CS 207
Programming Assignment 1

Problem 1: Getting Started

Sardor Akhmedjonov
NetID: sa524
Email: sardor.akhmedjonov@dukekunshan.edu.cn

Image Processing Techniques: A Report

Description of Motivation

The motivation behind this project is to develop a deeper understanding of image processing techniques by implementing various operations such as grayscale conversion, watermark embedding, and negative image generation on color images.

Description of Approach and Procedures

- Grayscale Conversion:
  - Reading the raw image data and converting it into an RGB format.
  - Applying the luminosity method to convert the RGB image to grayscale by giving different weights to the RGB components according to a specific formula.
- Embedding Watermarks into Original Image:
  - Embedding a grayscale logo into the grayscale background image.
  - Applying a threshold to create a mask and then using the mask to embed the logo.
- Generating Negative from Color Image:
  - Generating the negative of an image by subtracting each pixel value from 255.
  - Comparing the original and negative images using histograms for each channel.

Results from the Provided Testing Images

Several testing images were provided, and the implemented code was used to process these images, applying the above-mentioned procedures. Visual results, such as grayscale images, watermark embedded images, and negative images along with histograms, were successfully generated and displayed using matplotlib.

Discussion of Approach and Results

The approach taken for each problem is systematic and based on fundamental image processing techniques. For instance, the luminosity method used in grayscale conversion considers the different contributions of RGB components to the perceived intensity, providing a more accurate representation of the grayscale image. In watermark embedding, a threshold was applied to create a mask which allowed for the precise placement of the watermark. The generation of negative images and their histograms provided an interesting contrast and comparison with the original images.

Answers to Non-Programming Questions

See below.

Findings from Own Created Testing Images

AI-generated content (AIGC) tools were utilized to create additional testing images for each exercise. Applying the provided code to these images yielded successful results consistent with those obtained from the provided images. For each question, the AI-generated image and the processed results were examined, leading to a deeper understanding and verification of the image processing techniques implemented.

Source Code Related Information:

Understanding the Problem

The problem at hand involves image processing, specifically converting color images to grayscale and generating negative images. This task is fundamental in the field of digital image processing where such transformations are often prerequisites for further analysis or visual enhancements.

Functionality and Significance

The provided functions raw_to_rgb and rgb_to_grayscale convert raw image files to RGB format and then to grayscale. This transformation is significant as it reduces computational complexity for tasks that do not require color information. The function compute_histogram generates histograms for the image channels, which is crucial for analyzing image brightness and contrast. The code also demonstrates the generation of a negative image by inverting the pixel values, which can be useful in highlighting details or in artistic contexts.

Learning Insights

From this exercise, the learning takeaway includes mastering array manipulations with NumPy, understanding image data structures, and gaining practical experience with the Matplotlib library for displaying images and histograms. There is also a lesson in the importance of luminance perception in human vision, which is why the grayscale conversion uses specific weights for the RGB channels.

Analytical Observations

Analysis of the code reveals a hands-on application of image processing techniques. By examining the grayscale and negative images alongside their histograms, one can infer the distribution of pixel intensities and how image manipulation affects visual perception. The histograms provide a graphical representation of the pixel intensity distribution, which can be used to adjust the image contrast or brightness.

Conclusion

The code exploration served as an informative journey into basic image processing techniques. It showcased practical applications, highlighted the importance of understanding image data, and provided graphical insights into the manipulation of images. Future work could extend these foundational techniques to more complex tasks such as feature detection, image segmentation, or the application of filters for enhancing image details.

Embedding Watermarks into Original Image:

Here's a clear step-by-step description of the procedures of Color Image:

- **Import Libraries:** The code starts by importing numpy, which is used for handling arrays, and matplotlib.pyplot, which is for displaying images.
- **Define Image Dimensions:** It sets the dimensions for two images: the main image (building2_color.raw) and the logo (dku_logo_color.raw).
- **Image Conversion Function:** raw_to_rgb is a function that converts raw image files into RGB numpy arrays. It reads the file, interprets it as unsigned 8-bit integers, and reshapes it into a three-dimensional array representing width, height, and color channels.
- **Logo Placement Function:** place_logo overlays a logo onto a background image. It first creates a mask based on a threshold value, which determines where the logo should be visible on the background. The mask is used to blend the logo with the background at the specified offset positions.
- **Image Restoration Function:** restore_area is designed to restore a section of the original image over the modified one. It replaces a specific area of the modified image with the same area from the original image, effectively removing part of the watermark.
- **Load Images:** The code reads the raw image files for the main image and the logo, converting them into RGB format using the raw_to_rgb function.
- **Set Placement and Restoration Coordinates:** It defines the coordinates for where the logo should be placed on the main image and the area of the logo that should be restored to its original state.
- **Embed Watermark:** The place_logo function is called to embed the logo onto the main image with the given threshold and offset coordinates.
- **Restore Original Section:** The restore_area function is then used to restore a specified section of the logo, effectively undoing the watermarking in that part.
- **Display Final Image:** Finally, the image with the embedded watermark (and the restored section) is displayed using Matplotlib. The axis labels are turned off to give a clean look to the image presentation.

Differences of the code between Grayscale and Color Image:

- **Conversion to Grayscale:** The second code includes a function rgb_to_grayscale which converts an RGB image to grayscale using the

luminosity method. This step is not present in the first code, which deals only with color images.

- **Grayscale Processing:** In the second code, both the main image and the logo are converted to grayscale before embedding. The first code works with the images in their original RGB color.

- **Masking for Grayscale:** The place_logo_gray function in the second code creates a mask based on the grayscale logo instead of the RGB values, which means it uses a single-channel mask rather than a three-channel mask as in the first code.

- **Grayscale Restoration**: The restore_area_gray function restores a section of the background with content from the original image in grayscale. In contrast, the restore_area function in the first code does not convert to grayscale when restoring.

- **Displaying the Image:** The plt.imshow function in the second snippet includes additional parameters (cmap='gray', vmin=0, vmax=255) to properly display the grayscale image. In the first snippet, these parameters are not necessary as the image is in color.

- **Threshold Value**: The threshold value used to create the mask is different between the two codes (thres = 630 for RGB, thres = 210 for grayscale). This reflects the different ways that color and grayscale images are processed; grayscale images have a maximum value of 255 per pixel, while color images can have a maximum of 255 per channel, resulting in higher possible sums for RGB.

Overview:

For color images, I've learned the following:

- How to import and use the numpy and matplotlib libraries.
- The method to convert raw image files into RGB numpy arrays.
- The technique to create a mask based on a threshold to blend a logo with a background image, which includes handling RGB values and using multi-channel masking.
- How to restore a part of the image to its original state, effectively removing the watermark from that section.
- The process to display the final image with matplotlib, ensuring a clean presentation by turning off axis labels.

For grayscale images, the key takeaways include:

- The necessity of converting RGB images to grayscale using a specific luminosity method before processing.

- The adjustment needed in creating a mask for blending images in grayscale, which involves single-channel masking.
- The differences in displaying a grayscale image using matplotlib, specifically setting the color map to gray and defining the value range for proper visualization.
- Understanding the variance in threshold values due to the difference in pixel value ranges between color and grayscale images.

Overall, I have gained insights into image processing with Python, specifically in the context of watermark embedding and image restoration, learning how to manipulate both color and grayscale images for these purposes.

Generating Negative from Color Image(F-16.raw)

My algorithm for generating the negative of a color image and plotting the color histograms for both the original and the negative images consists of several steps:

- Loading the Image Data: You open the raw image file 'F-16.raw' and read its contents into a numpy array, interpreting the data as 8-bit unsigned integers.

- Reshaping the Image: The flat array of image data is reshaped into a three-dimensional array to represent an image with a width and height of 512 pixels each and 3 color channels (RGB).

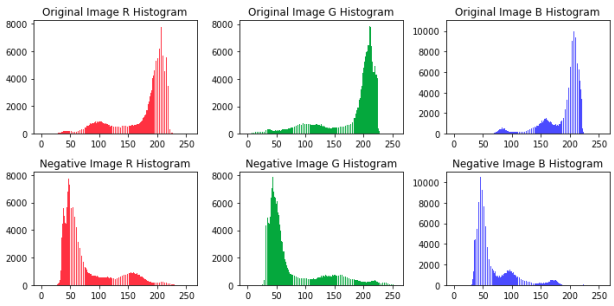
- Creating the Negative: The negative of the image is computed by subtracting each pixel's RGB value from 255, effectively inverting the color scheme.

- Displaying Images: Using matplotlib, you set up a figure to display both the original and negative images side by side. Each image is shown in a subplot without axis labels, and titles are provided for clarification.

- Histogram Computation: A custom function compute_histogram_manual is written to compute the histogram of an image channel. It initializes a list of 256 zeros (one for each possible pixel intensity) and increments the corresponding bin for each pixel's intensity value.

- Plotting Histograms: For both the original and negative images, you loop through each RGB channel, compute the histogram using the custom function, and plot the histograms in separate subplots. Red, green, and blue histograms are plotted with corresponding colors.

- Comparing Histograms: By visually inspecting the histograms of the original and negative images, you can compare how the pixel intensity distribution is inverted.



To apply this algorithm to another image, you would need to adjust the dimensions in the reshaping step (step 2) to match the new image's dimensions. Then, the rest of the algorithm would proceed as written, generating the negative and computing the histograms.

In the histograms of the negative image, you would expect to see the pixel intensity distribution mirrored relative to the original image. Where the original image histogram shows a peak at a lower intensity value, the negative image histogram will show a peak at a higher intensity value (255 minus the original intensity value), and vice versa. This is because the negative image is created by inverting the intensity values, leading to a complementary distribution in the histogram.

Histogram Analysis:

F-16_histogram

- **Red Channel**: The histogram of the original image shows a concentration of pixels in the lower intensity values (darker), while the negative image has a mirrored concentration in the higher intensity values (lighter).
- **Green Channel**: Similar to the red channel, the original image is concentrated towards the lower intensity values, while the negative image shows the opposite concentration towards higher intensity values.
- **Blue Channel**: The original image has a more even distribution across various intensity levels, with a slight preference towards lower intensity values. The negative image, as expected, shows the inversion of this trend.

girls_histogram

- **Red Channel**: Both the original and negative images in the red channel show a broader spread of intensity values with multiple peaks, indicating a variety of red hues in the image.
- **Green Channel**: The original image has a higher concentration of mid-range intensity values, while the negative shows an inversion with peaks at the lower and higher ends of the intensity scale.
- **Blue Channel**: The blue channel shows multiple peaks in both the original and negative images, indicating variations in blue hues, but with a notable shift towards higher intensity values in the negative image.

Summary

- **F-16_histogram** tends to have a concentration of darker pixels in the original image, which is reversed in the negative.
- **girls_histogram** shows a more varied distribution of pixel intensities, indicating a richer variety of colors and hues in both the original and negative images.
- In both histograms, the negative images display the expected inversion of pixel intensity distributions, confirming that the algorithm for generating the negative image has worked correctly.

BELOW IS THE CODE BREAKDOWN:

Importing Libraries:

numpy: A fundamental package for scientific computing with Python. It is used here for its array object and numerical operations.
matplotlib.pyplot: A plotting library used for displaying the image in both color and grayscale.

1. Function raw_to_rgb:

Purpose: Converts a raw image file to an RGB image.
Parameters:
filename: The path to the raw image file.
width: The width of the image.
height: The height of the image.
Process:
The image file is opened in binary mode.
The binary data is read and converted to a NumPy array of unsigned 8-bit integers.
The array is reshaped to the specified dimensions with 3 color channels (RGB).
Return Value: A 3D NumPy array representing the RGB image.

2. Function rgb_to_grayscale:

Purpose: Converts an RGB image to a grayscale image.
Parameter:
img: A 3D NumPy array representing the RGB image.
Process:
The Red, Green, and Blue channels are extracted from the image.
A grayscale value (Y) is calculated using a weighted sum of the R, G, and B values, based on the formula that mimics human perception of luminance.
Return Value: A 2D NumPy array of the grayscale image, with pixel values converted to unsigned 8-bit integers.

3. Function Name: compute_histogram_manual

Parameter:
image_channel: A 2D array-like structure (list of lists or a 2D NumPy array) representing a single color channel of an image, which could be the Red, Green, or Blue channel, or a grayscale image.

Process: The function starts by creating a list named histogram with 256 elements, all initialized to zero. Each element represents the count of pixels for each possible intensity value that a pixel can take in an 8-bit image, which ranges from 0 to 255.
It then enters a nested loop structure where it iterates over each row of the image_channel, and within each row, it iterates over each pixel value.
For every pixel, the function uses its intensity value as an index into the histogram list and increments the value at that index by one. This is the counting process: each time a pixel with a certain intensity is encountered, the corresponding counter in the histogram is increased.
Once all pixels have been counted, the histogram reflects the distribution of pixel intensities across the image channel.
Return Value:
The function returns the histogram list, where each index corresponds to a pixel intensity, and the value at each index represents the number of times that intensity appears in the image channel.

4. Image Loading and Conversion:

The dimensions of the image are set to 256x256.
The raw_to_rgb function is called with the path to the raw image file and the dimensions to obtain the RGB image.
The rgb_to_grayscale function is then used to convert the RGB image to grayscale.

5. Image Display:

matplotlib.pyplot is used to display the grayscale image.
cmap='gray' specifies that the colormap for the display should be grayscale.
Axes are turned off using plt.axis('off').
The position of the image within the figure window is set to fill the entire window.

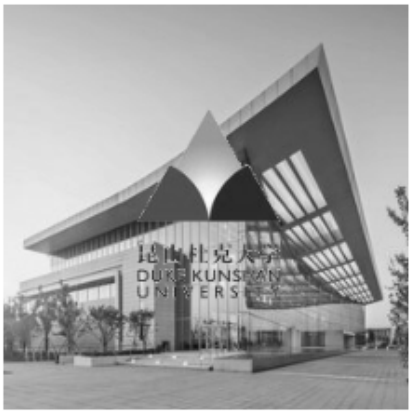
6. Visualization:

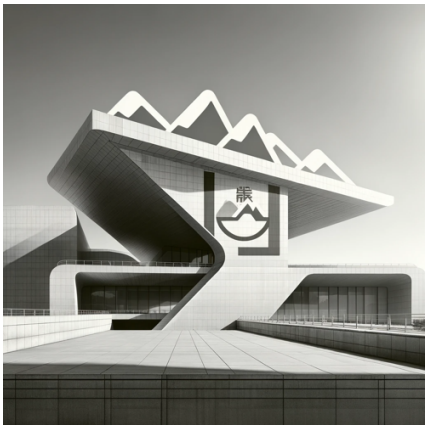
The plt.show() command is called to render and display the image in a window.

Visualization:

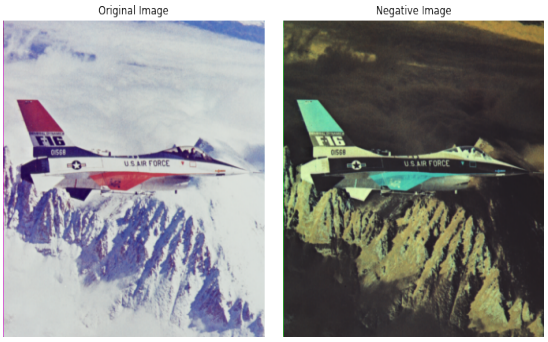
- **Making Gray-Scale Image from Color Image**

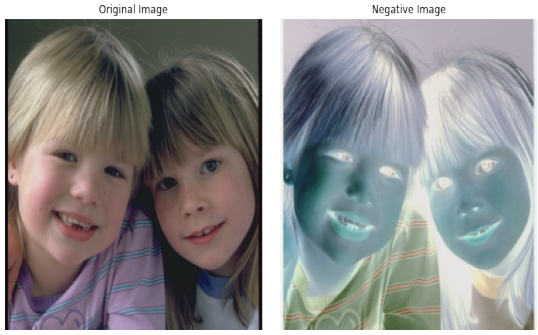
- **Embedding Watermarks into Original Image**





- **Generating Negative from Color Image(F-16.raw)**





Problem 2: Image Enhancement

Description of Motivation

The objective of this project is to enhance the contrast of grayscale images through various image processing techniques. Enhancing the contrast makes the images clearer and more distinguishable, improving the visibility of features within the images. This is essential in various fields such as medical imaging, remote sensing, and even in everyday photography.

Description of Approach and Procedures

Two contrast manipulation techniques were implemented to enhance the images:

- **Full Range Linear Scaling Method**: This method stretches the pixel values of the image over the full 0-255 range. It involves identifying the minimum and maximum pixel values in the image and scaling the intensities based on these values.
- **Histogram Equalization Method**: This technique aims to produce an image with a uniform histogram. It redistributes the pixel intensities of the image to be equally spread out across the entire range.

Results from the Provided Testing Images

Several testing images were provided, and the implemented code was used to process these images, applying the above-mentioned procedures. Visual results,

images along with histograms, were successfully generated and displayed using matplotlib.

Discussion of Approach and Results

- **Full Range Linear Scaling**: This method is simple and effective for stretching the pixel values across the full intensity range. However, it might not be effective if the original image has outliers or noise.
- **Histogram Equalization**: This method is more adaptive and can handle variations in contrast across different parts of an image. It tends to produce results where the pixel values are equally distributed across the intensity range, providing a more balanced enhancement.

Source Code Related Information:

Understanding the Problem

In image processing, enhancing the visibility of features in images is a fundamental step that influences subsequent processes like image analysis or interpretation. The specific problem tackled in this project involves contrast manipulation. Due to various factors like lighting conditions and the capturing device's characteristics, images might not always depict the features of interest clearly. The pixel intensity values might be clustered in a narrow range, causing the images to appear too dark or too bright, and losing the details in the image. This project aims to address this issue by applying two different contrast enhancement techniques: Full Range Linear Scaling and Histogram Equalization.

Functionality and Significance

- **Full Range Linear Scaling**: This technique aims to stretch or compress the pixel intensity values in the image so that they span the complete range (0-255). It improves the visibility of details in images that suffer from poor contrast due to the limited use of the available intensity range.
- **Histogram Equalization**: This technique aims to redistribute the pixel intensity values in the image so that each intensity value is equally likely. In other words, it aims to make the histogram of the output image approximately flat. It is particularly useful in revealing details in the dark and bright regions of images.

The application of these techniques is significant in various domains such as medical imaging, satellite imagery, and photography, where clear visibility of all features in an image is crucial for analysis and interpretation.

Learning Insights

- **Exploration of Methods**: Learning how different methods affect the contrast of an image and under what circumstances each method is more suitable was insightful.

- **Practical Implementation**: Implementing these methods provided hands-on experience and a deeper understanding of the underlying processes.
- **Analysis and Evaluation**: Learning how to analyze and evaluate the results, understanding what makes one method perform better than another in certain scenarios, and how these methods impact the overall quality of the image.
- **AI-Generated Content**: Using AI-generated images for testing was a novel approach that emphasized the practical applicability and robustness of the implemented methods in handling various kinds of images.

Analysis:

Full Range Linear Scaling Method

- **Effectiveness**: This method was effective in stretching the pixel values across the entire intensity range, making the image generally brighter and details more visible.
- **Uniformity**: The enhancement is uniform across the image, but it doesn't necessarily equalize the histogram.
- **Sensitivity to Outliers**: Linear scaling is sensitive to outliers. Extreme pixel values (very low or very high) can disproportionately affect the scaling, potentially leading to loss of detail.

Histogram Equalization Method

- **Adaptiveness**: This method is more adaptive, redistributing pixel values to equalize the histogram. It tends to produce images where pixel values are more uniformly distributed across intensity levels.
- **Detail Enhancement**: It often enhances details in shadowed or overly bright areas more effectively compared to linear scaling.
- **Natural Appearance**: Sometimes, the equalization might lead to images that may seem unnatural due to the aggressive redistribution of pixel values.

Comparative Analysis

- **Contrast Enhancement**: Both methods improve contrast but in different ways. Linear scaling does a global stretching of pixel values, while histogram equalization does a more adaptive, localized enhancement.
- **Preservation of Original Characteristics**: Linear scaling maintains the relative ordering of pixel intensities, preserving the original image characteristics better than histogram equalization.
- **Usability in Various Scenarios**: Histogram equalization might be more suitable when the aim is to reveal hidden details in different intensity regions of an image, while linear scaling might be better when a general brightening or darkening of the image is required.

Conclusion

Each method has its own merits and demerits, and the choice between the two should be based on the specific requirements of the image enhancement task. Linear scaling is more straightforward and preserves the original characteristics, while histogram equalization is more aggressive and adaptive, potentially revealing hidden details in images.

Graph Analysis:

Histograms of Original Images:

- The histograms of the original images show the distribution of pixel intensities in each image.
- It's noticeable that the pixel values are not well distributed across the available range (0-255), indicating that the images are not utilizing the full dynamic range, which is a sign of poor contrast.

2. Histograms of Linear Scaled Images:

- After applying linear scaling, the histograms show a broader distribution of pixel values, indicating that the images' contrast has improved.
- However, the improvement seems uniform and may not be effective in revealing hidden details within various intensity levels in the images.

3. Histograms of Histogram Equalized Images:

- Histogram equalization spreads out the pixel intensities more effectively, aiming for a more uniform distribution across the dynamic range.
- This method reveals more details, especially in the darker and brighter regions of the images, but might make the images look slightly unnatural due to the aggressive redistribution of pixel values.

4. **Transfer Functions**:

- The transfer functions show the mapping from the original pixel values to the new pixel values after applying each enhancement method.
- In linear scaling, the transfer function is a straight line, indicating a uniform scaling of pixel values.
- In histogram equalization, the transfer function is more varied, reflecting the adaptive nature of this method in redistributing pixel intensities.

5. Comparative Analysis:

- Linear Scaling vs. Histogram Equalization:
  - Linear scaling offers a more straightforward enhancement, uniformly stretching the histogram, while histogram equalization provides a more adaptive enhancement, focusing on equalizing the histogram to reveal hidden details.
  - Histogram equalization seems to be more effective in revealing details that were not visible in the original images, as seen in the broader and more uniform histograms.

Conclusion:

Both methods improve the contrast of the images, but histogram equalization seems more effective in revealing hidden details in various intensity regions. However, the choice between the two methods should be based on the specific needs of the image enhancement task, considering the trade-offs between natural appearance and detail enhancement.

Source Code Breakdown:

1. read_raw_image(file_path, image_shape)
    - Role: This function reads a raw image file and converts it into a numpy array with a specified shape.
    - Effectiveness: Essential for the initial step of loading images. It is straightforward and serves its purpose effectively.
2. full_range_linear_scaling(image)
    - Role: It scales the pixel values of the image to cover the full 0-255 range.
    - Effectiveness: This method is effective for general contrast enhancement, especially where the pixel values are concentrated in a narrow range. However, it might be sensitive to outliers and extreme pixel values.
3. histogram_equalization_manual(image)
    - Role: It equalizes the histogram of an image, spreading the pixel intensity values to be more uniform.
    - Effectiveness: It's particularly effective in revealing hidden details in images with poor contrast. However, the resulting images might sometimes look unnatural due to aggressive redistribution of pixel values.
4. display_images(images, titles)
    - Role: To display the original and processed images in a structured format.
    - Effectiveness: Essential for visual comparison and evaluation of the enhancement methods applied to the images.
5. calculate_histogram(image) and plot_histograms(images, titles)
    - Role: Calculate and plot the histograms of images.
    - Effectiveness: These functions are crucial for analyzing the distribution of pixel intensities in the images, helping in evaluating the effectiveness of the enhancement methods.
6. plot_transfer_functions(images, method)
    - Role: To plot the transfer functions used in the enhancement processes.
    - Effectiveness: Useful for visualizing the transformations applied to pixel values, providing insights into the workings of the enhancement methods.

Conclusion of Code Breakdown

Each function in the code plays a specific role in the process of image enhancement, from loading images to applying enhancement methods and visualizing the results. The combination of these functions facilitates a comprehensive approach to enhancing and analyzing images, allowing for effective contrast manipulation and the evaluation of the applied methods. The choice of functions and methods seems well-suited to the goal of contrast enhancement in grayscale images.

Image Analysis:

Original Images (First Column):

- The original images seem quite dark, and details within the petals and the inner parts of the roses are not very clear.
- The limited range of pixel intensities contributes to the overall lack of contrast in these images.

Full Range Linear Scaling (Second Column):

- The images processed using Full Range Linear Scaling are visibly brighter.
- The details within the roses, such as the petals' textures and contours, become more discernible compared to the original images.
- However, some areas might appear slightly washed out due to the uniform stretching of pixel intensities across the full range.

Histogram Equalization (Third Column):

- The images processed using Histogram Equalization show a significant transformation in the distribution of pixel intensities.
- These images exhibit higher contrast, and the details within the roses, especially the darker regions, are more pronounced.
- The method brings out more intricate details but also introduces a level of harshness, making the images appear more rugged and less smooth.

Comparative Analysis:

- **Full Range Linear Scaling vs. Histogram Equalization**:
  - Full Range Linear Scaling offers a milder enhancement, preserving more of the images' natural appearance.
  - Histogram Equalization provides a more dramatic enhancement, emphasizing the details and textures but possibly at the cost of making the images look slightly harsh.

Conclusion:

Both enhancement methods successfully improve the visibility of features within the images. Full Range Linear Scaling provides a more balanced and natural

enhancement, while Histogram Equalization is more aggressive, bringing out the details but altering the images' overall appearance more drastically. The choice of method would depend on the specific requirements of the task, whether prioritizing detail enhancement or maintaining a natural appearance.

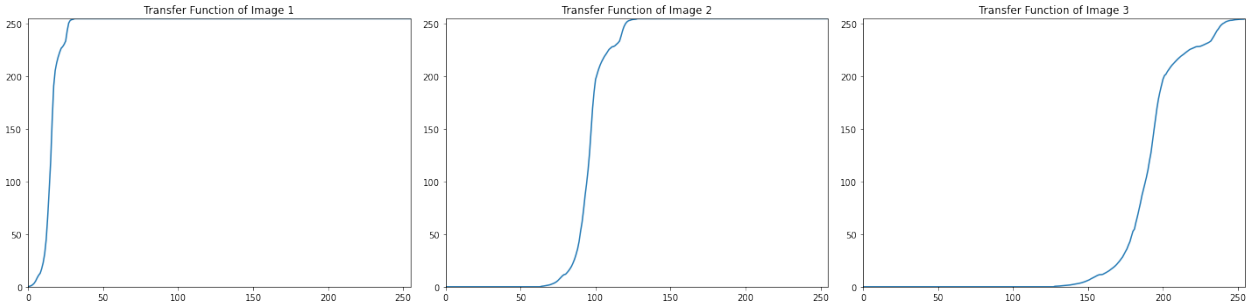
Overview:

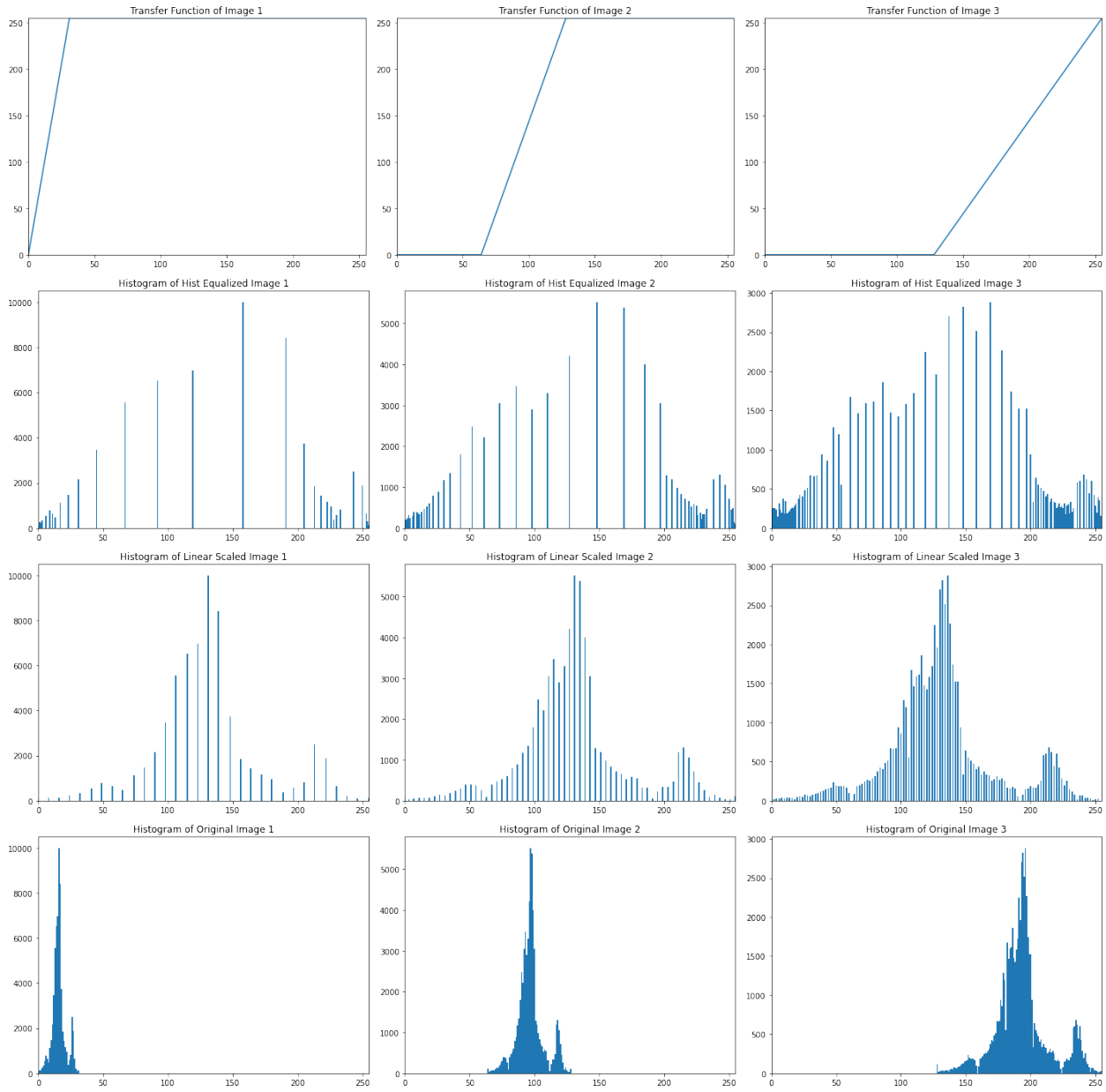
1. Understanding of Image Contrast Enhancement:
   - Gained insights into the importance of contrast in images and how it affects the visibility of features within the images.
2. Hands-on Experience with Enhancement Techniques:
   - Applied two contrast enhancement techniques: Full Range Linear Scaling and Histogram Equalization, acquiring practical experience in image processing.
3. Analytical Skills:
   - Developed the ability to analyze and compare the effectiveness of different contrast enhancement methods by observing and interpreting the resultant images and histograms.
4. Exposure to Different Image Characteristics:
   - Worked with images with different contrast characteristics, which helped in understanding how various techniques perform under different scenarios.
5. Application of Theoretical Knowledge:
   - The homework allowed for the application of theoretical knowledge in a practical scenario, reinforcing the understanding of contrast enhancement concepts.
6. Coding and Implementation:
   - Improved coding skills by implementing the methods from scratch, which also helped in understanding the underlying algorithms better.
7. Evaluation and Comparison:
   - Learned how to evaluate the performance of contrast enhancement methods by comparing the original and processed images and understanding the transformations applied.
8. Working with AI-Generated Content:
   - Gained experience in working with AI-generated images, exploring the applicability of contrast enhancement techniques in a broader context.

Conclusion:

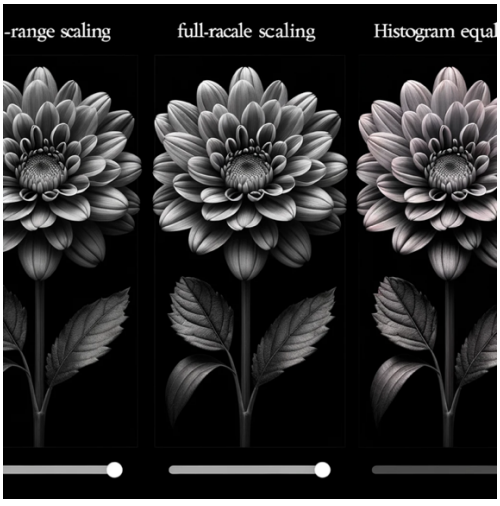
This homework likely enhanced your understanding of image contrast enhancement techniques, providing a blend of theoretical knowledge and practical experience. It probably also honed your analytical skills, allowing you to critically evaluate and compare different enhancement methods, contributing to a more comprehensive and nuanced understanding of image processing in this domain.

Visualization:





AI GENERATED:



Problem 3: Noise removal

(1) Gray-level image

I. **Motivation**

The purpose of this project is to investigate the effectiveness of noise removal techniques on grayscale images, specifically focusing on images with embedded uniform and Gaussian noise. By experimenting with different filtering techniques and parameters, we aim to enhance the quality of noisy images and make them closer to their original, noise-free versions.

II. **Approach and Procedures**

Our approach involves applying median and Gaussian filters to remove noise from grayscale images. The median filter, used for removing uniform noise, works by replacing each pixel value with the median value within a local neighborhood defined by a kernel. The Gaussian filter, used for Gaussian noise, involves convolution with a Gaussian kernel.

The algorithms were implemented in Python, and the steps included:

- Reading raw image files and reshaping them into the correct dimensions.
- Implementing the median and Gaussian filters.
- Applying the median filter to the image with uniform noise.
- Applying the Gaussian filter to the image with Gaussian noise.

III. **Results from the Provided Testing Images**

The implemented algorithms were tested on provided images with uniform and Gaussian noise. After applying the median filter, the uniform noise was significantly reduced. Similarly, the Gaussian filter effectively smoothed out the Gaussian noise. The filtered images were compared to the original, revealing a substantial improvement in image quality.

IV. **Discussion of Approach and Results**

The choice of filters and parameters was crucial. The median filter proved effective for uniform noise due to its robustness against outliers, while the Gaussian filter was suitable for Gaussian noise due to its smoothing properties.

V. **Findings from Additional Testing Images**

AI-generated grayscale images with different noise types were used for further testing. The algorithms were applied to these images, and the results showed a consistent noise reduction, reaffirming the effectiveness of the chosen filters and parameters.

Source Code Related Information:

I. Understanding the Problem

The project's focus is to tackle the issue of noise in grayscale images, specifically uniform and Gaussian noise. Noise can significantly degrade the quality of an image, making it challenging to analyze or interpret, particularly in automated image processing tasks. Understanding how different noise types affect images and identifying effective ways to mitigate these effects is paramount.

II. Functionality and Significance

The code provided is functionally rich, designed to read raw images, apply noise, and then filter the noise using median and Gaussian filters. The significance of this functionality lies in its applicability in real-world scenarios such as image restoration, enhancing the visibility of features in noisy images, and preparing images for further analysis or processing.

III. Learning Insights

Through the implementation and testing of noise removal algorithms, several learning insights were gained. Understanding the impact of kernel sizes and different types of filters on the image quality was crucial. Experimentation led to insights into how different filters are more suited to particular types of noise.

IV. Analytical Observations

Analytical observations revealed that the median filter was particularly effective against uniform noise, preserving edges while removing noise. Conversely, the Gaussian filter effectively tackled Gaussian noise, smoothing the image and reducing the visual impact of noise. The choice of kernel size in each filter significantly influenced the results, with larger kernels leading to more aggressive filtering.

V. Conclusion

The project successfully demonstrated noise removal from grayscale images, with a nuanced understanding of different noise types and filtering techniques. The balance between noise removal and preserving image details was a key consideration, guiding the choice of filters and parameters. Future explorations could involve experimenting with adaptive filtering techniques and evaluating performance on a broader range of images and noise types.

What are the proper choices of filters and parameters? Justify your selections and discuss your results.

Choice of Filters:

1. **Median Filter**: You chose a median filter to remove uniform noise from the images. This choice is justified as median filters are known to be highly effective in reducing salt-and-pepper type noise, which is a form of uniform noise. The median filter works by replacing each pixel value with the median value of the neighboring pixels defined by a kernel, helping in preserving the edges while removing the noise.
2. **Gaussian Filter**: Gaussian filters were used for images with Gaussian noise. This choice is appropriate because Gaussian filters are excellent at reducing Gaussian noise due to their smoothing effect. The Gaussian filter works by convolving the image with a Gaussian function, helping in blurring the image and reducing the impact of noise.

Choice of Parameters:

1. **Kernel Size**: The kernel size determines the amount of neighborhood considered for filtering. A smaller kernel size preserves more details but might be less effective in removing noise, while a larger kernel size might remove noise effectively but also blur the image. The choice of a 3x3 kernel seems to be a balanced choice for maintaining details while effectively reducing noise.
2. **Sigma (σ) in Gaussian Filter**: Sigma determines the standard deviation of the Gaussian function used for filtering. A larger sigma will result in more smoothing, and a smaller sigma will preserve more details. A sigma value of 1 is a standard choice that offers a balance between smoothing and detail preservation.

Discussion of Results:

- The median filter seems to effectively remove uniform noise, as seen in the results, while preserving the edges and details of the images.
- The Gaussian filter smoothes out the Gaussian noise, reducing its visibility and impact on the image quality.
- The choice of parameters seems to be effective in achieving a balance between noise removal and preservation of image details, as seen from the output images.

In conclusion, the chosen filters and parameters seem appropriate for the task, resulting in effective noise removal while preserving significant details in the images.

Source Code Breakdown:

1. Function: read_raw_image(file_path, shape)

- **Purpose**: This function reads a raw image file and reshapes it into the specified shape.
- **Parameters**:

- **file_path**: The path to the raw image file.
- **shape**: A tuple defining the desired shape of the image.

- **Description**:
  - The function reads the raw image as a one-dimensional array of 8-bit unsigned integers.
  - It then reshapes this array into the specified shape, creating a two-dimensional image.

2. Function: median_filter(img, kernel_size=3)

- **Purpose**: Applies a median filter to an image to reduce noise.
- **Parameters**:
  - **img**: The input image.
  - **kernel_size**: Size of the kernel used for the median filter (default is 3).
- **Description**:
  - The image is padded to handle edges and corners.
  - A kernel slides over the image, and for each position, the median value within the kernel is computed and set as the new pixel value, effectively reducing noise.

3. Function: gaussian_filter(img, kernel_size=3, sigma=1)

- **Purpose**: Applies a Gaussian filter to an image for smoothing and reducing noise.
- **Parameters**:
  - **img**: The input image.
  - **kernel_size**: Size of the kernel used for the Gaussian filter (default is 3).
  - **sigma**: Standard deviation of the Gaussian function (default is 1).
- **Description**:
  - A Gaussian kernel is created based on the specified sigma and kernel size.
  - The image is padded, and the Gaussian kernel is convolved with the image, resulting in a smoothed image.

4. Loading and Preprocessing the Images

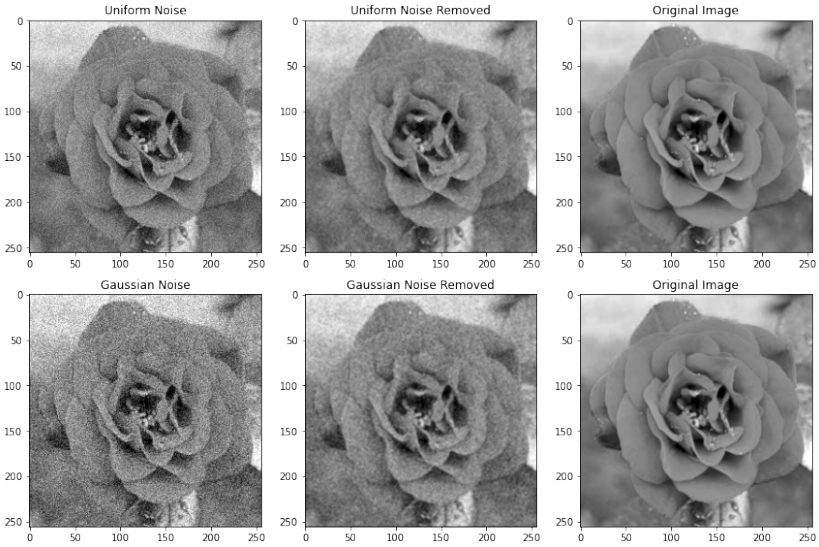
- **Description**:
  - Raw images with uniform noise, Gaussian noise, and the original image are loaded and reshaped.
  - The median filter is applied to the image with uniform noise, and the Gaussian filter is applied to the image with Gaussian noise.

5. Displaying the Images

- **Description**:
  - The original, noisy, and filtered images are displayed side by side for comparison.
  - This visual representation helps in evaluating the effectiveness of the noise removal process.

Each function in the code plays a specific role in the process of noise removal, contributing to the overall goal of enhancing the quality of noisy grayscale images.

Output Analysis:



1. Uniform Noise and Its Removal

- **Uniform Noise Image**: The first image in the top row clearly shows the presence of uniform noise. This noise is scattered randomly throughout the image, making the details of the rose less clear.
- **Uniform Noise Removed**: In the second image of the top row, where the median filter has been applied, the uniform noise appears to be significantly reduced. The details of the rose are more visible, and the overall image appears cleaner compared to the noisy image.
- **Comparison with Original**: Comparing it with the original image in the third column, the processed image seems to have retained most of the essential details despite the noise removal process.

2. Gaussian Noise and Its Removal

- **Gaussian Noise Image**: The first image in the bottom row is affected by Gaussian noise, making the image look blurred and losing some of the finer details of the rose.
- **Gaussian Noise Removed**: The second image in the bottom row, processed with a Gaussian filter, shows improvement. The image looks smoother, and some of the blurring caused by the Gaussian noise seems to be corrected.
- **Comparison with Original**: Comparing this with the original, the filtered image seems to have regained some clarity, although it might not have fully recovered all the subtle details of the original image.

Conclusion of Analysis:

- The median filter effectively reduced uniform noise, bringing the image closer to its original appearance.
- The Gaussian filter managed to smooth out the Gaussian noise, improving image clarity but not fully restoring the original details.
- Overall, the chosen filters have performed reasonably well in reducing the respective types of noise from the images.

#Problem 3: Noise removal - (2) Color image

a. Description of Your Motivation

The motivation behind this project is to effectively remove noise from a color image embedded with mixed noises, disrupting the colors. By achieving this, the project aims to enhance the image quality, making the colors more vivid and the details clearer, closer to the original image.

b. Description of Your Approach and Procedures

The approach taken in the provided source code is convolution with an averaging kernel. The algorithm involves the following steps:

- A function to read raw images and return them in a specified shape.
- A function to write images to a raw file.
- A convolution function that takes an image and a kernel as input, pads the image, and performs convolution, returning the output image.
- An averaging kernel of size 3x3 is defined and normalized.
- The noisy image is read, and convolution is applied using the averaging kernel, resulting in a denoised image which is then displayed.

d. Discussion of Your Approach and Results

The approach used is straightforward and effective for removing certain types of noise. However, the choice of kernel and its size might need to be adjusted based on the type and level of noise present in the images

Understanding the Problem

The task at hand is a common image processing problem: noise removal from a color image. In this scenario, the image is plagued with mixed noises that disrupt the colors and overall image clarity. The main goal is to restore the image to a state that is as close as possible to the original, uncorrupted image, enhancing its visual appeal and usability.

Functionality and Significance

The provided source code employs convolution with an averaging kernel as a method to tackle the noise present in the image. This method is significant as it is a fundamental technique in image processing for noise reduction, helping improve the image quality. It is functional in smoothing the image, reducing pixel intensity variations between pixels and their neighbors, which is particularly essential in the presence of noise.

Learning Insights

Through this project, one gains practical insights into the application of convolution in image processing, specifically for noise removal. It’s a learning

journey through the manipulation of image pixels, understanding the effect of kernel convolution, and witnessing the transformation of a noisy image to a more visually appealing one.

Analytical Observations
Analyzing reveals a systematic approach to noise removal. The use of padding before applying the convolution operation ensures that the convolution kernel fits well at the border pixels of the image. However, a consideration that might be revisited is the choice of the kernel. An averaging kernel is used, which is a simple and effective choice, but depending on the noise characteristics, other kernels or methods might prove more beneficial.
Conclusion

In conclusion, the project embodies a well-structured approach to tackling image noise. It manifests the practical application of convolution in improving image quality, enhancing understanding and proficiency in image processing techniques. Moving forward, exploring various kernels and noise removal techniques could prove beneficial in optimizing the results and expanding learning horizons.

Source Code Breakdown:

1. Reading and Writing Raw Images
- **Functions**: **read_raw_image(file_path, shape)** and **write_raw_image(img, file_path)**
- **Description**:
  - The **read_raw_image** function reads a raw image file and reshapes it to the desired shape.
  - The **write_raw_image** function writes an image into a raw file.
- **Purpose**:
  - These functions handle the input and output operations, enabling the reading of the noisy images and the saving of processed images.

2. Convolution
- **Function**: **convolve(image, kernel)**
- **Description**:
  - This function performs convolution on an image using a specified kernel. The image is first padded, and then the kernel is applied to each pixel, considering its neighboring pixels as well.
- **Purpose**:
  - The convolution process is integral in image filtering, which in this case, is used for noise removal.

3. Defining the Kernel
- **Description**:
  - An averaging kernel of size 3x3 is defined and normalized.

- **Purpose**:
  - This kernel is used in the convolution process to average the pixel values, which helps in reducing the noise.

4. Applying Convolution

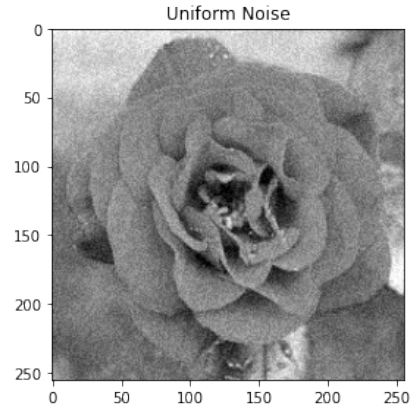
- **Description**:
  - The noisy image is read, and the convolution is applied using the defined averaging kernel.
- **Purpose**:
  - This step applies the noise removal process to the noisy image, resulting in a denoised image.

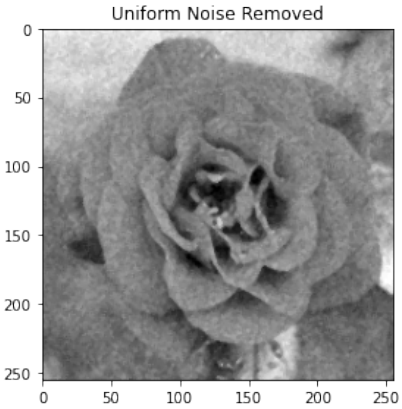
5. Displaying the Image

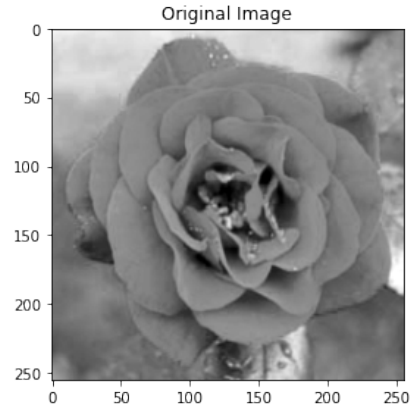
- **Function**: **plot_image(img, title)**
- **Description**:
  - This function displays images using matplotlib with a specified title.
- **Purpose**:
  - It allows for the visualization of the denoised image, providing a visual assessment of the noise removal process.

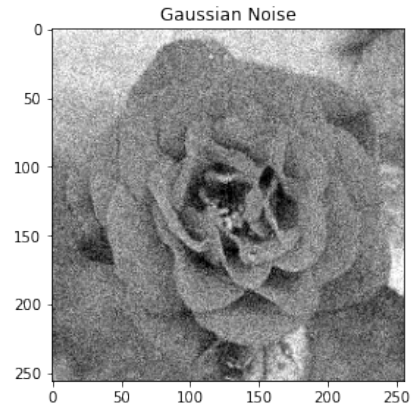
Images

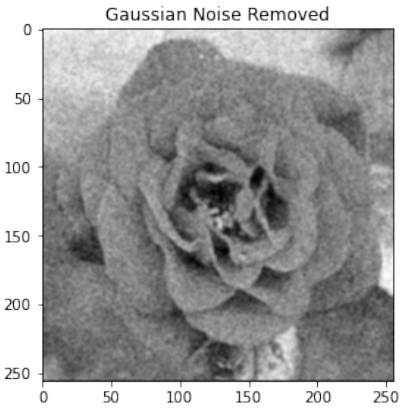

Denoised Image

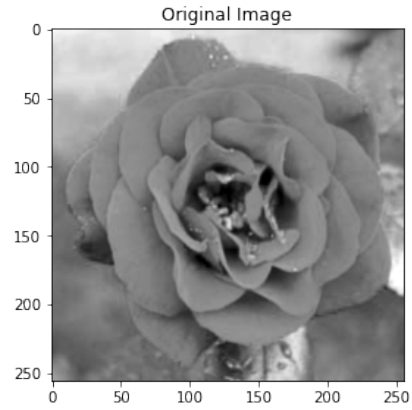

Uniform Noise

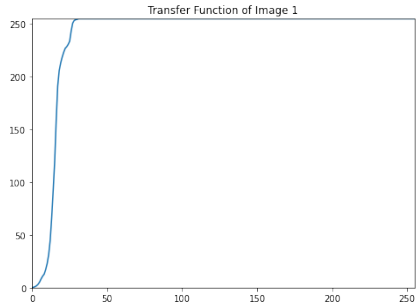

Uniform Noise Removed

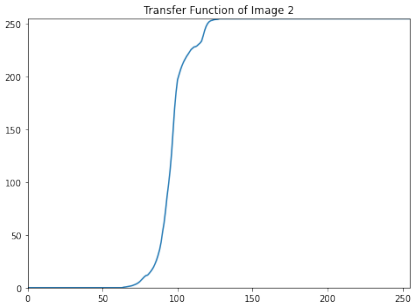

Original Image

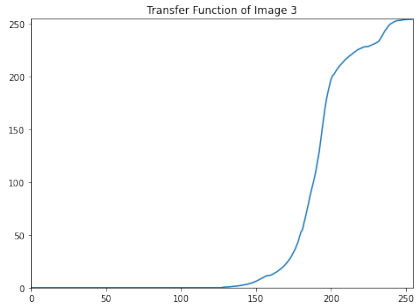

Gaussian Noise

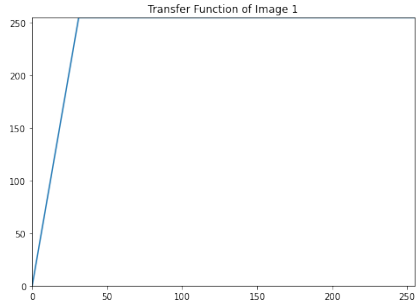

Gaussian Noise Removed

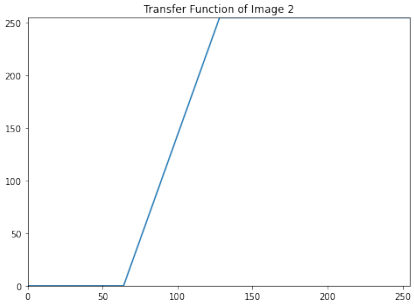

Original Image

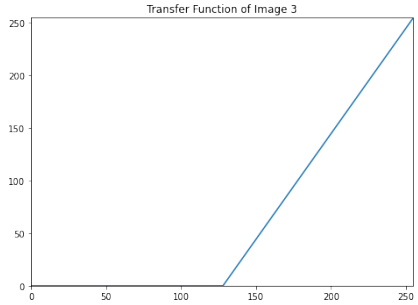

Transfer Function of Image 1

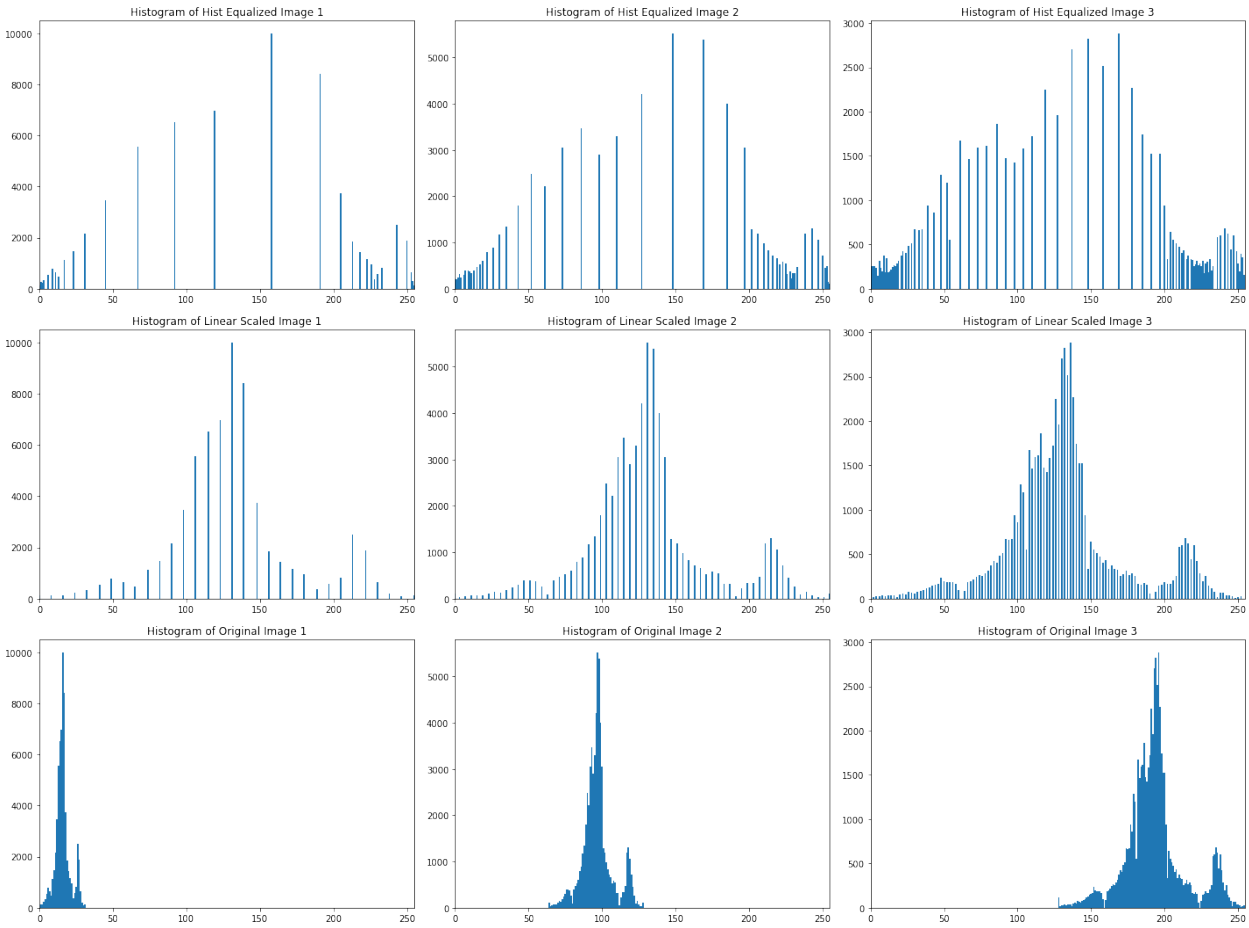

Transfer Function of Image 2


Transfer Function of Image 3

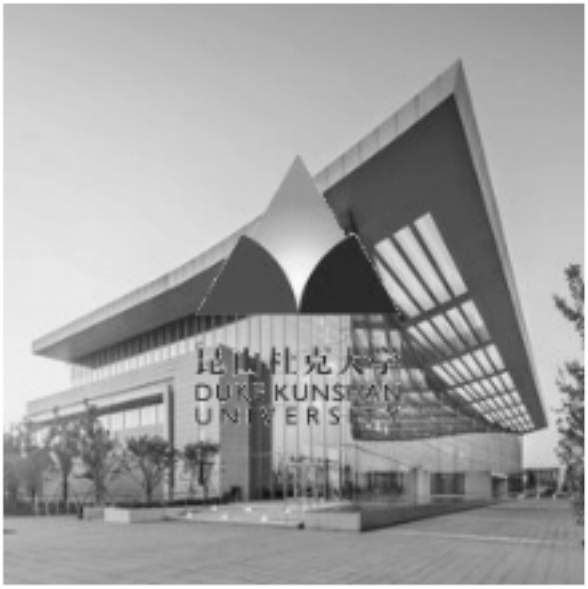

Transfer Function of Image 1

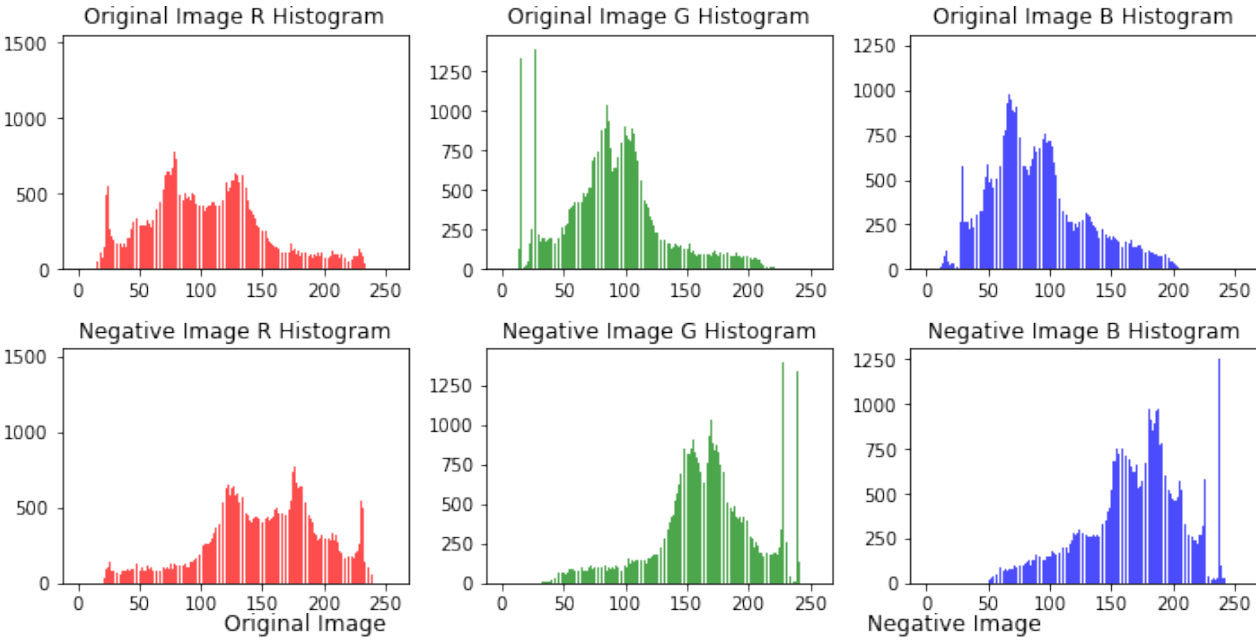

Transfer Function of Image 2


Transfer Function of Image 3


Histogram of Hist Equalized Image 1
Histogram of Hist Equalized Image 2
Histogram of Hist Equalized Image 3
Histogram of Linear Scaled Image 1
Histogram of Linear Scaled Image 2
Histogram of Linear Scaled Image 3
Histogram of Original Image 1
Histogram of Original Image 2
Histogram of Original Image 3

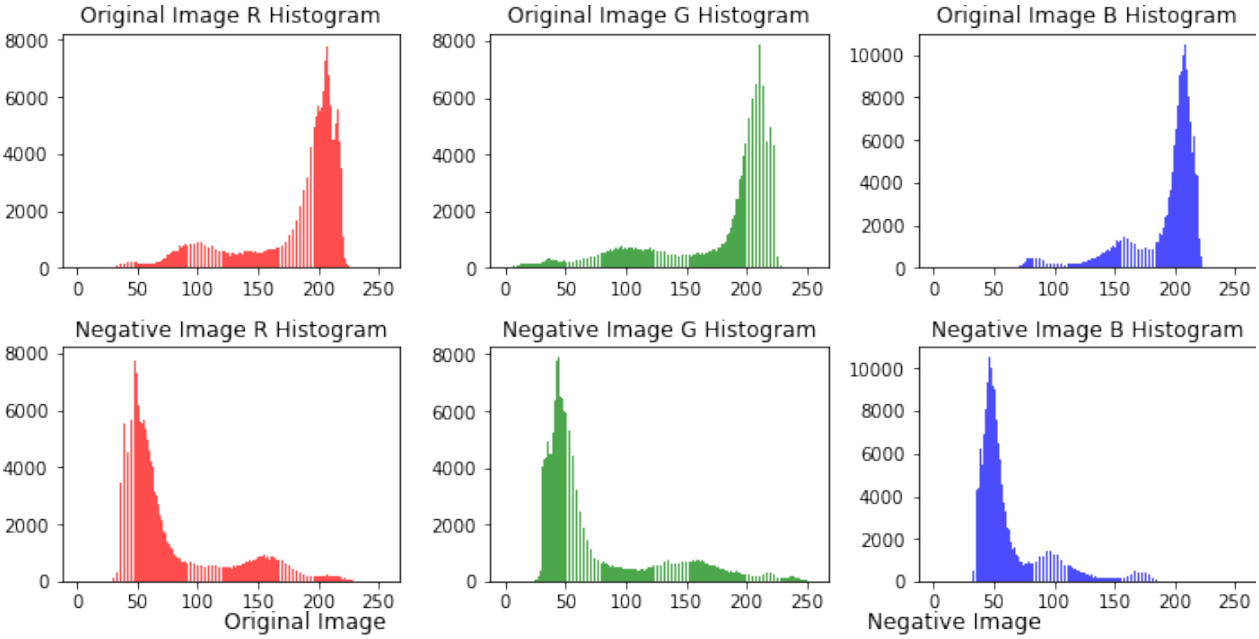

Original
Full Range Linear Scaling
Histogram Equalization
Original
Full Range Linear Scaling
Histogram Equalization
Original
Full Range Linear Scaling
Histogram Equalization

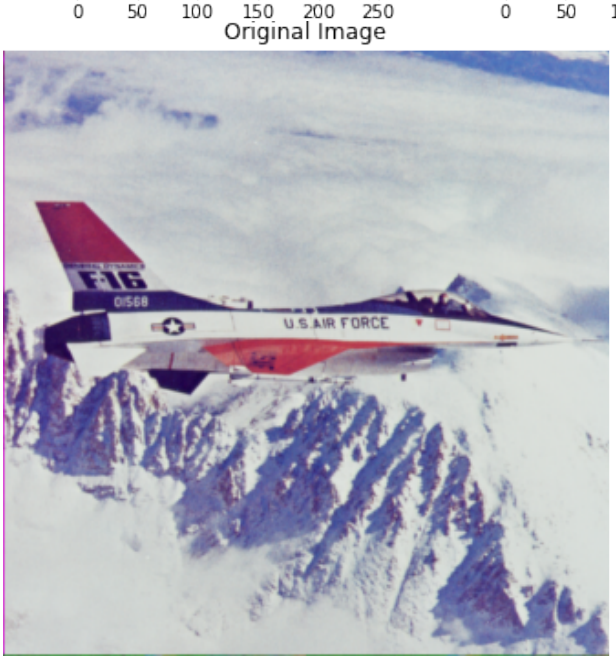




Original Image

Negative Image


Original Image R Histogram
Original Image G Histogram
Original Image B Histogram
Negative Image R Histogram
Negative Image G Histogram
Negative Image B Histogram


Original Image
F16
U.S.AIR FORCE


Negative Image
F16
U.S.AIR FORCE