**(0, 255).**astype**(**np**.uint8) gamma4 = 0.67

power\_law\_img\_4 = **np**.power(img, gamma4).**clip**(0, 255).**astype**(**np**.uint8) gamma5 = 1

power\_law\_img\_5 = **np**.power(img, gamma5).**clip**(0, 255).**astype**(**np**.uint8) gamma6 = 1.03

power\_law\_img\_6 = **np**.power(img, gamma6).**clip**(0, 255).**astype**(**np**.uint8) gamma7 = 1.06

power\_law\_img\_7 = **np**.power(img, gamma7).**clip**(0, 255).**astype**(**np**.uint8) gamma8 = 1.09

power\_law\_img\_8 = **np**.power(img, gamma8).**clip**(0, 255).**astype**(**np**.uint8) gamma9 = 1.12

power\_law\_img\_9 = **np**.power(img, gamma9).**clip**(0, 255).**astype**(**np**.uint8)

**# 5. Plotting the transformed images**

*# Power transformations*

**plt**.**figure**(figsize=(12, 5.5))

**plt**.**suptitle**("Power Transformations", fontsize=16)

**plt**.**subplot**(2, 5, 1) **plt**.**imshow**(power\_law\_img\_1, cmap="gray") **plt**.**title**("γ = 0.34")

**plt**.**subplot**(2, 5, 2) **plt**.**imshow**(power\_law\_img\_2, cmap="gray") **plt**.**title**("γ = 0.4")

**plt**.**subplot**(2, 5, 3) **plt**.**imshow**(power\_law\_img\_3, cmap="gray") **plt**.**title**("γ = 0.5")

**plt**.**subplot**(2, 5, 4) **plt**.**imshow**(power\_law\_img\_4, cmap="gray") **plt**.**title**("γ = 0.67")

**plt**.**subplot**(2, 5, 5) **plt**.**imshow**(power\_law\_img\_5, cmap="gray") **plt**.**title**("γ = 1")

**plt**.**subplot**(2, 5, 6) **plt**.**imshow**(power\_law\_img\_6, cmap="gray") **plt**.**title**("γ = 1.03")

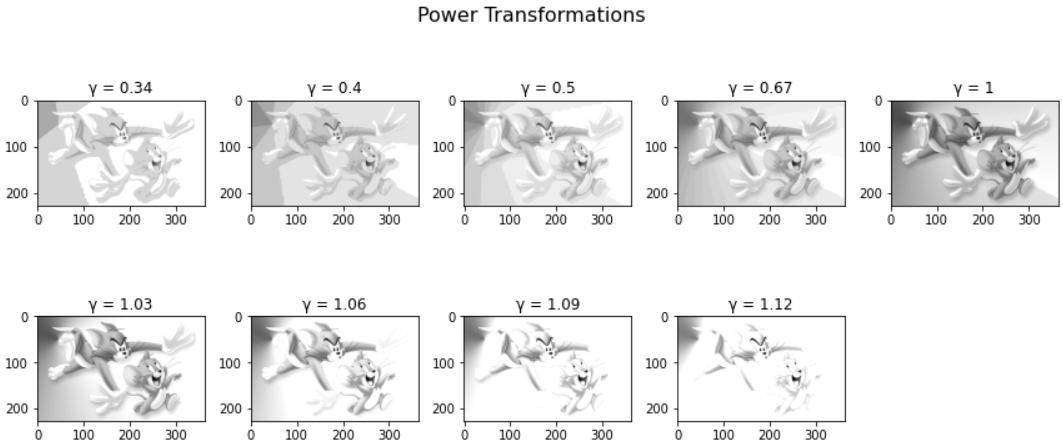
**plt**.**subplot**(2, 5, 7) **plt**.**imshow**(power\_law\_img\_7, cmap="gray") **plt**.**title**("γ = 1.06")

**plt**.**subplot**(2, 5, 8) **plt**.**imshow**(power\_law\_img\_8, cmap="gray") **plt**.**title**("γ = 1.09")

**plt**.**subplot**(2, 5, 9) **plt**.**imshow**(power\_law\_img\_9, cmap="gray") **plt**.**title**("γ = 1.12")

**plt**.**tight\_layout**() **plt**.**show**()

# OUTPUT:



**Practical 4: Perform histogram equalization for a grayscale and a RGB image.**

# THEORETICAL DISCUSSION:

* Histogram equalization is a contrast enhancement technique used in image processing and computer vision.
* The primary goal of histogram equalization is to adjust the pixel intensities in an image so that they cover the entire available intensity range, making the image more visually appealing.
* The process begins by computing the histogram of the input image, which is a distribution of pixel intensities, usually represented as a graph.
* Histogram equalization works by transforming the cumulative distribution function (CDF) of the image's histogram into a linear function. This effectively stretches the pixel intensity values across the entire range.
* The transformation is achieved by mapping the original pixel intensities to new values based on their position in the CDF. This ensures that areas of the image with low contrast are stretched while maintaining the relationships between pixel intensities.
* Histogram equalization can be applied to both grayscale and color images, with separate histograms computed for each channel in the case of color images.
* While histogram equalization can enhance image contrast, it may also exaggerate noise and other artifacts in the image. To mitigate this, techniques such as adaptive histogram equalization are used to limit contrast enhancement in uniform regions.
* One limitation of histogram equalization is that it may not always produce aesthetically pleasing results, especially in cases where the image has extreme variations in pixel intensities.
* Despite its limitations, histogram equalization is widely used in various image processing applications, including medical imaging, computer vision, and remote sensing, where improving contrast can lead to better feature extraction and analysis.
* Histogram equalization is a non-linear image enhancement technique and is reversible, meaning the original image can be accurately reconstructed from the equalized image data, given the cumulative distribution function.

# IMPLEMENTATION DETAILS:

**# 1. Importing Libraries**

import **cv2** as **cv**

import **numpy** as **np**

import **matplotlib**.**pyplot** as **plt**

1. ***Grayscale Image***

**# 2. Reading and plotting original grayscale image**

gray\_img = **cv**.**imread**("fruits.png", **cv**.IMREAD\_GRAYSCALE)

*# Plot the original grayscale image* **plt**.**figure**(figsize=(4, 3)) **plt**.**imshow**(gray\_img, cmap="gray") **plt**.**title**("Original Grayscale Image") **plt**.**tight\_layout**()

**# 3. Plotting Histogram and CDF of original grayscale image**

*# Calculate histogram of the grayscale image*

hist, bins = **np**.**histogram**(gray\_img.**flatten**(), 256, [0, 256])

*# Calculate the cumulative distribution function (CDF)*

cdf = hist.**cumsum**()

cdf\_normalized = cdf **\*** hist.**max**() / cdf.**max**() fig, axs = **plt**.**subplots**(1, 2, figsize=(15, 4))

*# Plot histogram*

axs[0].hist(gray\_img.**flatten**(), 256, [0, 256], color='k')

axs[0].set\_xlim([0, 256])

axs[0].set\_title('Original Grayscale Image Histogram')

*# Plot CDF* axs[1].plot(cdf\_normalized, color='k') axs[1].set\_xlim([0, 256])

axs[1].set\_title('Original Grayscale Image CDF')

*# Show the figure* **plt**.**tight\_layout**() **plt**.**show**()

**# 4. Equalizing the grayscale image**

equalized\_gray\_img = **cv**.**equalizeHist**(gray\_img)

**# 5. Plotting Histogram and CDF of the equalized grayscale image**

*# Calculate histogram of the equalized grayscale image*

hist, bins = **np**.**histogram**(equalized\_gray\_img.**flatten**(), 256, [0, 256])

*# Calculate the cumulative distribution function (CDF)*

cdf = hist.**cumsum**()

cdf\_normalized = cdf **\*** hist.**max**() / cdf.**max**() fig, axs = **plt**.**subplots**(1, 2, figsize=(15, 4))

*# Plot histogram*

axs[0].hist(equalized\_gray\_img.**flatten**(), 256, [0, 256], color='k')

axs[0].set\_xlim([0, 256])

axs[0].set\_title('Equalized Grayscale Image Histogram')

*# Plot CDF* axs[1].plot(cdf\_normalized, color='k') axs[1].set\_xlim([0, 256])

axs[1].set\_title('Equalized Grayscale Image CDF')

*# Show the figure* **plt**.**tight\_layout**() **plt**.**show**()

**# 6. Plotting and saving the equalized grayscale image**

**cv**.**imwrite**("Equalized Grayscale Image.png", equalized\_gray\_img)

*# Plot the equalized grayscale image* **plt**.**figure**(figsize=(4, 3)) **plt**.**imshow**(equalized\_gray\_img, cmap="gray") **plt**.**title**("Equalized Grayscale Image") **plt**.**tight\_layout**()

## RGB Image

**# 7. Reading and plotting original RGB image**

img = **cv**.**imread**("fruits.png")

rgb\_img = **cv**.**cvtColor**(img, **cv**.COLOR\_BGR2RGB)

*# Plot the original RGB image* **plt**.**figure**(figsize=(4, 3)) **plt**.**imshow**(rgb\_img) **plt**.**title**("Original RGB Image") **plt**.**tight\_layout**()

**# 8. Splitting the original RGB image into its red, green and blue colour channels**

b, g, r = **cv**.**split**(img)

**# 9. Plotting Histogram and CDF for each original channel**

fig, axs = **plt**.**subplots**(2, 3, figsize=(15, 8))

*# Plot histograms and CDFs for each channel*

for i, (channel, color) in **enumerate**(**zip**([r, g, b], ['r', 'g', 'b'])): hist, bins = **np**.**histogram**(channel.**flatten**(), 256, [0, 256]) cdf = hist.**cumsum**()

cdf\_normalized = cdf **\* float**(hist.**max**()) **/** cdf.**max**()

*# Plot histogram*

axs[0, i].hist(channel.**flatten**(), 256, [0, 256], color=color)

axs[0, i].set\_xlim([0, 256])

axs[0, i].set\_title(f'Original {color} Histogram')

*# Plot CDF*

axs[1, i].plot(cdf\_normalized, color=color) axs[1, i].set\_xlim([0, 256])

axs[1, i].set\_title(f'Original {color} CDF')

*# Show the figure* **plt**.**tight\_layout**() **plt**.**show**()

**# 10. Equalizing each channel in the RGB image**

r\_equalize = **cv**.**equalizeHist**(r) g\_equalize = **cv**.**equalizeHist**(g) b\_equalize = **cv**.**equalizeHist**(b)

**# 11. Plotting Histogram and CDF for each equalized channel**

fig, axs = **plt**.**subplots**(2, 3, figsize=(15, 8))

*# Plot histograms and CDFs for each channel*

for i, (channel, color) in **enumerate**(**zip**([r\_equalize, g\_equalize, b\_equalize], ['r', 'g', 'b'])):

hist, bins = **np**.**histogram**(channel.**flatten**(), 256, [0, 256]) cdf = hist.**cumsum**()

cdf\_normalized = cdf **\* float**(hist.**max**()) **/** cdf.**max**()

*# Plot histogram*

axs[0, i].hist(channel.**flatten**(), 256, [0, 256], color=color)

axs[0, i].set\_xlim([0, 256])

axs[0, i].set\_title(f'Equalized {color} Histogram')

*# Plot CDF*

axs[1, i].plot(cdf\_normalized, color=color) axs[1, i].set\_xlim([0, 256])

axs[1, i].set\_title(f'Equalized {color} CDF')

*# Show the figure* **plt**.**tight\_layout**() **plt**.**show**()

**# 12. Merging the equalized red, green and blue channels to obtain the equalized RGB image**

equalized\_RGB\_img = **cv**.**merge**([b\_equalize, g\_equalize, r\_equalize])

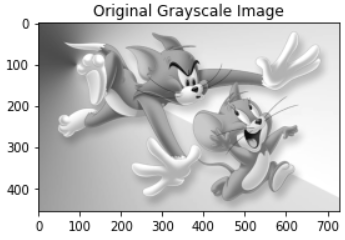
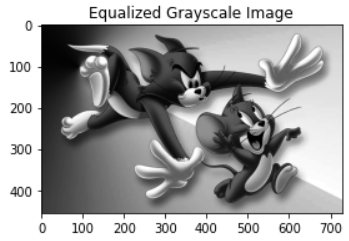
**# 13. Plotting and saving the equalized RGB image**

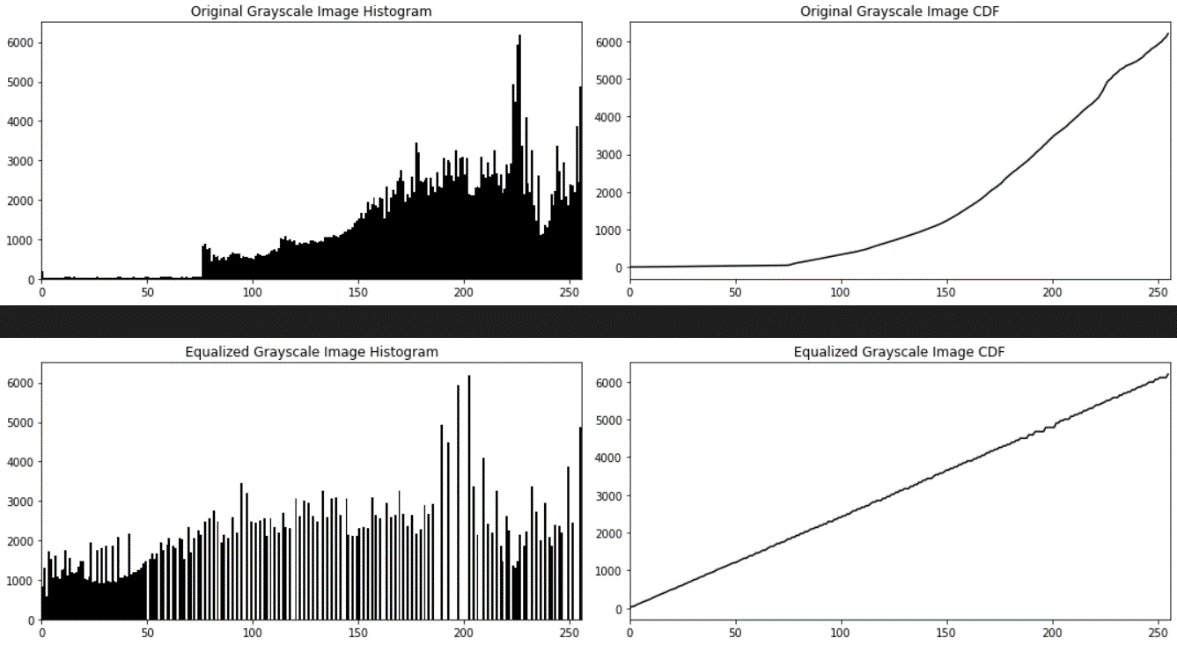
**cv**.**imwrite**("Equalized RGB Image.png", equalized\_RGB\_img) equalized\_rgb\_img = **cv**.**cvtColor**(equalized\_RGB\_img, **cv**.COLOR\_BGR2RGB)

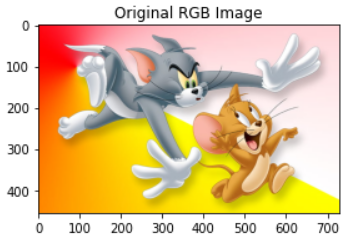
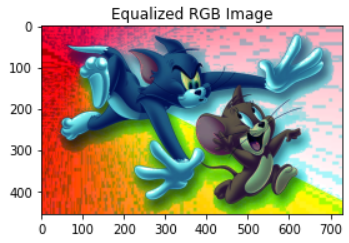
*# Plot the equalized RGB image* **plt**.**figure**(figsize=(4, 3)) **plt**.**imshow**(equalized\_rgb\_img) **plt**.**title**("Equalized RGB Image") **plt**.**tight\_layout**()

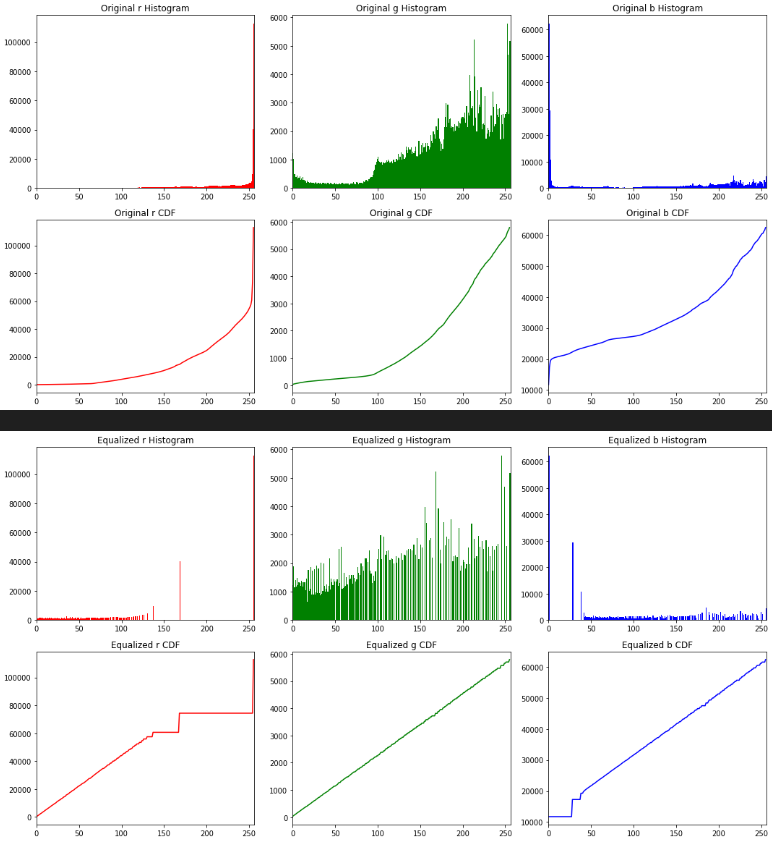
# OUTPUT:

## Grayscale Image





* 1. ***RGB Image***



# Practical 5: Perform correlation on an image using filters of size 3x3, 5x5, 7x7 & 9x9 and compare their outputs.

**THEORETICAL DISCUSSION:**

**Correlation**: Correlation is a fundamental operation in digital image processing that quantifies the similarity between two signals or images. It measures the degree to which one image resembles another in terms of feature patterns.

**Mathematical Representation**: Correlation is typically represented as a mathematical operation involving two functions, f (x, y) and g (x, y), where f represents the image and g represents the kernel or filter.

**Kernel (Filter)**: The kernel or filter, g (x, y), is a small matrix used for pattern matching within the image. It serves as a template that is slid over the input image to calculate the correlation at each position.

**Pixel-wise Multiplication**: At each position, the kernel is element-wise multiplied with the corresponding region of the image. This multiplication emphasizes regions where the image and kernel have similar values.

**Summation**: The products obtained after pixel-wise multiplication are then summed to calculate the correlation value for that position. This process is repeated for each position in the image.

**Kernel Size**: The size of the kernel or filter is an important parameter in correlation. Different kernel sizes can capture different levels of detail and features within the image.

**Centering Kernel**: Properly centering the kernel within the image is crucial to calculate the correlation accurately. The kernel should align with the region of interest in the image.

**Normalization**: In some cases, it is essential to normalize the correlation result to ensure that it remains within a certain range (e.g., between 0 and 1). This is often done to facilitate further processing.

**Applications**: Correlation is used in various image processing tasks, including template matching, pattern recognition, object detection, and feature extraction. It plays a vital role in locating objects or patterns within an image.

# IMPLEMENTATION DETAILS:

**# 1. Importing Libraries**

import **cv2** as **cv**

import **numpy** as **np**

import **matplotlib**.**pyplot** as **plt**

**# 2. Reading image and resizing**

img = **cv**.**imread**("mountains.png")

img = **cv**.**cvtColor**(img, **cv**.COLOR\_BGR2RGB) h = **int**(img.shape[0]\*0.4)

w = **int**(img.shape[1]\*0.4) img = **cv**.**resize**(img, (w, h))

**# 3. Defining kernels and performing correlation**

*# Define the correlation kernels (filters)*

kernel\_3x3 = **np**.**array**([[-1, 0, 1],

[-1, 1, 1],

[-1, 0, 1]], dtype=**np**.float32)

kernel\_5x5 = **np**.**array**([[-1, -1, 0, 1, 1],

[-1, -1, 0, 1, 1],

[-1, -1, 1, 1, 1],

[-1, -1, 0, 1, 1],

[-1, -1, 0, 1, 1]], dtype=**np**.float32)

kernel\_7x7 = **np**.**array**([[-1, -1, -1, 0, 1, 1, 1],

[-1, -1, -1, 0, 1, 1, 1],

[-1, -1, -1, 0, 1, 1, 1],

[-1, -1, -1, 1, 1, 1, 1],

[-1, -1, -1, 0, 1, 1, 1],

[-1, -1, -1, 0, 1, 1, 1],

[-1, -1, -1, 0, 1, 1, 1]], dtype=**np**.float32)

kernel\_9x9 = **np**.**array**([[-1, -1, -1, -1, -1, 0, 1, 1, 1, 1, 1],

[-1, -1, -1, -1, -1, 0, 1, 1, 1, 1, 1],

[-1, -1, -1, -1, -1, 0, 1, 1, 1, 1, 1],

[-1, -1, -1, -1, -1, 0, 1, 1, 1, 1, 1],

[-1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1],

[-1, -1, -1, -1, -1, 0, 1, 1, 1, 1, 1],

[-1, -1, -1, -1, -1, 0, 1, 1, 1, 1, 1],

[-1, -1, -1, -1, -1, 0, 1, 1, 1, 1, 1],

[-1, -1, -1, -1, -1, 0, 1, 1, 1, 1, 1],], dtype=**np**.float32)

*# Perform correlation using cv2.filter2D* correlation\_result\_3x3 = **cv**.**filter2D**(img, -1, kernel\_3x3) correlation\_result\_5x5 = **cv**.**filter2D**(img, -1, kernel\_5x5) correlation\_result\_7x7 = **cv**.**filter2D**(img, -1, kernel\_7x7) correlation\_result\_9x9 = **cv**.**filter2D**(img, -1, kernel\_9x9)

**# 4. Plotting the resultant images**

*# Plot the original image and the correlation results*

**plt**.**figure**(figsize=(10, 7))

**plt**.**suptitle**("Correlation Results", fontsize=16)

**plt**.**subplot**(2, 3, 1) **plt**.**imshow**(img) **plt**.**title**("Original Image")

**plt**.**subplot**(2, 3, 2) **plt**.**imshow**(correlation\_result\_3x3) **plt**.**title**("Correlation 3x3")

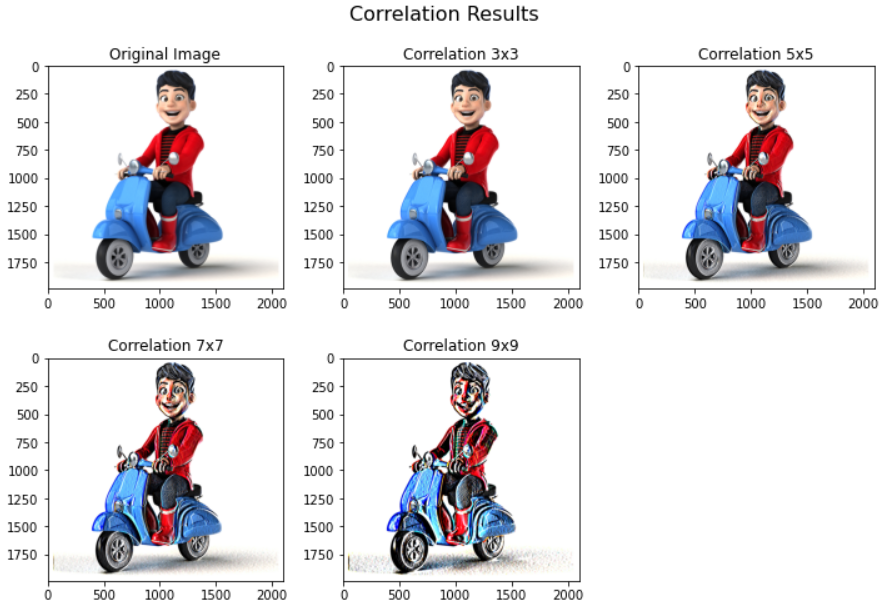
**plt**.**subplot**(2, 3, 3) **plt**.**imshow**(correlation\_result\_5x5) **plt**.**title**("Correlation 5x5")

**plt**.**subplot**(2, 3, 4) **plt**.**imshow**(correlation\_result\_7x7) **plt**.**title**("Correlation 7x7")

**plt**.**subplot**(2, 3, 5) **plt**.**imshow**(correlation\_result\_9x9) **plt**.**title**("Correlation 9x9")

**plt**.**tight\_layout**() **plt**.**show**()

# OUTPUT:



**Practical 6: Perform correlation and convolution on an image using a given filter and compare their outputs.**

# THEORETICAL DISCUSSION:

* **Correlation**:

1. *Mathematical Operation*: Correlation is a mathematical operation used to measure the similarity between an image and a filter (kernel).
2. *Symmetry*: In correlation, the kernel is not flipped or rotated; it remains in its original orientation when compared to the image.
3. *Purpose*: Correlation is often used in pattern matching and template matching to find instances of a template image within a larger image.
4. *Peak Response*: The output of correlation indicates where the template image best matches the larger image. High correlation values represent strong matches.
5. *Applications*: Correlation is used in applications like object detection, template matching, and optical character recognition (OCR).

* **Convolution**:

1. *Mathematical Operation*: Convolution is a mathematical operation used for filtering and feature extraction in image processing.
2. *Kernel Flipping*: In convolution, the kernel is flipped or rotated 180 degrees before being applied to the image.
3. *Purpose*: Convolution is used for tasks like blurring, sharpening, edge detection, and various other image filtering operations.
4. *Effect on Image*: Convolution can modify the image by enhancing or suppressing certain features based on the characteristics of the filter.
5. *Applications*: Convolution is widely used in image enhancement, noise reduction, and feature extraction in tasks such as computer vision and image processing.

In summary, correlation measures the similarity between an image and a template without flipping the kernel, while convolution applies a flipped kernel to the image for various filtering operations, which can alter the image's appearance.

# IMPLEMENTATION DETAILS:

**# 1. Importing Libraries**

import **cv2** as **cv**

import **numpy** as **np**

import **matplotlib**.**pyplot** as **plt**

**# 2. Reading image and resizing**

img = **cv**.**imread**("mountains.png")

img = **cv**.**cvtColor**(img, **cv**.COLOR\_BGR2RGB) h = **int**(img.shape[0]\*0.4)

w = **int**(img.shape[1]\*0.4) img = **cv**.**resize**(img, (w, h))

**# 3. Defining kernel and performing correlation and convolution**

*# Define the kernel (filter)*

kernel = **np**.**array**([[-3, 0, -1],

[0, 0.8, 0],

[1, 0, 3]], dtype=**np**.float32) kernel\_correlation = kernel

kernel\_convolution = **np**.**flipud**(**np**.**fliplr**(kernel))

*# Perform correlation and convolution using cv2.filter2D* correlation\_result = **cv**.**filter2D**(img, -1, kernel\_correlation) convolution\_result = **cv**.**filter2D**(img, -1, kernel\_convolution)

**# 4. Plotting the resultant images**

**plt**.**figure**(figsize=(10, 4))

**plt**.**suptitle**("Results", fontsize=16)

**plt**.**subplot**(1, 3, 1) **plt**.**imshow**(img) **plt**.**title**("Original Image")

**plt**.**subplot**(1, 3, 2) **plt**.**imshow**(correlation\_result) **plt**.**title**("Correlation")

**plt**.**subplot**(1, 3, 3) **plt**.**imshow**(convolution\_result) **plt**.**title**("Convolution")

**plt**.**tight\_layout**() **plt**.**show**()

# OUTPUT:

