This Homework consists of 3 main functions. In the first place we need to apply a Mean Filter to the two testing images that we have. Then, we will apply a Median Filter. Finally, we are going to create histograms for the different generated images as well as the original images.

Let's start with the Mean Filter. This function applies a mean filter to an image. Here's a step-by-step explanation:

```python
def mean_filter(image, kernel_size=3, padding=1):
    """
    Apply a mean filter to an image manually without using convolve2d.

    :param image: Input image as a numpy array.
    :param kernel_size: Size of the mean filter kernel.
    :param padding: Padding applied to the image.
    :return: Filtered image as a numpy array.
    """
    # Convert image to grayscale if it's not already
    if len(image.shape) == 3:
        image = np.mean(image, axis=2)

    # Pad the image
    if padding > 0:
        image = np.pad(image, pad_width=padding, mode='constant', constant_values=0)

    # Initialize the output image
    height, width = image.shape
    output = np.zeros((height, width))

    # Calculate the offset for the kernel
    offset = kernel_size // 2

    # Iterate over each pixel in the image
    for y in range(offset, height - offset):
        for x in range(offset, width - offset):
            # Calculate the average value within the kernel's neighborhood
            sum_value = 0.0
            for ky in range(-offset, offset + 1):
                for kx in range(-offset, offset + 1):
                    sum_value += image[y + ky, x + kx]
            average = sum_value / (kernel_size ** 2)
            output[y, x] = average

    # Remove the padding
    output = output[offset:height - offset, offset:width - offset]

    return output
```

Check if Color Image:

If the input image has three dimensions (indicating a color image), it is converted to grayscale by averaging the color channels.

Padding:

The image is padded with zeros on all sides. The amount of padding is specified by the padding parameter. Padding helps handle the borders of the image when applying the kernel.

Output Image Initialization:

A new, empty array (output) is created with the same dimensions as the padded image. This will store the filtered image.

Kernel Offset Calculation:

The offset is half the kernel size, used to center the kernel over each pixel.

Iterating Over Each Pixel:

The function iterates over each pixel in the image (excluding the padding).

For each pixel, it considers a neighborhood defined by the kernel size.

Applying the Mean Filter:

Within the kernel's neighborhood, it sums up all the pixel values and then divides this sum by the total number of pixels in the kernel (i.e., kernel_size ** 2) to find the average.

This average value is assigned to the corresponding pixel in the output array.

Removing Padding:

Finally, the padding is removed from the output image, and the filtered image is returned.

Now, we will explain the Median Filter. This function manually applies a median filter to an image:

```python
def manual_median_filter(image, kernel_size=3, padding=1):
    """
    Apply a median filter to an image manually.

    :param image: Input image as a numpy array.
    :param kernel_size: Size of the median filter kernel.
    :param padding: Padding applied to the image.
    :return: Filtered image as a numpy array.
    """
    # Convert image to grayscale if it's not already
    if len(image.shape) == 3:
        image = np.mean(image, axis=2)

    # Pad the image
    if padding > 0:
        image = np.pad(image, pad_width=padding, mode='constant', constant_values=0)

    # Initialize the output image
    height, width = image.shape
    output = np.zeros((height, width))

    # Calculate the offset for the kernel
    offset = kernel_size // 2

    # Iterate over each pixel in the image
    for y in range(offset, height - offset):
        for x in range(offset, width - offset):
            # Collect the values within the kernel's neighborhood
            neighborhood = []
            for ky in range(-offset, offset + 1):
                for kx in range(-offset, offset + 1):
                    neighborhood.append(image[y + ky, x + kx])

            # Find the median value
            median_value = np.median(neighborhood)
            output[y, x] = median_value

    # Remove the padding
    output = output[offset:height - offset, offset:width - offset]

    return output
```

Conversion, Padding, Output Initialization:

Like the mean filter, it converts the image to grayscale, if necessary, applies padding, and initializes an output array.

Iterating Over Pixels:

The function iterates over each pixel in the padded image.

Collecting Neighborhood Values:

For each pixel, it collects the values of all the pixels in its neighborhood (defined by the kernel size).

Finding the Median:

It then calculates the median of these values. The median is the middle value when all the values are sorted. This is less sensitive to outliers than the mean, which is why median filtering is effective at removing 'salt and pepper' noise.

Assigning the Median Value:

The median value is assigned to the corresponding pixel in the output image.

Removing Padding and Returning:

The padding is removed, and the median-filtered image is returned.

Finally, here is the explanation of the create and save the histograms. This function generates a histogram of an image and saves it:

```
# Function to create and save histograms using Matplotlib
def create_and_save_histogram(image, filename, title):
    if len(image.shape) == 3:
        image = image.mean(axis=2).astype(np.uint8)

    plt.figure()
    plt.hist(image.flatten(), bins=256, range=[0, 256], color='black')
    plt.title(title)
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.savefig(filename)
    plt.close()
```

Grayscale Conversion:

If the image is not in grayscale, it is converted to grayscale. This simplifies the histogram to focus on intensity values only.

Histogram Plotting:

A histogram is created using plt.hist(), which bins the pixel values (ranging from 0 to 255) and shows their frequency.

The histogram is plotted with the specified title, and labels are added for the x-axis ('Pixel Value') and y-axis ('Frequency').

Saving the Histogram:

The histogram is saved as an image file with the given filename.

The plot is then closed to free up resources.

In this section, you can find the original and the results images.

Original images:

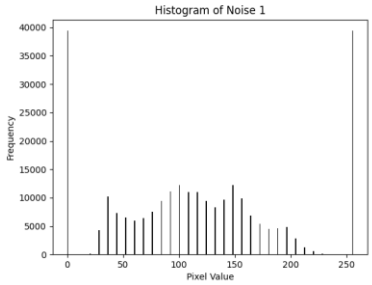
Filtered images using Mean Filter:

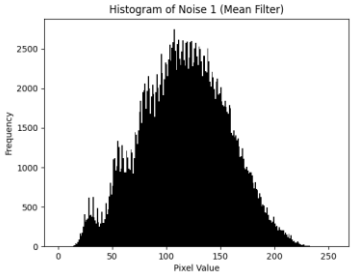
Filtered images using Median Filter:

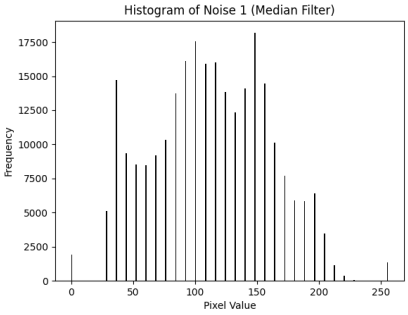
Please note that we could get rid of all the salt and paper noise by choosing a bigger kernel's size, but the resulting image would be blurred and with missing features. Thus, I choose to keep the kernel size to 3.

Histograms:

First image:







Observations from the Histograms

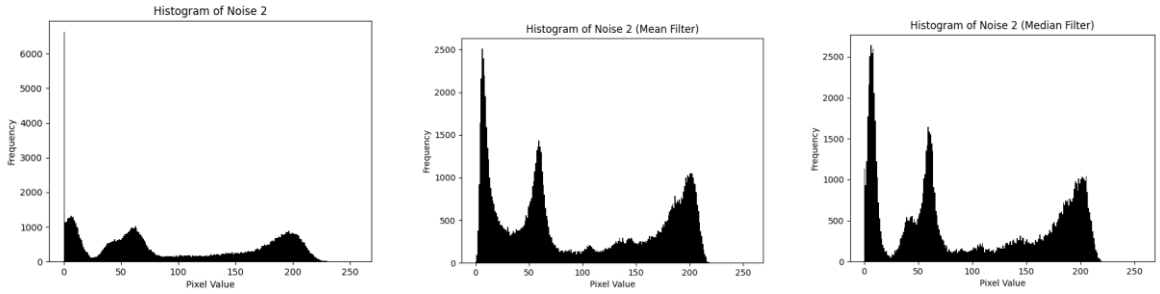
1. **Original Image (noise1.png)**:
    - The histogram shows a bimodal distribution, characteristic of 'salt and pepper' noise. There are high frequencies at the extremes (near 0 and 255), representing the black and white dots of the noise.
    - The rest of the histogram is relatively flat, indicating a spread of other pixel values, likely from the actual image content.
2. **After Mean Filter (noise1_q1.png)**:
    - The histogram displays a smoothing of the extremes. The peaks at 0 and 255 are less pronounced, indicating that the mean filter has reduced some of the noise.
    - The distribution becomes more centralized, showing that the mean filter has averaged out some of the noise, blending it with the neighboring pixels.
3. **After Median Filter (noise1_q2.png)**:
    - The histogram here also shows a reduction in the extremes, similar to the mean filter, but typically with a better preservation of edges and details.
    - The median filter tends to be more effective in reducing 'salt and pepper' noise without blurring the image as much as the mean filter.

Key Points

- **Noise Reduction**: Both filters reduce the 'salt and pepper' noise, as seen by the decrease in the high frequencies at the pixel values 0 and 255.
- **Effect on Image Content**: The mean filter can blur the image more than the median filter. This is because the mean filter averages pixel values, which can blend edges and details, whereas the median filter preserves edges better by selecting the median value.

- **Histogram Changes**: The changes in the histograms reflect the nature of each filter. The mean filter tends to create a more 'bell-shaped' curve due to averaging, while the median filter maintains more of the original image structure.

Second image:



Observations from the Histograms

1. **Original Image (noise2.png)**:
    - The histogram shows a wide distribution of pixel values, indicating a variety of intensities in the original image.
    - There are no pronounced peaks at the extremes, suggesting that the noise in this image might be different from the typical 'salt and pepper' noise.
2. **After Mean Filter (noise2_q1.png)**:
    - The histogram shows a smoother distribution. The mean filter has averaged the pixel values, reducing the variations in intensity.
    - This smoothing effect can help reduce noise but may also lead to some loss of detail in the image.
3. **After Median Filter (noise2_q2.png)**:
    - The histogram after the median filter is somewhat similar to the mean filter but maintains more variation in intensity values.
    - The median filter is effective at reducing noise while preserving edges, which is likely reflected in the slightly more varied histogram compared to the mean-filtered image.

Key Points

- **Noise Characteristics**: The nature of the noise in **noise2.png** is different from the typical 'salt and pepper' noise, as indicated by the original histogram. Therefore, the effects of the mean and median filters might be different compared to the first image.

- **Effect on Image Content**: Both filters alter the pixel value distribution, aiming to reduce noise. The mean filter creates a more uniform distribution, while the median filter preserves more of the original image's variation.
- **Detail Preservation**: The median filter is generally better at preserving image details compared to the mean filter, which is likely reflected in the histograms.