National University of Computer & Emerging Sciences, Karachi
Artificial Intelligence-School of Computing
Fall 2024, Lab Manual - 02

Course Code (AI4002)	**Course: Computer Vision Lab**
Instructor(s):	Muhammad Khalid

Objectives:

1. Understand the basics of digital images and their representation.
2. Learn the concept of point processing and its application in modifying image pixel values.
3. Understand pixel transformations to enhance or alter image appearance.
4. Explore color spaces and their importance in digital image processing.
5. Understand color transformations and their applications in image enhancement.
6. Apply histogram equalization to improve image contrast and visibility

1. Digital Images and their representation

Digital images are visual representations stored and processed using computers. They consist of a grid of tiny colored dots called pixels. Each pixel holds information about color and intensity, allowing them to collectively create images.

Representation: Digital images are represented as matrices or grids of pixels. Each pixel is made up of color components, often red, green, and blue (RGB), which determine the pixel's color. The intensity of these components is usually represented by numbers ranging from 0 to 255.

Example: Consider a 3x3 pixel image using RGB. Each pixel is represented as (R, G, B) where each component can take values from 0 to 255.

Image:
(RGB) (RGB) (RGB)
(100, 0, 0) (0, 150, 0) (0, 0, 200)
(255, 255, 255) (128, 128, 128) (0, 0, 0)

In this example, the pixel at the top-left has a red value of 100, no green, and no blue, resulting in a reddish color. The pixel at the top-center is a shade of green, and the one at the top-right is a shade of blue. The pixel at the center is white because all RGB components are set to their maximum values. The pixel at the bottom-center is gray due to equal RGB components, and the one at the bottom-right is black with all RGB components set to 0.

```
import cv2
# Load an image from file
image_path = 'image.jpg'
image = cv2.imread(image_path)

# Convert to grayscale
gray_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)
```

```
# Access pixel value at coordinates (row, column)
row = 100
column = 150
pixel_value = image[row, column]

# Split the image into color channels
blue_channel, green_channel, red_channel = cv2.split(color_image)

# Display images side by side
stacked_images = cv2.hconcat([image, cv2.cvtColor(gray_image, cv2.COLOR_GRAY2BGR)]), # add images as per your choice

# Display the image
cv2.imshow('Loaded Image', image)
cv2.waitKey(0)  # Wait for a key press
cv2.destroyAllWindows()  # Close all windows
```

2. Point processing and its application in modifying image pixel values

Point processing is a fundamental image processing technique that involves modifying individual pixel values in an image without considering their neighboring pixels. It's a simple and powerful method used to enhance or adjust the appearance of images.

Application of point processing in modifying image pixel values:

- **Brightness Adjustment:** By uniformly adding or subtracting a constant value to all pixel intensities, the overall brightness of an image can be adjusted. Increasing the value makes the image brighter, while decreasing it makes the image darker.
- **Contrast Enhancement:** Contrast refers to the difference between the lightest and darkest parts of an image. Stretching the range of pixel values can enhance contrast, making details more visible.
- **Thresholding:** This involves converting a grayscale image into a binary image by setting pixels above a certain threshold to white and those below to black. It's often used for image segmentation.
- **Color Adjustment**: For color images, point processing can be applied independently to each color channel (R, G, B) to modify the overall color balance.
- **Negative Image:** Inverting pixel values by subtracting them from the maximum value can create a negative version of the image.
- **Gamma Correction:** Adjusting pixel values using a gamma function can modify the image's overall brightness and contrast to better match human perception.
- **Histogram Equalization:** This technique redistributes pixel intensities to enhance the image's contrast, making it visually more appealing.
- **Color Filtering:** Applying point processing on color channels can selectively emphasize or suppress specific colors in an image.

```
import cv2
import numpy as np

# Load an image from file
image_path = 'image.jpg'
image = cv2.imread(image_path)

# Define a brightness adjustment factor
brightness_factor = 50

# Apply the point transformation for brightness adjustment
brightened_image = np.clip(image + brightness_factor, 0, 255)

# Define contrast enhancement parameters
alpha = 1.5
beta = 50

# Apply the point transformation for contrast enhancement
enhanced_image = np.clip(alpha * gray_image + beta, 0, 255).astype(np.uint8)

# Apply histogram equalization
equalized_image = cv2.equalizeHist(gray_image)

# Apply binary thresholding
_, binary_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY)




# Display the original and brightened images side by side
stacked_images = np.hstack([image, brightened_image])
cv2.imshow('Original vs Brightened', stacked_images)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. Pixel transformations to enhance or alter image appearance

Pixel transformations are operations applied to individual pixels in an image to enhance or alter its appearance. These transformations can change pixel values based on certain rules or mathematical functions, leading to various visual effects. Here are some examples of pixel transformations:

- **Logarithmic Transformation:**
  Explanation: A logarithmic transformation enhances the darker regions of an image while compressing the brighter regions. It's useful for revealing details in shadowy areas.
  Mathematically: New pixel value = c * log(1 + old pixel value)
  Example: Consider an image where the pixel values are mostly concentrated in the darker range. Applying a logarithmic transformation can enhance these dark areas, revealing more details.

- **Power-Law Transformation (Gamma Correction):**
  Explanation: A power-law transformation adjusts the overall brightness and contrast of an image using a power function. It's commonly used for gamma correction.
  Mathematically: New pixel value = c * (old pixel value ^ γ)
  Example: In a photograph that looks overly bright or washed out, applying a power-law transformation with a gamma value less than 1.0 can correct the appearance by enhancing midtones and shadows.

- **Piecewise Linear Transformation:**
  Explanation: A piecewise linear transformation involves dividing the input intensity range into segments and applying different linear transformations to each segment. It's useful for adjusting specific ranges of pixel values.
  Mathematically: For each segment, calculate new pixel value = (old pixel value - a) * (new_max - new_min) / (old_max - old_min) + new_min
  Example: Suppose an image has intensity values predominantly in the range [0, 100]. You can apply a piecewise linear transformation to stretch these values across the full range [0, 255] to enhance contrast.

  In all these transformations, **'c', 'γ', 'a', 'old pixel value', 'new pixel value', 'old_max', and 'old_min'** are constants or variables defined by the transformation technique. '**new_max**' **and** '**new_min**' represent the desired range of pixel values after transformation.

- **Brightness Adjustment:**
  Transformation: New pixel value = Old pixel value + A
  Example: Increasing the pixel values of all pixels in an image by a constant value A will make the image brighter.

- **Contrast Stretching:**
  Transformation: New pixel value = (Old pixel value - Min) * (255 / (Max - Min))
  Example: Stretching the pixel value range from Min to Max to the full range (0 to 255) enhances image contrast.

- **Gamma Correction:**
  Transformation: New pixel value = 255 * (Old pixel value / 255) ^ γ
  Example: Applying a gamma value less than 1.0 brightens the midtones and darkens the shadows, while a gamma value greater than 1.0 has the opposite effect.

- **Thresholding:**
  Transformation: New pixel value = 0 or 255 based on a threshold
  Example: Pixels above a certain threshold become white, and those below become black, resulting in a binary image.

- **Color Balance Adjustment:**
  Transformation: New pixel value for each channel = Old pixel value * Gain
  Example: Increasing the gain for a specific color channel in each pixel can shift the color balance.

- **Negative Image:**
  Transformation: New pixel value = 255 - Old pixel value
  Example: Inverting pixel values results in a negative version of the image.

- **Histogram Equalization:**
  Transformation: Apply a mapping function based on the image's cumulative histogram to spread out pixel values.
  Example: Enhancing image contrast by redistributing pixel values more uniformly.

```
import cv2
import numpy as np

# Load a grayscale image from file
image_path = 'gray_image.jpg'
gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Define the logarithmic transformation parameters
c = 255 / np.log(1 + np.max(gray_image))

# Apply the logarithmic transformation
log_transformed_image = c * np.log(1 + gray_image)

# Convert to uint8 type for display
log_transformed_image = np.array(log_transformed_image, dtype=np.uint8)

# Define the gamma correction parameters
gamma = 0.5

# Apply the power-law transformation (gamma correction)
gamma_corrected_image = np.power(gray_image, gamma)

# Normalize pixel values to [0, 255]
gamma_corrected_image = np.clip(gamma_corrected_image * (255 / np.max(gamma_corrected_image)), 0, 255).astype(np.uint8)
```

```
# Define the piecewise linear transformation parameters
input_min = np.min(gray_image)
input_max = np.max(gray_image)
output_min = 0
output_max = 255

# Apply the piecewise linear transformation (contrast stretching)
piecewise_transformed_image = ((gray_image - input_min) / (input_max - input_min)) * (output_max - output_min) + output_min

# Convert to uint8 type for display
piecewise_transformed_image = np.array(piecewise_transformed_image, dtype=np.uint8)

# Define a brightness adjustment factor
brightness_factor = 50

# Apply the point transformation for brightness adjustment
brightened_image = np.clip(image + brightness_factor, 0, 255)


# Define contrast enhancement parameters
alpha = 1.5
beta = 50

# Apply the point transformation for contrast enhancement
enhanced_image = np.clip(alpha * gray_image + beta, 0, 255).astype(np.uint8)




# Display the original and logarithmically transformed images side by side
stacked_images = cv2.hconcat([gray_image, log_transformed_image])
cv2.imshow('Original vs Log Transformed', stacked_images)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4. Color Spaces and their importance in digital image processing

Color spaces are standardized systems that represent colors in a way that facilitates their reproduction in various devices and contexts. They provide a structured method to define colors using coordinates or values. Different color spaces are designed to capture various aspects of color, such as luminance, chrominance, and perceptual uniformity.

Examples of color spaces and their importance in digital image processing:

- **RGB (Red-Green-Blue):**

  Explanation: Represents colors as combinations of red, green, and blue light intensities. Commonly used in displays like monitors and digital cameras.
  Importance: Essential for digital imaging because it directly corresponds to how images are displayed on screens and captured by cameras. RGB values can be manipulated for color correction, enhancement, and artistic effects.

- **CMY (Cyan-Magenta-Yellow) and CMYK (Cyan-Magenta-Yellow-Black):**
  Explanation: CMY represents colors subtractively, used in color printing and mixing pigments. CMYK extends CMY by adding a black channel for better color reproduction in printing.
  Importance: Crucial for print design and reproduction, as they determine how colors mix on paper. Conversion between RGB and CMYK is necessary for accurate color rendering in printed materials.

- **HSV or HSB (Hue-Saturation-Value/Brightness):**
  Explanation: Represents colors based on their perceptual attributes - hue (color tone), saturation (vibrancy), and value/brightness (intensity).
  Importance: HSV separates color information from brightness information, making it useful for tasks like color correction, image segmentation, and color-based object detection.

- **LAB (CIELAB):**
  Explanation: A perceptually uniform color space that separates luminance (L) from color information (A and B channels), allowing consistent color differences.
  Importance: Useful in color analysis, image quality assessment, and color manipulation where accurate perceptual differences are crucial, like in image compression or color correction.

- **YUV or YCbCr:**
  Explanation: YUV separates color information (U and V channels) from luminance (Y), used in analog TV broadcasting and digital video compression.
  Importance: YUV allows efficient video compression since human perception is more sensitive to changes in luminance than color. It's also used for video processing and transmission.

```
import cv2
import numpy as np

#Color Balance Adjustment
# Load a color image from file
image_path = 'color_image.jpg'
color_image = cv2.imread(image_path)

# Define color balance adjustment factors for each channel
blue_scale = 1.2
green_scale = 1.0
red_scale = 0.9
```

```
# Apply the color balance adjustment
adjusted_image = np.clip(color_image * [blue_scale, green_scale, red_scale], 0, 255).astype(np.uint8)

#Color map
# Apply a colormap to the grayscale image
colormap_image = cv2.applyColorMap(gray_image, cv2.COLORMAP_JET)

#Channel Swapping
# Swap red and blue channels
swapped_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2RGB)

#Color Inversion
# Invert the colors of the image
inverted_image = cv2.bitwise_not(color_image)

#Multimodel
# Multi-modal fusion with weighted overlay
weight_xray = 0.6
weight_mri = 0.4
fused_image = cv2.addWeighted(colormap_xray, weight_xray, colormap_mri, weight_mri, 0)
```

Note: cv2.applyColorMap() in OpenCV supports a variety of predefined colormaps (color mapping schemes) that you can use to apply different color representations to your images. Some of the common predefined colormaps available in OpenCV are:

- **COLORMAP_JET:** The "jet" colormap provides a smooth transition through blue, cyan, green, yellow, and red colors. It's often used to represent scalar values.
- **COLORMAP_AUTUMN:** The "autumn" colormap represents a range of colors from dark red to bright yellow.
- **COLORMAP_BONE**: The "bone" colormap is a grayscale colormap with a linear gradient.
- **COLORMAP_COOL**: The "cool" colormap transitions through shades of cyan and blue.
- **COLORMAP_HOT:** The "hot" colormap represents a range of colors from black to bright red.
- **COLORMAP_HSV**: The "hsv" colormap is based on the hue-saturation-value color space.
- **COLORMAP_RAINBOW**: The "rainbow" colormap spans the entire visible spectrum with a transition from blue to red via green and yellow.
- **COLORMAP_SPRING:** The "spring" colormap starts from magenta and transitions to yellow.
- **COLORMAP_SUMMER:** The "summer" colormap transitions from green to yellow.
- **COLORMAP_WINTER:** The "winter" colormap transitions from blue to green.

Task:

1. **Enhancing and Analyzing Medical Image Quality**

   You are a junior researcher at a medical imaging lab, and your team has been working on improving the quality of medical images for accurate diagnosis and analysis. The lab has received a set of grayscale X-ray images from a hospital, and your task is to enhance the images using various techniques and analyze their impact on medical diagnosis. The images suffer from poor contrast and visibility issues, making it challenging for doctors to identify subtle details. You need to perform a series of operations to enhance the images and demonstrate their effectiveness.

- ✓ **Load and Display**: Start by loading and displaying one of the X-ray images using OpenCV to get a sense of the image quality.
- ✓ **Contrast Enhancement:** Apply histogram equalization to enhance the contrast and visibility of the X-ray image. Compare the enhanced image with the original and note any improvements in terms of details and contrast.
- ✓ **Color Mapping**: Convert the enhanced grayscale image to a color-coded heatmap using the "jet" colormap. Visualize how this color mapping can help doctors identify intensity variations more effectively.
- ✓ **Color Balance:** The X-ray images might have color casts due to variations in lighting during capture. Apply a color balance adjustment to remove the casts and restore the true color representation.
- ✓ **Color Filtering**: Since X-ray images may exhibit specific color patterns due to different tissue densities, filter out a specific color range that corresponds to a particular density. Visualize the filtered image to highlight specific areas of interest.
- ✓ **Logarithmic Transformation:** Apply a logarithmic transformation to enhance the visibility of darker areas in the image, which might contain critical details.
- ✓ **Power-Law Transformation:** Use a power-law transformation to fine-tune the contrast of the enhanced image, further improving details' visibility.

2. **Enhancing Multi-Modal Medical Image Fusion for Precise Diagnostics**

   You are now a senior researcher at a cutting-edge medical imaging institute. Your team is tasked with improving diagnostic accuracy by fusing multiple imaging modalities—X-ray and MRI scans—into a single enhanced image. This combined approach can provide comprehensive insights into both anatomical structures and tissue properties. However, the integration is complex due to varying contrasts and spatial resolutions in the modalities. Your goal is to apply the techniques you've learned to perform fusion and enhance visibility for doctors.

- ✓ **Load Modalities:** Load an X-ray and an MRI image, both capturing the same anatomical region, using OpenCV.
- ✓ **Histogram Equalization for Each Modality:** Apply histogram equalization separately to the X-ray and MRI images to enhance their respective contrasts.
- ✓ **Color Mapping and Fusion:** Convert the enhanced X-ray and MRI images to colored heatmaps using "jet" colormaps. Overlay the colored MRI heatmap onto the X-ray heatmap, creating a fused image.
- ✓ **Multi-Modal Weighted Fusion**: Adjust the weighting of the X-ray and MRI images during fusion to enhance features from each modality optimally. Use more weight for the modality with higher resolution, and less weight for the other.
- ✓ **Logarithmic and Power-Law Transformations:** Apply logarithmic and power-law transformations to the fused image to improve visibility of subtle structures and enhance contrast.
- ✓ **Comparative Analysis:** Compare the original X-ray and MRI images with the fused image to evaluate the diagnostic benefits of the multi-modal fusion approach.

3. Real-Time Video Enhancement and Analysis

In this task, you will capture a live video stream from your camera and apply a series of image enhancement operations to each frame in real-time. You will then display the original video stream along with individual frames enhanced using various techniques. This will allow you to observe the immediate effects of each operation on the video feed.

Video Capture Setup:

Initialize the camera for video capture using OpenCV.
Configure the camera settings (resolution, frame rate, etc.) if needed.

Real-Time Video Processing Loop:

Start a loop to continuously capture video frames from the camera.
For each captured frame:
Apply histogram equalization for contrast enhancement.
Convert the frame to a colored heatmap using the "jet" colormap.
Apply color balance adjustment to correct any color casts.
Apply a logarithmic transformation to enhance visibility in dark regions.
Apply a power-law transformation for fine-tuning contrast.
Display the original live video stream.
Display individual frames with each operation applied side by side for comparison.