1 Pre-Recorded Tasks

1.1 Linear Filtering

1. A 1D low-pass filter $h[n]$ shall be used for filtering an image. The impulse response of the filter is given as

$$h[n] = \frac{1}{4} \cdot [1\ 2\ 1], \quad n = [-1\ 0\ 1].$$

For 1-dimensional filtering, this filter shall now be applied to the image $x[n_1, n_2]$ in order to get the filtered values $y[n_1, n_2]$. Calculate the filtered values for the marked pixels!

$$x[n_1, n_2] = \begin{bmatrix} 10 & 10 & 20 & 20 \\ 10 & \boxed{10} & \boxed{20} & 20 \\ 10 & \boxed{10} & \boxed{30} & 30 \\ 5 & 5 & 40 & 40 \end{bmatrix}$$

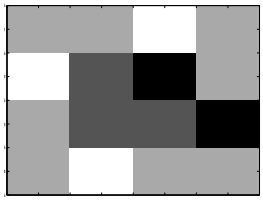
2. For filtering at the border the filter will reach outside the image. What can we do?

3. For an image of size $M \times M$ and a filter of length K, how many operations do we need for filtering? Find the solution for 1D and 2D separable filtering! How many operations do we need for non-separable filtering with a filter of size $K \times K$?

1.2 Non-linear Median Filter

The following matrix of picture values was transmitted using a high-frequency band. So called “Salt and Pepper Noise” was added. (Assumption: 10 & 20: Transmitted pixel values, 0: “Pepper”, 30: “Salt”)

$$x[n_1, n_2] = \begin{bmatrix} 20 & 20 & 30 & 20 \\ 30 & 10 & 0 & 20 \\ 20 & 10 & 10 & 0 \\ 20 & 30 & 20 & 20 \end{bmatrix}$$



Now, a median filtering with the following mask shall be applied:

$$S = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

For the edges, the filter mask shall be adapted and less pixel values shall be taken into account for filtering.

1. As a first step, determine which window sizes occur due to this kind of border handling!

2. Now calculate the median image!

3. Which window dimensions occur when the edge behavior is changed to ’last-value’?

4. What is the advantage of ’last-value’ when compared to window size adaptation, especially when you count the number of elements?

2 Self-Study Matlab Tasks

2.1 Implement a function for linear filtering

Write a function for performing a linear filtering operation. Use the script `filter_image_test.m` as a test setup!

1. Write a function `filter_image(x,h)` which accepts a 2D gray scale image $x[n_1, n_2]$ of size $M \times N$ and an 1D filter $h[i]$ of length I. The filter is non-causal (the center of the filter is in the middle). The function should perform the following tasks:

   - Extract the sizes M, N, I from the input matrices!
   - Go through all pixels in loops[1] and ignore those at the border! (at least for now)
   - Calculate the filtered values for each of these pixels!

   Test your function by using the signal and filter given in the first problem. Compare your results with the built-in filter function from MATLAB `imfilter(x,h)` and your previously calculated results!

2. For filtering at the border the regular code will not work. Extend your function `filter_image(x,h)` to handle filtering at the border. Implement one method of your choice (e.g. zero-padding, pixel-replication, kernel-truncation, border mirroring, cyclic extension)!

[1] In general, this is a really bad idea in MATLAB, but in this case we do it for the purpose of better illustration!

1 Pre-Recorded Tasks

1.1 Linear Filtering

1. For 1-dimensional filtering, this filter shall now be applied to the image $x[n_1, n_2]$ in order to get the filtered values $y[n_1, n_2]$. Calculate the filtered values for the marked pixels!

   $y[2,2] = \frac{50}{4}$, $y[2,3] = \frac{70}{4}$, $y[3,2] = \frac{60}{4}$, $y[3,3] = \frac{100}{4}$

2. For filtering at the border the filter will reach outside the image. What can we do?

   We can extend the image with different values:

   - Zero-padding: Values outside will be assumed to be zero. Unfortunately, this generates attenuation at the image border and does not produce good results.
   - Pixel replication: The edge pixels are repeated.
   - Border mirroring: If multiple pixels on the outside are required, we can flip the image at the border.
   - Cyclic extension: If we filter with the DFT a cyclic extension will happen. This is not reasonable for images as the left and right border have nothing in common.

3. For an image of size $M \times M$ and a filter of length K, how many operations do we need for filtering? Find the solution for 1D and 2D separable filtering! How many operations do we need for non-separable filtering with a filter of size $K \times K$?

   Operations are often counted in MAC (multiply accumulate) operations. Filtering needs to be done for every pixel $M \times M$:

   - 1D filtering: $K \cdot M^2$ operations
   - Separable 2D filtering: $2 \cdot K \cdot M^2$ operations
   - Non-separable 2D filtering: $K^2 \cdot M^2$ operations

1.2 Non-linear Median Filter

1. As a first step, determine which window sizes occur due to this kind of border handling!

   $2 \times 2 = 4$, $2 \times 3 = 6$, $3 \times 2 = 6$, $3 \times 3 = 9$

2. Now calculate the median image!

$$\begin{bmatrix} 20 & 20 & 20 & 20 \\ 20 & 20 & 10 & 15 \\ 20 & 20 & 10 & 15 \\ 20 & 20 & 15 & 15 \end{bmatrix}$$

3. Which window dimensions occur when the edge behavior is changed to 'last-value'?

   $3 \times 3 = 9$

4. What is the advantage of 'last-value' when compared to window size adaptation, especially when you count the number of elements?

   The number of values in the sorting vector is odd in all cases, i. e., there is no need for calculating the average of the two values in the middle. Therefore, the resulting pixel value is already present in the input image.

2 Self-Study Matlab Tasks

2.1 Implement a function for linear filtering

Write a function for performing a linear filtering operation. Use the script `filter_image_test.m` as a test setup!

1. Write a function `filter_image(x,h)` which accepts a 2D gray scale image $x[n_1, n_2]$ of size $M \times N$ and an 1D filter $h[i]$ of length I. The filter is non-causal (the center of the filter is in the middle). The function should perform the following tasks:

   - Extract the sizes M, N, I from the input matrices!
   - Go through all pixels in loops[1] and ignore those at the border! (at least for now)
   - Calculate the filtered values for each of these pixels!

   Test your function by using the signal and filter given in the first problem. Compare your results with the built-in filter function from MATLAB `imfilter(x,h)` and your previously calculated results!

   - Values at the border don't match.

2. For filtering at the border the regular code will not work. Extend your function `filter_image(x,h)` to handle filtering at the border. Implement one method of your choice (e.g. zero-padding, pixel-replication, kernel-truncation, border mirroring, cyclic extension)!

[1] In general, this is a really bad idea in MATLAB, but in this case we do it for the purpose of better illustration!