DIP Assignment-2 Report

(Kedarnath P - 20902)

(1) Image Display

Inferences Drawn from Observations:

- **Given Code**: Loads a grayscale image and displays it using `matplotlib`'s `imshow` function.
- Displayed Image size: (796x600). Which Indicates that it is a Grayscale Image with single channel.
- Displayed Image datatype: (uint8). Which Indicates that pixel values lie between [0, 255].
- The original input grayscale image, when displayed using `imshow`, undergoes an autoscaling of pixel values, potentially altering its appearance.
- This autoscaling can lead to contrast-stretching effects if the pixel values don't span the entire expected range.

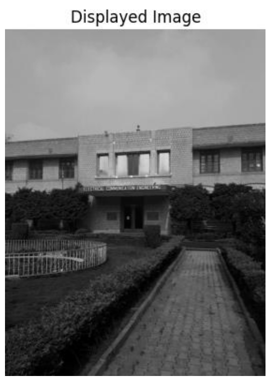
Autoscaling in `matplotlib`'s `imshow` Function:

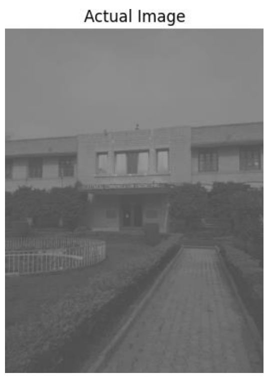
- `imshow` autoscales the contrast of single-channel images by default.
- This means pixel values are stretched or shrunk to span the entire colormap.
- For instance, in grayscale images, the minimum pixel value might be mapped and displayed as black(0) and the maximum pixel value as white(255), with other values linearly interpolated in between.

Two Ways to Prevent Contrast Stretching (Autoscaling) in `imshow`:

1. **Explicit Value Range**: Check the image data type and display the image with appropriate `vmin` and `vmax` values to prevent contrast stretching.
    - Determine the image's data type (`float64` or `uint8`).
    - Set the `vmin` and `vmax` parameters of `imshow` accordingly.
    - For `float64`, use [0, 1]. For `uint8`, use [0, 255].
    - Displayed Image size: (796x600)

2. **Conversion to RGB**: Convert the grayscale image to an RGB format by replicating its values across three channels, and then displays the resulting output image. As a result, the image will still appear grayscale to the eye, but it will be represented in memory as an RGB image.
    - Convert the grayscale image to an RGB format.
    - This is done by replicating the grayscale values across the R, G, and B channels.
    - `imshow` doesn't autoscale the contrast for multi-channel images, so the RGB image appears as expected.
    - Displayed Image size: (796x600x3)





(2). Contrast Stretching

Implementation:

- **Gamma Transformation**:
  - The function `gamma_transform` takes an image and a gamma value as inputs.
  - The image is normalized to the range [0, 1] by dividing every pixel by 255.
  - Gamma correction is applied using the formula I^{γ}.
  - The image is then de-normalized back to the range [0, 255].
- **Histogram Equalization**:
  - The function `histogram_equalization` takes an image as input.
  - A histogram of the image is computed.
  - The cumulative distribution function (CDF) of the histogram is calculated.
  - The CDF is then equalized to spread out the pixel values in the image.
  - Pixels in the original image are mapped to their corresponding equalized values to obtain the histogram-equalized image.
- **Mean Squared Error (MSE)**:
  - The function `calculate_mse` calculates the mean squared error between two images. It's used to compute the difference between the histogram-equalized image and the gamma-transformed image for various gamma values.
- **Finding Optimal Gamma**:
  - A range of gamma values between 0.1 and 5 is sampled.
  - For each gamma value, the image is transformed, and its MSE with respect to the histogram-equalized image is calculated.
  - The gamma value that gives the minimum MSE is chosen as the optimal gamma.
- **Visualization**:
  - The histograms of the original, histogram-equalized, and optimally gamma-transformed images are also plotted.

Observations:

- The original image appears to be hazy with poor contrast.
- Histogram equalization enhances the image by spreading out the pixel intensities, thereby improving the contrast.
- The gamma transformation, especially with the optimal gamma value, also enhances the contrast in the image, but it seems slightly less aggressive than histogram equalization. However, the enhancement is more gentle than the histogram equalization, preserving more of the original characteristics of the image.

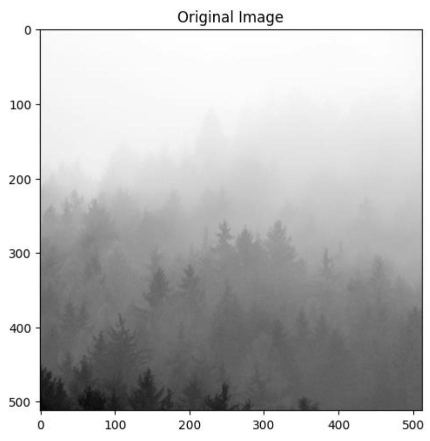
Inference from Observations:

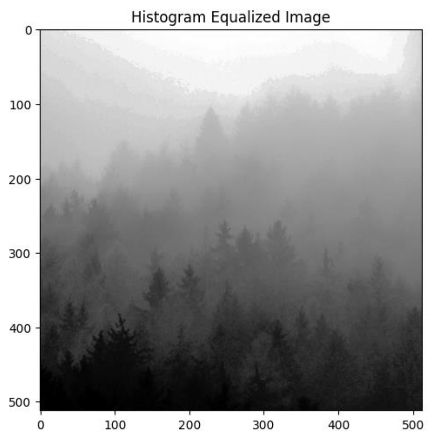
- **Histogram Analysis**:
  - The histogram of the original image is concentrated in a narrow range, indicating low contrast.
  - Histogram equalization spreads out the pixel intensities, resulting in a more uniform distribution across the entire range.
  - The histogram of the optimally gamma-transformed image is also spread out but it's slightly skewed towards the darker intensities and retains some of the original shape, suggesting a balanced contrast enhancement.

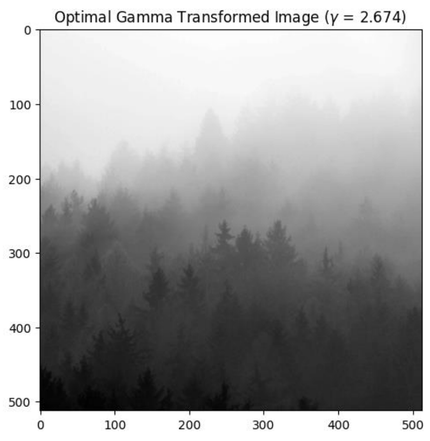
- **Significance of Gamma Values**:
    - Gamma values control the degree of transformation.
    - If $\gamma < 1$: The image will be transformed in such a way that it will enhance the darker regions of the image, making it look brighter.
    - If $\gamma = 1$: The image will remain unchanged.
    - If $\gamma > 1$: The image will be transformed to enhance the brighter regions, making it look darker.

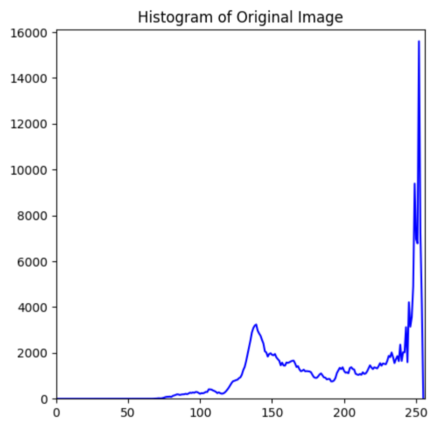
- **Use of MSE for Optimal Gamma**:
    - MSE measures the difference between two images. By computing the MSE between the gamma-transformed image and the histogram-equalized image, we aim to find a gamma transformation that closely resembles the effect of histogram equalization.
    - The gamma value that minimizes the MSE gives a transformation that's most similar to histogram equalization in terms of pixel intensity distribution.
    - The optimal gamma was found to be equal to **`2.6737373737373374`.**

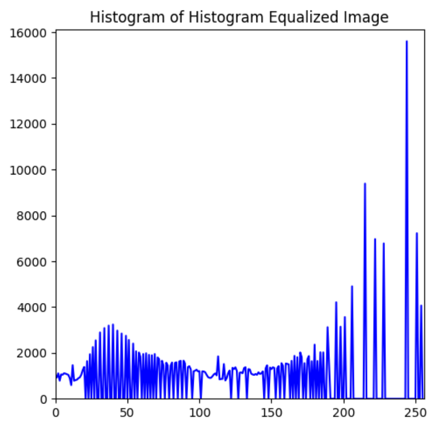
- **Histogram Equalization vs. Gamma Transformation**:
    - If the **goal** is to enhance as many details as possible, the **Histogram Equalized Image** might be preferred. (Applications: Medical or Satellite Imaging)
    - If the **goal** is a balance between enhancement and preserving a natural look, the **Optimal Gamma Transformed Image** might be the choice. (Applications: Photography)

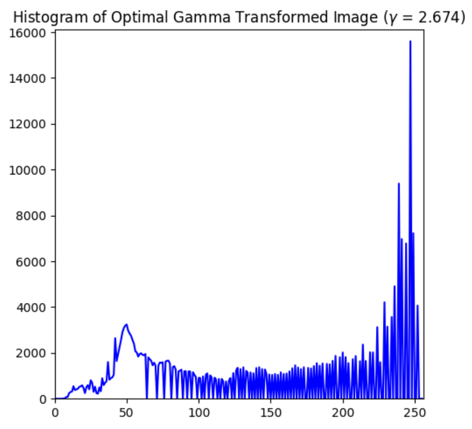












(3). Image Rotation

Implementation:

- **Steps involved inside the `rotate_image` Function:**
    - Convert the rotation angle from degrees to radians.
    - Determine the cosine and sine values for the rotation angle.
    - Acquire the height and width of the image.
    - Determine the center of the image.
    - Compute the new coordinates of the image corners after rotation.
    - Calculate the dimensions of the new bounding box for the rotated image.
    - Initialize the rotated image with a white background.
    - For every pixel in the new image, calculate the corresponding position in the original image.
        - If `nearest` is chosen as the interpolation method, the nearest neighboring pixel's value is used.
        - If `bilinear` is chosen, the weighted average of the four nearest pixels' values is used.
- **Post `rotate_image` Function Definition:**
    - The grayscale image is loaded.
    - The image is rotated 5° clockwise and 30° counterclockwise using both nearest neighbor and bilinear interpolation methods.
    - The original and rotated images are displayed.

Observations:

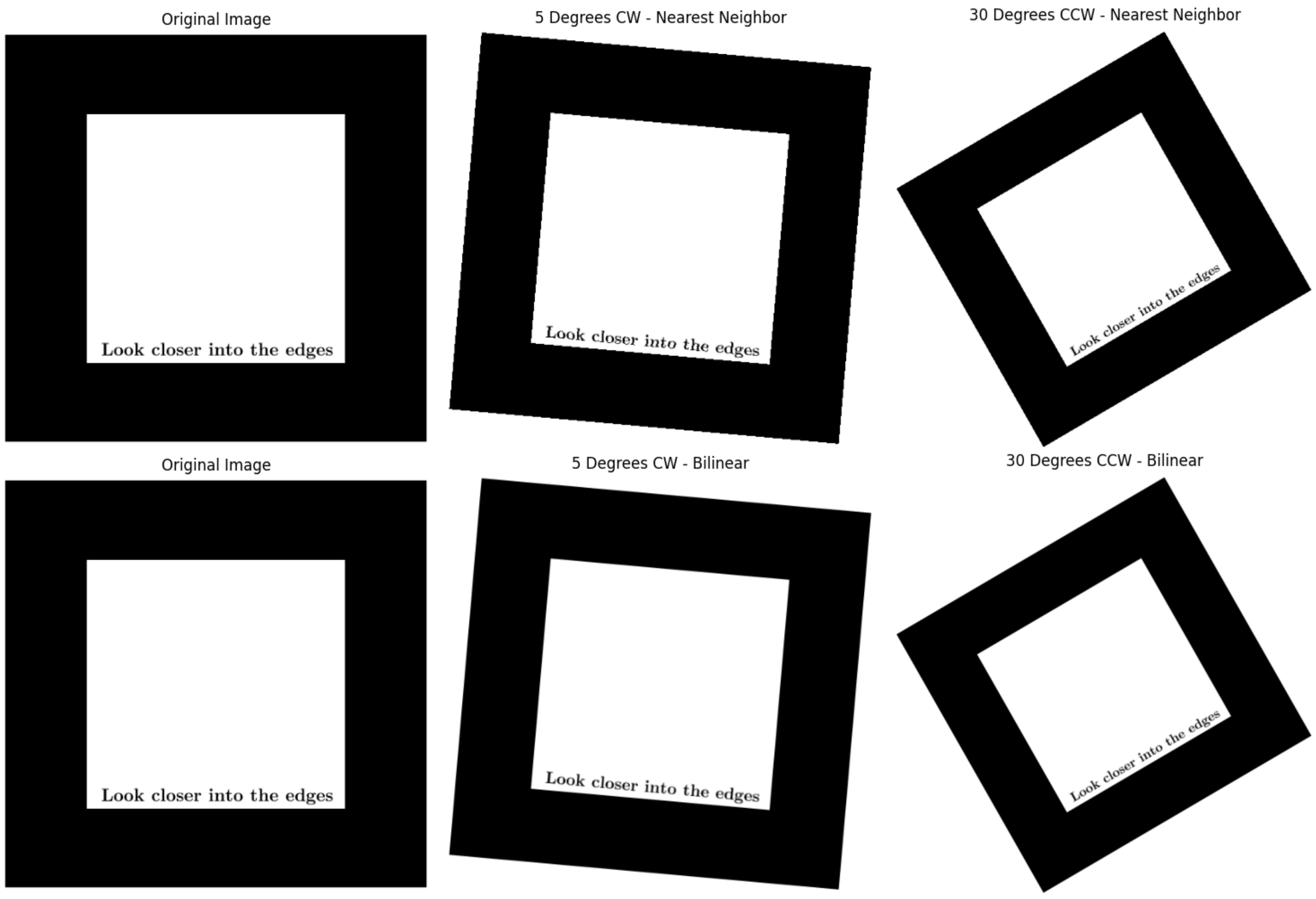
- **5° Clockwise Rotation:**
    - Both interpolation methods keep the image recognizable.
    - Nearest Neighbor shows slight jaggedness.
    - Bilinear yields a smoother edge.
- **30° Counter-Clockwise Rotation:**
    - The effects are more pronounced due to the greater rotation angle.
    - The jaggedness in Nearest Neighbor becomes more noticeable, especially around the corners and edges.
    - Bilinear provides a smoother transition but at the cost of minor blurring.

Inference from Observations:

- **Introduction:**
    - Image rotation is a transformation that repositions pixels based on a given angle.
    - Rotation can lead to gaps which require an interpolation method to fill.
    - Interpolation methods used are Nearest Neighbor and Bilinear interpolation.

- **Nearest Neighbor vs. Bilinear Interpolation:**
    - **Nearest Neighbor:**
        - Simpler and faster.
        - Chooses the value of the nearest neighboring pixel.
        - Produced jagged or stair-stepped edges (upon taking a closer look by zooming).
    - **Bilinear Interpolation:**
        - A bit more complex and slower.
        - Takes a weighted average of the 4 nearest pixel values.
        - Produces smoother images but introduced slight blurring (upon zooming).

- **Comparison to Original Image:**
    - The rotated images have been expanded to fit the entire rotated content, ensuring no cropping.
    - The original image has sharp and defined edges, while the rotated images might distortions due to the interpolation methods.


Original Image
5 Degrees CW - Nearest Neighbor
30 Degrees CCW - Nearest Neighbor
Look closer into the edges
Original Image
5 Degrees CW - Bilinear
30 Degrees CCW - Bilinear
Look closer into the edges

(4). Spatial Filtering

Implementation:

- **Average Filter Function** (`average_filter`):
    - This function applies an average filter to the image using the specified kernel size, resulting in a blurred version of the original image.
- **High-Boost Filter Function** (`high_boost_filter`):
    - The function first blurs the input image using the `average_filter` function.
    - Then, it calculates the mask, which represents the details or high-frequency components of the image.
    - It uses the mask and scaling constant k to sharpen the image.
- **Procedure**:
    - The original image is loaded and converted to grayscale.
    - For the first scenario, the input image is blurred with a kernel size of 5, this blurred image is subtracted from input image to get the mask and then the input image is sharpened using high-boost filtering (by adding k times the mask to input image).
    - For the second scenario, the original image is first blurred with a kernel size of 3. This blurred image is treated as input image now and it undergoes further processing: same as the first scenario.
- **Visualization**:
    - Relevant images, including the input, blurred versions, mask, and sharpened outputs, are displayed for both scenarios.

Observations:

- The original image provides a reference for comparing the effects of sharpening.
- The blurred images (from average filtering) demonstrate a loss of detail, especially noticeable at edges and in textured areas.
- The sharpened images exhibit enhanced details and contrast, especially around edges.
- After applying the high-boost filter on the blurred image, the sharpening effect is less pronounced compared to the first case where we directly applied high-boost filtering on the original image.

Inference from Observations:

- **Difference in Unsharp Mask between two cases**:
    - The unsharp mask for the direct high-boost approach on the original image retains more details compared to the one from the second scenario. This is because the second approach first blurs the image, which leads to the loss of some details even before the high-boost filtering process.

- **Significance of k Value**:
    - The scaling constant k amplifies the high-pass component (or mask) of the image.
    - A higher k would result in a more pronounced sharpening effect, whereas a lower k would yield subtler results.

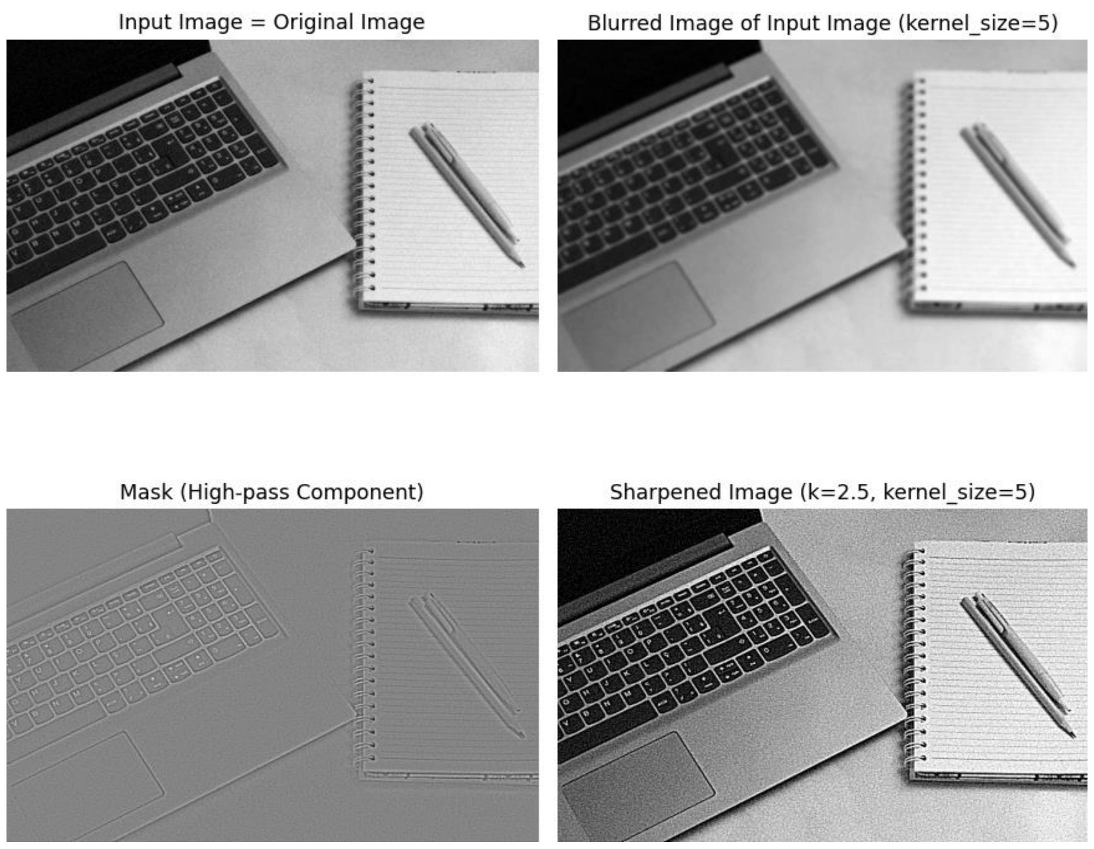
- **Intention Behind the Unsharp Mask Formula**:
    - The formula for unsharp mask, which involves subtracting the blurred image from the original, isolates the details or high-frequency components (edge part) of the image.
    - This mask can then be used to enhance the details in the original image.

- **High-Boost Filtering Formula**:
  - The high-boost filtering formula involves adding k times the mask to the original image. This step amplifies the details, resulting in a sharpened image.
  - The intention is to boost or emphasize the high-frequency components to enhance details and improve image sharpness.

Conclusions:

- The high-boost filtering method effectively enhances the details in the image.
- The amount of sharpening can be controlled using the scaling constant k.
- Directly applying high-boost filtering to the original image yields sharper results compared to when applied to an image that has been previously blurred.
- The unsharp mask is crucial for isolating image details, and its effectiveness can vary based on the degree of blurring applied to the image.

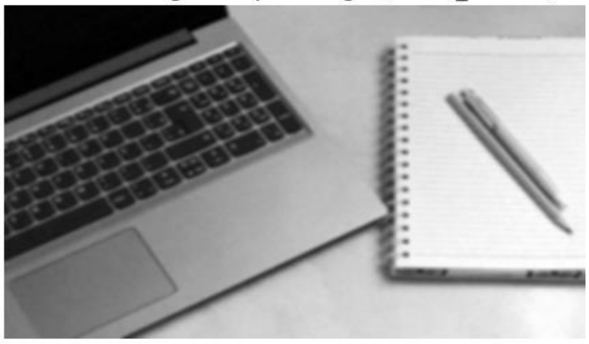
Scenario 1


Input Image = Original Image
Blurred Image of Input Image (kernel_size=5)
Mask (High-pass Component)
Sharpened Image (k=2.5, kernel_size=5)

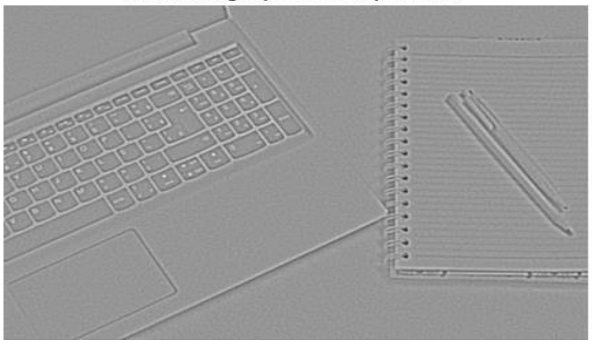
Scenario 2

Input Image = Blurred Image (kernel_size=3)

Blurred Image of Input Image (kernel_size=5)



Mask (High-pass Component)



Sharpened Image (k=2.5, kernel_size=5)