TDI2131 Digital Image Processing (Week 4)
Tutorial 3

Note: All images used in this tutorial belong to the Image Processing Toolbox.

1. Spatial Filtering (by hand)

(a) Below is an 8-bit grayscale image of size 8x8 pixels. Apply the following filters listed below to the inner portion of the image (which means only pixels within the inner 6x6 portion is filtered) and show the resulting image.

<table>
<thead>
<tr>
<th>150</th>
<th>170</th>
<th>190</th>
<th>190</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
<td>250</td>
<td>170</td>
<td>180</td>
<td>11</td>
<td>13</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>175</td>
<td>160</td>
<td>170</td>
<td>175</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>175</td>
<td>185</td>
<td>185</td>
<td>170</td>
<td>13</td>
<td>11</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>100</td>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
</tr>
<tr>
<td>100</td>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
<td>60</td>
<td>100</td>
</tr>
</tbody>
</table>

(i) (ii) (iii) Basic Gradient operator (iv) Sobel operators
\[
\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \Delta f \equiv |G_x| + |G_y|
\]

(b) Show the resulting image if the following filter is applied to the same 8x8 pixel image in (a) with zero padding at the image borders.
\[
\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}
\]

2. Simple Averaging

(a) Simply create a 5x5 pixel matrix,
\[
>> x = \text{magic}(5);
\]
\[
\begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{bmatrix}
\]

It's not necessary to understand the function magic, but go ahead if you wish to know!
If we perform filtering with a 3x3 mask by default (no zero padding), we should be expecting a 3x3 image as output. Let’s start from the top left corner of the image.

The average of the first 3x3 portion from the top left corner:

\[
\text{mean2}(x(1:3,1:3))
\]
and moving on to the second neighborhood one step to the right,

\[
\text{mean2}(x(1:3,2:4))
\]
and so on...

As we continue moving on, we should eventually obtain all 9 averaged values in the 3x3 output image:

\[
\begin{array}{ccc}
11.1111 & 10.8889 & 12.8889 \\
11.0000 & 13.0000 & 15.0000 \\
13.1111 & 15.1111 & 14.8889
\end{array}
\]

Looks like we may need to write some loops to scan through the rows and columns involved, otherwise it will be so tedious...

3. Filtering with \texttt{filter2}

But Matlab has provided a function to help us do that. The \texttt{filter2} function does the job of linear filtering for us. Its usage:

\[
\text{filter2} \text{(filter, image, shape)}
\]

and the result is a matrix of datatype \texttt{double}. The parameter \texttt{shape} is optional; it describes the method for dealing with the image borders.

(a) Let’s start with averaging filters. Start by creating the filter.

\[
\text{a=ones(3,3)/9;}
\]
This creates a 3x3 matrix of values 1/9 each.

Read an image into \texttt{x}, and perform filtering with filter \texttt{a} using zero padding,

\[
\text{filter2(a,x,'same');}
\]
Zero padding is the default setting, if the option is not specified, it will use this setting.

Filter only the inside portion of the image

\[
\text{filter2(a,x,'valid');}
\]
The result will always be smaller than the original.

This method creates filters by \textbf{manually specifying the filter matrix}. 

\texttt{JS/2010 2}
(b) You can also create predefined filters using the fspecial function. This function has many options, which allows easy creation of many different filters. However, the drawback is the limitation to the freedom to create or design your own filters.

```matlab
>> fspecial('average',3)
>> fspecial('average',11)
```

If you leave out the final number, the 3x3 averaging filter is returned by default.

```matlab
>> c = imread('cameraman.tif');
>> f1 = fspecial('average');
>> cf1 = filter2(f1,c);
```

We now have a matrix of type double. To display this, we can do any of the following:
- Transform it to a matrix of type uint8 for use with imshow
- Divide its values by 255 to obtain a matrix with normalized values in the 0.1-1.0 range for use with imshow.

![Image](image1.png)

Notice how zero padding used at the edges has resulted in a dark border appearing around the image. This is even more noticeable when a larger filter is used.

Test out different sizes of the averaging filter. What do you observe when the filter size increases?

(c) The default predefined Laplacian filter from fspecial function,

```matlab
>> f = fspecial('laplacian')
```

yields the mask

```
0.1667 0.6667 0.1667
0.6667 -3.333 0.6667
0.1667 0.6667 0.1667
```

By filtering the same image with the Laplacian filter, a highpass filter,

```matlab
>> cf = filter2(f,c);
```

it is expected that the output graylevel values may be outside the range of 0 – 255. Check out the max and min values in the image to see if this is true,
Use the function `mat2gray` to linearly scale the matrix elements to displayable values (0 – 255, or 0.0 to 1.0 for normalized values).

```matlab
>> cf_scaled = mat2gray(cf);
>> imshow(cf_scaled);
```

Or alternatively, to obtain a better result for display purposes, we can divide the result of the filtering by a constant just before displaying it,

```matlab
>> figure, imshow(cf/100);
```

4. Write a simple program to filter `cameraman.tif` with the following smoothing, sharpening and gradient filters. Display the output images, and make comparisons between them.

**Lowpass filters (smoothing):**

(a) \[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

(b) \[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 2 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

(c) \[
\begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{bmatrix}
\]

**Highpass filters (sharpening):**

(d) \[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 9 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]

(e) \[
\begin{bmatrix}
1 & -2 & 1 \\
-2 & 5 & -2 \\
1 & -2 & 1 \\
\end{bmatrix}
\]

(f) \[
\begin{bmatrix}
0 & -1 & 0 \\
-1 & 5 & -1 \\
0 & -1 & 0 \\
\end{bmatrix}
\]

**Gradient filters:**

(g) Basic Gradient operators

(h) Sobel operators

(Hint: After filtering the image, remember to convert your image back to `uint8` or rescale it appropriately to display the image correctly)
5. We know that in the concept of unsharp masking,
   \[ \text{sharpened image} = \text{original image} - \text{blurred image} \]
   and, we also refer to that as
   \[ \text{highpass-filtered image} = \text{original image} - \text{lowpass-filtered image} \]

Steps:
1. Using this method, obtain the highpass-filtered (sharpened) image using the lowpass-filtered (smoothened) image.
2. To correct the effect of featureless background which is a common output of highpass filters, either
   - Add the original image and highpass-filtered image together
   - Re-scale the highpass-filtered image accordingly. Requires trial-and-error.

Write a program to test the result with rice.tif using an averaging filter as your lowpass filter.

(Note: You can further experiment with other lowpass filters such as Gaussian filter, which is predefined with fspecial)

6. High-boost filters are obtained by
   \[ \text{high-boost image} = A \times \text{original image} - \text{lowpass-filtered image} \]
   where \( A \) is an amplification or boost factor.

Write a program to implement high-boost filtering with rice.tif (or any other image of your choice). Test and compare the results with \( A = 0.5, 1, 1.5, 3 \).

Homework

1. Experiment with median filtering (non-linear) using medfilt2 function.

2. Can unsharp masking be used to reverse the effects of blurring? Apply an unsharp masking filter after a 3x3 averaging filter and describe the result.