LAB-1

Digital Image Processing (EE-608P)

Aim: -To perform Image sampling and quantization using Python.

Explanation:

Image Sampling and Quantization

Image processing involves three fundamental steps for converting an analog image into a digital image: **sampling**, **quantization and encoding**.

1. Image Sampling

- **Definition:** Sampling refers to selecting discrete points from a continuous image. It determines the **spatial resolution** of the image.
- **How It Works:** An image is divided into a grid of pixels, and each pixel represents a small portion of the image.
- **Effect of Sampling:**
  - Higher sampling rates (more pixels) result in finer details and better image quality.
  - Lower sampling rates lead to blocky, pixelated images.

Example of Sampling

- **High Sampling Rate**: 1920×1080 pixels (Full HD) → Clearer image
- **Low Sampling Rate**: 100×100 pixels → Blurry image

2. Image Quantization

- **Definition:** Quantization is the process of mapping the continuous range of pixel intensity values to a finite set of levels. It determines the **bit depth** (number of possible intensity values per pixel).
- **How It Works:** Each sampled pixel's intensity **value is approximated to the nearest available level.**
- **Effect of Quantization:**
  - Higher quantization levels (more bits per pixel) result in smoother transitions and more color detail.
  - Lower quantization levels cause noticeable **color banding** (loss of detail in smooth gradients).

Example of Quantization

- **8-bit quantization** → 256 levels (grayscale: 0-255) → Smooth shading
- **2-bit quantization** → 4 levels (0, 85, 170, 255) → Posterized image.

Relationship Between Sampling and Quantization

- **High Sampling + High Quantization** → Best quality, large file size
- **Low Sampling + Low Quantization** → Poor quality, small file size
- **Balanced Sampling & Quantization** → Optimized quality and file size

Output:

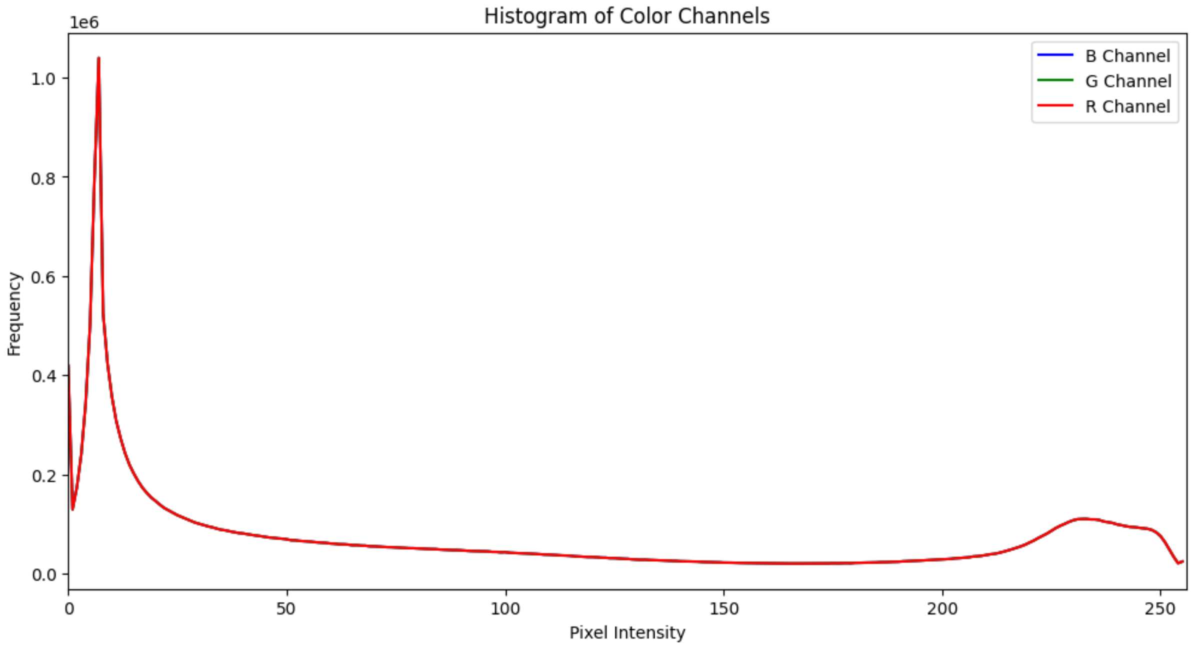
Aim: -2 Analysis of intensity, special resolution and histogram of colour channels of an image using python.

Explanation of the Analysis:

1. **Intensity Distribution (Grayscale Image):**
   - Shows how pixel intensities are distributed.
   - Brighter areas have higher intensity values.
2. **Spatial Resolution:**
   - The script generates **2x and 4x lower resolutions** to visualize the loss of detail.
   - Helps in understanding how reducing resolution affects image clarity.
3. **Histogram of Color Channels:**

- Shows the intensity distribution for **red, green, and blue** channels.
- Helps analyse color balance and contrast.

Outputs:



Aim: -3 Perform Intensity transformation of images using Python.

Explanation:

Intensity Transformations of Images

Intensity transformations are used in image processing to adjust the pixel values for contrast enhancement, brightness adjustments, or feature enhancement.

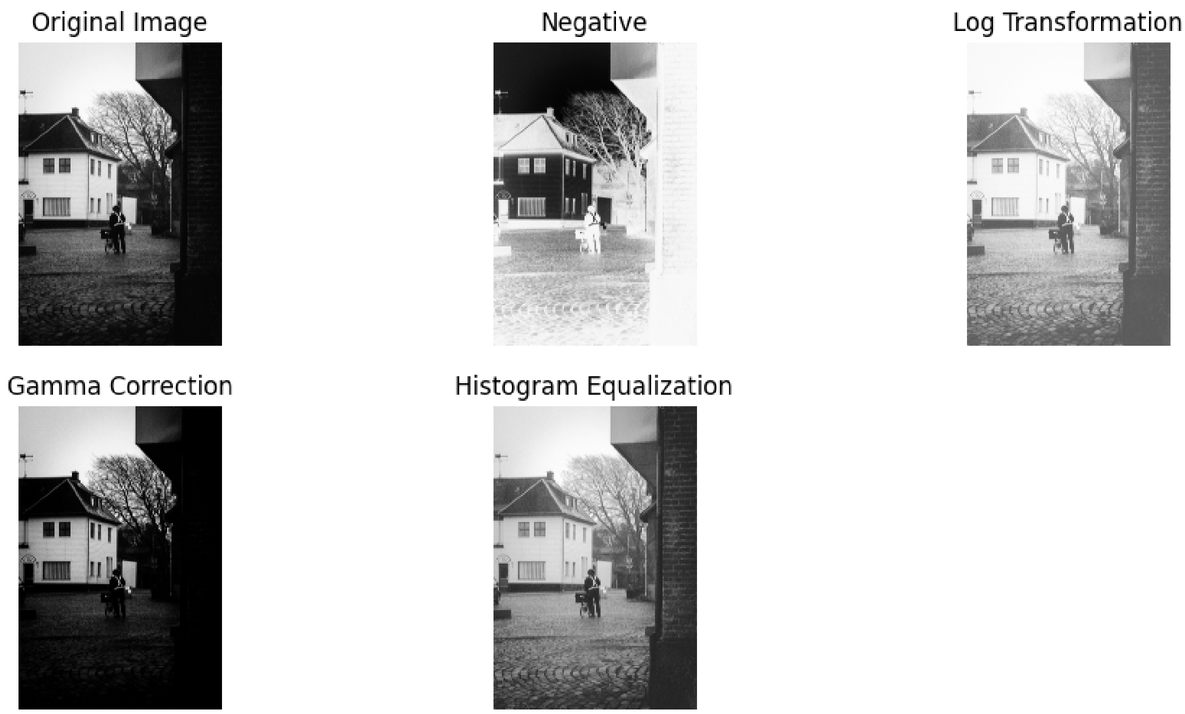
Types of Intensity Transformations

1. **Linear Transformations:**
    - Contrast stretching
    - Negative transformation
    - Brightness adjustment
2. **Non-Linear Transformations:**
    - Log transformation
    - Power-law (gamma) transformation
3. **Histogram-Based Transformations:**
    - Histogram equalization
    - Adaptive histogram equalization

Explanation of Transformations

1. **Negative Transformation (255 - pixel value)**
    - Inverts pixel intensities (useful for medical images).
2. **Log Transformation (c * log(1 + pixel))**
    - Enhances darker regions while compressing bright areas.
3. **Gamma Correction (pixel^(gamma))**
    - Adjusts brightness non-linearly.
    - **Gamma < 1:** Brightens the image.
    - **Gamma > 1:** Darkens the image.
4. **Histogram Equalization**
    - Redistributes pixel intensities for better contrast.

Output:



Aim: -4 Perform Reconstruction of an image Using Python.

Explanation:

Image Reconstruction in Image Processing

Image reconstruction refers to the process of restoring or rebuilding an image from incomplete, noisy, or distorted data. Some common techniques include:

1. **Inverse Filtering** – Used for deblurring images.
2. **Image Inpainting** – Filling missing parts of an image.
3. **Interpolation Methods** – Used to reconstruct images from down sampled versions.
4. **Fourier Transform-Based Reconstruction** – Uses frequency domain techniques.

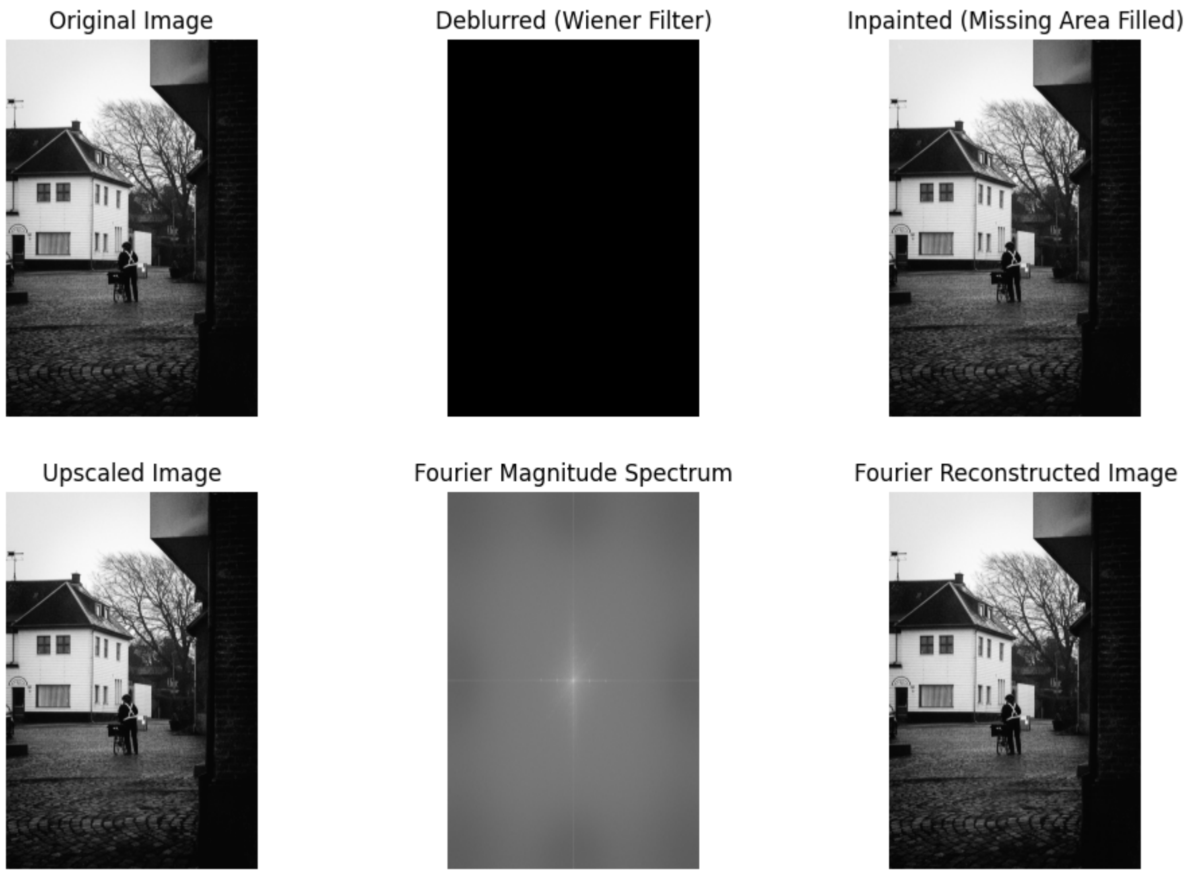
I'll demonstrate multiple **image reconstruction techniques** using Python:

1. **Inverse Filtering** – Deblurring an image.
2. **Image Inpainting** – Filling missing parts.
3. **Interpolation-Based Reconstruction** – Upscaling a low-resolution image.
4. **Fourier Transform-Based Reconstruction** – Reconstructing an image using frequency components.

Explanation of the Techniques

1. **Inverse Filtering (Wiener Filter)**
   - Used for **deblurring** images affected by motion blur or noise.
   - The Wiener filter restores an image using a frequency-domain approach.
2. **Image Inpainting**
   - Used to **fill missing areas** using surrounding pixels.
   - OpenCV's cv2.INPAINT_TELEA method smoothly reconstructs missing parts.
3. **Interpolation-Based Reconstruction**
   - Used to **upscale low-resolution images**.
   - The cv2.INTER_CUBIC method provides high-quality upscaling.
4. **Fourier Transform-Based Reconstruction**
   - Analyses image frequencies and **reconstructs the image** using inverse Fourier transform.
   - The **magnitude spectrum** shows high-frequency and low-frequency components.

Output: Masking and Inpainting:



Aim: -5 For the given images, apply decimation (down sampling), nearest neighbour interpolation, bilinear interpolation and bicubic interpolation. Analyse results.