CV201, HW #4

Oren Freifeld and Meitar Ronen
The Department of Computer Science, Ben-Gurion University

Release Date: 7/12/2019
Submission Deadline: 14/1/2020, 20:00

Abstract

Please make sure you have carefully read the general instructions in the course website at https://www.cs.bgu.ac.il/~cv201/HW_Instructions. These instructions address, among other things, what to do or not do with figures, Jupyter Notebooks, *etc.*, as well as what compression files are legit or not.

This HW assignment is focused on image filters.

Version Log

- 1.02, 22/12/2019. Problem 5: In the footnote, referred to HW #5 for the definition of a matrix group.
- 1.01, 7/12/2019. Fixed deadline (from Jan 2019 to Jan 2020)
- 1.00, 7/12/2019. Initial release.

Contents

1 A Few Words Before You Start

1. Throughout all computer exercises below, do not implement the convolutions yourself; rather, use standard implementations such as those in scipy.ndimage or OpenCV (*e.g.*, cv2.filter2D). For oft-used filters, there

are usually already functions that will spare you the need to pass the filter as an argument (*e.g.*, cv2.GaussianBlur).

2. Get the data:
   https://www.cs.bgu.ac.il/~cv201/wiki.files/hw4_data.tar

3. Some of the data is saved as (Matlab) mat files. To read these in python, use scipy.io.loadmat, which returns a dictionary. For example:

```
Code:
import scipy.io as sio
mdict = sio.loadmat("are_these_separable_filters.mat")
print ([k for k in mdict.keys() if not k.startswith("__")])
Output:
['K3', 'K2', 'K1']
```

   Then you can get the data more directly (in effect, as numpy arrays) via:

```
K1=mdict["K1"]
K2=mdict["K2"]
K3=mdict["K3"]
```

2 Filtering

2.1 Convolution

Let I and h be two 2D digital arrays, of possibly different sizes. We will usually think of I as the input image, and of h as a filter, which is usually smaller.

Definition 1 (The correlation operator) *The* ***correlation operator****,* $\otimes$*, creates a new image,* $I \otimes h$*, via*

$$(I \otimes h)(i,j) = \sum_{k,l} I(i+k, j+l)h(k,l) \overset{\substack{k'=i+k \\ l'=j+l}}{=} \sum_{k',l'} I(k',l')h(k'-i, l'-j) \quad \Diamond$$

Definition 2 (The convolution operator) *The* ***convolution operator****,* $*$*, creates a new image,* $I * h$*, via*

$$(I * h)(i,j) = \sum_{k,l} I(i-k, j-l)h(k,l) \overset{\substack{k'=i-k \\ l'=j-l}}{=} \underbrace{\sum_{k',l'} I(k',l')h(i-k', j-l')}_{(I \otimes h_{\text{flipped}})(i,j)}$$

$$= (h * I)(i,j) \quad \Diamond$$

The summation in the definitions above is done over all relevant pixels (*i.e.*, where h is defined), and it is assumed that the central pixel of h is the origin; *e.g.*, if h is 3×3, the notation $\sum_{k,l} I(i-k, j-l)h(k,l)$ is short for

$$\sum_{k=-1}^{1} \sum_{l=-1}^{1} I(i-k, j-l)h(k,l)\,.$$

Both the convolution and correlation operators are linear.

A slightly different way of writing the same thing is as

$$\sum_{\boldsymbol{x}_i} I(\boldsymbol{x}-\boldsymbol{x}_i)h(\boldsymbol{x}_i) = \sum_{\boldsymbol{x}_i} h(\boldsymbol{x}-\boldsymbol{x}_i)I(\boldsymbol{x}_i) \tag{1}$$

where $\boldsymbol{x}$ is the pixel under consideration. Note that $\boldsymbol{x}$ enters the computation only via its difference from each of the $\boldsymbol{x}_i$'s.

For example, if the filter h is Gaussian, and $g = I * h$, then

$$g(\boldsymbol{x}) = \frac{1}{c}\sum_{\boldsymbol{x}_i} \exp\left(-\frac{1}{2}\frac{(\boldsymbol{x}-\boldsymbol{x}_i)^2}{\sigma^2}\right) I(\boldsymbol{x}_i) \tag{2}$$

where c typically is taken as a normalizer: $c = \sum_{\boldsymbol{x}_i} \exp\left(-\frac{1}{2}\frac{(\boldsymbol{x}-\boldsymbol{x}_i)^2}{\sigma^2}\right)$.

Remark 1 *You should recognize this type of expression from our discussion on,* e.g.*, the Lucas-Kanade method.* ♢

Remark 2 *Usually, if $g = I * h$, we consider g only on the same domain as that of I. For example, we do not bother with computing $g(-1, 17)$, since the first argument, -1, is outside the domain of I. A different question, however, is what do when computing, say, $g(1, 17)$ (assuming 1-based indexing), which is a pixel at the boundary of g; the problem is that if h is, say, 3×3, then the summation includes terms such as $I(-1, 17)h(-1, 0)$, $I(-1, 16)h(-1, -1)$, and $I(-1, 18)h(-1, 1)$ but $I(-1, 16)$, $I(-1, 17)$, and $I(-1, 18)$ are undefined. In other words, we need to decide how to handle boundary effects. There are several ways to go about it. For example, treating I as zero at every out-of-domain pixel. In this class (unlike standard classes on signal/mage processing) we will not explore the implications of the various choices one can make for handling the boundary effects.* ♢

Problem 1 (convolution is associative) *Let a, b and c be three digital images. Show that*

$$(a * b) * c = a * (b * c)$$

(thus we can drop the parentheses and simply write $a * b * c$). ♢

Definition 3 (2D discrete-domain impulse signal)

$$\delta(i, j) = \begin{cases} 1 & \text{if } i = 0 \text{ and } j = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

♢

A filter h is also called the "impulse response" (of some system) since its convolution with an impulse signal is $h * \delta = \delta * h = h$.

Problem 2 *Show that $h * \delta = \delta * h = h$.* ♢

Computer Exercise 1 *The image used here is **mandrill.png**. Display it and the results for convolving it with each one of the following filters: an isotropic 7×7 Gaussian with $\sigma = 3$; an isotropic 21×21 Gaussian with $\sigma = 10$; a uniform 21×21 blur kernel (*e.g.*, in Python: h=np.ones((21,21))/(21**2)).* ♢

`Some Python commands you may find useful for the exercise above: scipy.ndimage.gaussian_filter; scipy.ndimage.uniform_filter; cv2.blur,cv2.GaussianBlur.`

2.1.1 Viewing convolution as as one big matrix operation

Let $x \in \mathbb{R}^{M\times N}$ be an input image. Let $h \in \mathbb{R}^{m\times n}$. Let $y = x * h$. Since convolution is linear, and since y is also $M \times N$, we may rewrite this operation as

$$\underbrace{\boldsymbol{y}}_{(MN)\times 1} = \underbrace{\boldsymbol{H}}_{(MN)\times(MN)} \underbrace{\boldsymbol{x}}_{(MN)\times 1} \tag{4}$$

where $\boldsymbol{x}$ is a vectorized version of x and $\boldsymbol{y}$ is a vectorized version of y. The values of $\boldsymbol{H}$ are determined by h. If h is small (or large but sparse) then $\boldsymbol{H}$ is sparse (*i.e.*, most of its elements are zero).

Problem 3 *Assume a zero-boundary condition. In effect, in 1-based indexing, we set $I(i-k, j-l) = 0$ whenever at least one of the following is true: $i-k \le 0$; $i-k > M$; $j-l \le 0$; $j-l > N$. Assume the vectorization is done by stacking the rows on top of each other, and that $h \in \mathbb{R}^{3\times 3}$:*

$$\begin{aligned}
\boldsymbol{x} &= \begin{bmatrix} x(1,1) & \dots & x(1,N) & \dots & x(M,1) & \dots & I(M,N) \end{bmatrix}^T ; \\
\boldsymbol{y} &= \begin{bmatrix} y(1,1) & \dots & y(1,N) & \dots & y(M,1) & \dots & y(M,N) \end{bmatrix}^T ; \\
h &= \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & h_{3,3} \end{bmatrix} \in \mathbb{R}^{3\times 3} .
\end{aligned}$$

Write the expression for the entries of $\boldsymbol{H}$; i.e.*, write down the expression for $\boldsymbol{H}_{i,j}$ for every i and j such that $i \in \{1, \dots, MN\}$ and $j \in \{1, \dots, MN\}$ where $\boldsymbol{H}_{i,j}$ is the entry of $\boldsymbol{H}$ in the i^{th} row and j^{th} column. Note that most these entries are zero;* i.e.*, $\boldsymbol{H}$ is (very) sparse.* ♢

Computer Exercise 2 *In continue to the previous exercise, let $M = 12$ and $N = 16$. Implement $\boldsymbol{H}$ for the three different filters,*

$$h_1 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad h_2 = \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad h_3 = h_2^T , \tag{5}$$

and display (using imshow with "interpolation" set to "None"; use colorbar and informative titles) the resulting $\boldsymbol{H}$ matrices (as opposed to, say, displaying some image convolved with h).

Also note that: h_1 is a uniform blur filter; h_2 is the horizontal Sobel filter; h_3 is the vertical Sobel filter.

Remark: if M and N are much larger (e.g.,500), we can still easily implement $\boldsymbol{H}$ using a sparse matrix (e.g., using scipy.sparse.lil). But for displaying purposes, we will have to convert it to a dense matrix, and then memory might be an issue. ♢

`Some Python commands you may find useful in the exercise above:`
`a.ravel; a.reshape (when a is a numpy array); plt.title; plt.colorbar.`

Problem 4 *Let*

$$h = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} . \tag{6}$$

What is the effect of convolving an image with this h? (optional: try it yourself on a computer to verify your answer) ♢

2.1.2 Separable Filters

If we are given two 1D filters, we can build a separable filter from them (see the practical session). Now we will address the other direction: given a 2D filter, we would like to know whether it is separable or not. We will do this using SVD (defined below).

Definition 4 (a square orthogonal matrix) *An $n \times n$ matrix $\boldsymbol{Y}$ is called orthogonal if $\boldsymbol{Y}\boldsymbol{Y}^T = I_{n\times n}$.* ♢

Problem 5 *Let $\boldsymbol{Y}$ be an orthogonal square matrix. Let $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ denote the columns of $\boldsymbol{Y}$.*

Part (i) *Can you uniquely determine $\det(\boldsymbol{Y})$? If not, is there still something you can say about $\det(\boldsymbol{Y})$?*

Part (ii) *Show that $\boldsymbol{Y}^T$ is an orthogonal square matrix.*

Part (iii) *Let $i \in \{1, \ldots, n\}$. Find $\boldsymbol{y}_i^T \boldsymbol{y}_i$.*

Part (iv) *Let $i \in \{1, \ldots, n\}$. Find $\|\boldsymbol{y}_i\|_{\ell_2}$.*

Part (v) *Let $i, j \in \{1, \ldots, n\}$ where $i \neq j$. Find $\boldsymbol{y}_i^T \boldsymbol{y}_j$. What is the angle between $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$?*

Part (vi) *Fix n, a positive integer. Show that $n \times n$ orthogonal matrices form a matrix group[1]. This group is called the orthogonal group – that is the standard name, but perhaps the "orthonormal group" would be a more informative name.* ♢

Definition 5 (the Singular Value Decomposition (SVD) of a square matrix) *The SVD of an $n \times n$ matrix $\boldsymbol{A}$ is*

$$\boldsymbol{A} = \underbrace{\boldsymbol{U}}_{n\times n}\underbrace{\boldsymbol{S}}_{n\times n}\underbrace{\boldsymbol{V}^T}_{n\times n} \tag{7}$$

$\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal square matrices. Columns of $\boldsymbol{U}$ (resp. $\boldsymbol{V}$) are called the left (right) singular vectors of $\boldsymbol{A}$. The matrix $\boldsymbol{S}$ is diagonal. Its diagonal elements, $\{s_i\}_{i=1}^n$, are nonnegative and called the singular values of $\boldsymbol{A}$. SVD routines usually return the singular values sorted with $s_1 \geq s_2 \geq s_n$. ♢

Fact 1 *Let $\boldsymbol{K}$ be a square 2D filter. It is separable if and only if its SVD has exactly one non-zero singular value.* ♢

Example 1 *The SVD of a 3×3 $\boldsymbol{K}$ is*

$$\begin{aligned}\boldsymbol{K} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T &= \begin{bmatrix} \boldsymbol{u}_1 & \boldsymbol{u}_2 & \boldsymbol{u}_3 \end{bmatrix} \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 \end{bmatrix}^T \\ &= \sum_{i=1}^{3} s_i \boldsymbol{u}_i \boldsymbol{v}_i^T \,. \end{aligned} \tag{8}$$

[1] For the definition of a matrix group, see the beginning of HW #5.

If $s_1 > 0$ *and* $s_2 = s_3 = 0$ *then*

$$\boldsymbol{K} = s_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T \tag{9}$$

and $\boldsymbol{K}$ *is evidently separable, with,* e.g.*,*

- *a vertical filter* $\sqrt{s_1}\boldsymbol{u}_1$.
- *a horizontal filter* $\sqrt{s_1}\boldsymbol{v}_1^T$. ◊

Computer Exercise 3 *Get* K_1*,* K_2 *and* K_3 *from* `are_these_separable_filters.mat`*. Using svd, determine, based on the singular values of these filters, which one is separable and which is not. Remark: due to numerical errors, you are not going to get perfect zeros even if the filter is separable. You will have to settle for treating very small numbers (*e.g.*,* 10^{-12}*) as zero.* ◊

`Some Python commands you may find useful for the exercise above: scipy.linalg.svd, np.diag, np.allclose.`

Fact 2 *If* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *are two, possibly-rectangular, matrices such that their matrix product,* AB*, is defined, then*

$$\mathrm{rank}(AB) \leq \min(\mathrm{rank}(A), \mathrm{rank}(B))\,.$$

◊

Problem 6 *Let* $\boldsymbol{K}$ *be an* $n \times n$ *separable filter. Is the* $n \times n$ *matrix* $\boldsymbol{K}$ *invertible? (note well, you are not asked whether the operation of convolving an image with* $\boldsymbol{K}$ *is invertible).* ◊

2.2 More General Filters

The convolution operator (similarly to the correlation operator) is not just a linear operation, it is also space invariant (also called location invariant).

More generally, we can define the following operation: I and h is given by

$$g(i,j) = \sum_{k,l} I(i-k, j-l) h_{i,j}(k,l)\,. \tag{10}$$

The difference from convolution is that here the filter depends on the location, (i,j). For example, consider a situation where we want to blur the image but also have reasons to expect more noise on the right part of the image. In which case, we may want to use a Gaussian filter such that its standard deviation increases with x. While this filter is space-dependent (and thus applying it is not a convolution with some filter), this filter is still linear and still does not depend on I.

An even more general approach is to adapt the filter to the image itself; *i.e.*, make it a function of I. For example, suppose we want to blur an image but do

it in a way that preserves edges (more on that later in this document). There are several ways to go about it. One of them is what is called bilateral filtering:

$$g(\boldsymbol{x}) = \frac{1}{c}\sum_{\boldsymbol{x}_i} f^r(|I(\boldsymbol{x}) - I(\boldsymbol{x}_i)|) f^d(|\boldsymbol{x} - \boldsymbol{x}_i|) I(\boldsymbol{x}_i) \tag{11}$$

$$c = \sum_{\boldsymbol{x}_i} f^r(|I(\boldsymbol{x}) - I(\boldsymbol{x}_i)|) f^d(|\boldsymbol{x} - \boldsymbol{x}_i|) \tag{12}$$

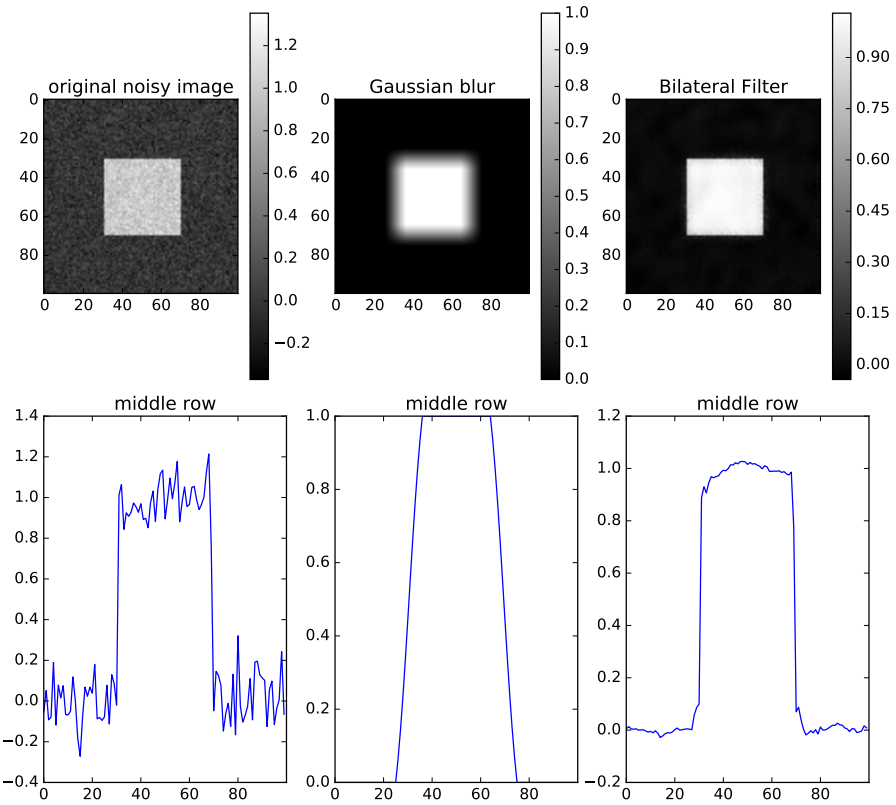
where f_r controls the weight as a function of the distance in the range (*i.e.*, intensity difference) and f_d controls the weight as as function of the distance in the domain (*i.e.*, spatial distance) For example we can take both these to be Gaussians, of different variances:

$$g(\boldsymbol{x}) = \frac{1}{c}\sum_{\boldsymbol{x}_i} \exp\left(-\frac{1}{2}\frac{I(\boldsymbol{x}) - I(\boldsymbol{x}_i))^2}{\sigma_r^2}\right) \exp\left(-\frac{1}{2}\frac{(\boldsymbol{x} - \boldsymbol{x}_i)^2}{\sigma_d^2}\right) I(\boldsymbol{x}_i) \tag{13}$$

$$c = \sum_{\boldsymbol{x}_i} \exp\left(-\frac{1}{2}\frac{I(\boldsymbol{x}) - I(\boldsymbol{x}_i))^2}{\sigma_r^2}\right) \exp\left(-\frac{1}{2}\frac{(\boldsymbol{x} - \boldsymbol{x}_i)^2}{\sigma_d^2}\right). \tag{14}$$

Problem 7 *Is bilateral filtering a linear operation? Explain.* ◊

The main purpose of the bilateral filter is to preserve edges while blurring, as shown in the figure below:



Computer Exercise 4 *Get the (noisy) image from* `bilateral.mat` *(shown in the left column in the figure above), and, using bilateral filtering and playing with its parameters, get a result similar to the one in the rightmost figure above.* ♢

`A Python command you may find useful for the exercise above: cv2.bilateralFilter`

2.3 Laplacian and Laplacian of Gaussian

Problem 8 *Calculate, by taking analytical derivatives, the Laplacian of an isotropic Gaussian (in a continuous setting);* i.e.*,*

$$\nabla^2 G(x,y,\sigma) \triangleq \frac{\partial^2 G(x,y,\sigma)}{\partial x^2} + \frac{\partial^2 G(x,y,\sigma)}{\partial y^2} = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) G(x,y,\sigma)\,,$$

where

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2}\exp(-\frac{x^2+y^2}{2\sigma^2})\,,$$

and show it is indeed given by

$$\nabla^2 G(x,y,\sigma) = \left(\frac{x^2+y^2}{\sigma^4} - \frac{2}{\sigma^2}\right) G(x,y,\sigma)\,.$$

(an expression you saw in class). ♢

Since $G(x,y,\sigma) = G(x^2+y^2,\sigma)$ (a sloppy notation indicating that $G(x,y,\sigma)$ depends on x and y only through $r^2 \triangleq x^2+y^2$) and since $\left(\frac{x^2+y^2}{\sigma^4} - \frac{2}{\sigma^2}\right)$ can be written as another function of x^2+y^2 (and σ), it is obvious that, at a given point (x,y), $\nabla^2 G(x,y,\sigma)$ is invariant w.r.t. spatial rotations of I about this point. It is tempting to think that this is solely because G is rotational invariant. However, it turns out that the Laplacian of I itself is also rotationally invariant:

Problem 9 *Let*

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

and show that, independently of the value of $\theta \in [0, 2\pi)$*, it always holds (assuming* I *is twice differentiable) that*

$$\nabla^2 I(x,y) = \nabla^2 I(x'(x,y), y'(x,y))\,,$$

Equivalently,

$$\frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2} = \frac{\partial^2 I(x'(x,y),y'(x,y))}{\partial (x')^2} + \frac{\partial^2 I(x'(x,y),y'(x,y))}{\partial (y')^2}\,.$$

♢

Two 3×3 simple discrete approximations of the Laplacian operator are convolutions with one of the following filters:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}. \tag{15}$$

Consistently with the previous exercise, we can see some (approximated) radial symmetry (more so in the second filter).

2.4 Directional Filters

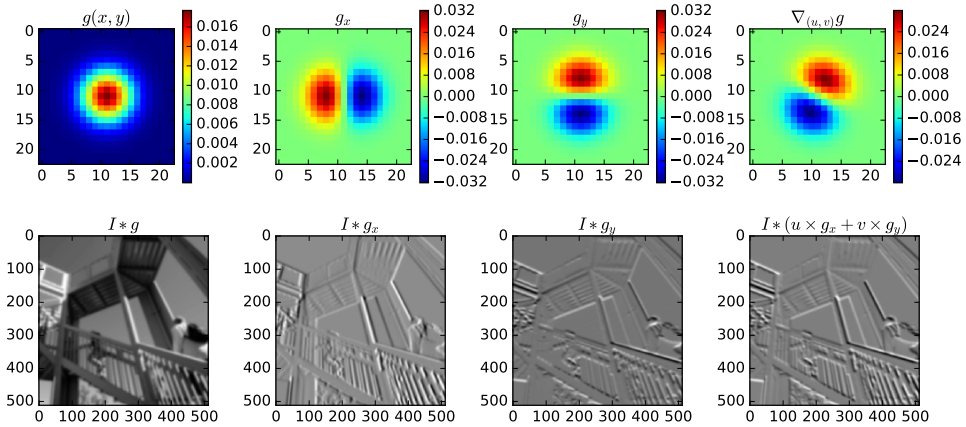
Let

$$g(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{x^2+y^2}{\sigma^2}\right).$$

Note that:

$$\frac{\partial}{\partial x}g(x,y) = -\frac{x}{\sigma^2}g(x,y) \qquad \frac{\partial}{\partial y}g(x,y) = -\frac{y}{\sigma^2}g(x,y) \tag{16}$$

The top row of the figure below shows g, $\frac{\partial}{\partial x}g$, $\frac{\partial}{\partial x}g$ and $u\frac{\partial}{\partial x}g + v\frac{\partial}{\partial y}g$ where $u = \cos\theta$ and $v = \sin\theta$ with $\theta = 115°$. In other words, the last filter provides an approximated directional derivative in the (u,v) direction. The second row shows the results of convolving an image (from `ascent.jpg`) with these filters.



Computer Exercise 5 *The image used here is* `ascent.jpg`. *Create 5 filters that correspond to the directional derivatives in 5 different angles of your choosing. Show the filters and the results of applying them to the image.* ♢

`Some Python commands you may find useful in the exercise above: np.mgrid, np.exp,np.cos,np.sin, a.sum() (where a is a numpy array), ndimage.convolve, plt.subplots_adjust`

2.5 Computing the Spatial Derivatives (will be Required in a Later Assignment for Optical Flow)

Computer Exercise 6 *Get img1 from* `imgs_for_optical_flow.mat`. *Using derivative filters, compute, for I=img1, the following images:*

$$I_x = \frac{\partial}{\partial x}I = I * h_x, \quad I_y = \frac{\partial}{\partial y}I = I * h_y,$$
$$I_{xx} = \frac{\partial^2}{\partial x^2}I = I * h_{xx}, \quad I_{yy} = \frac{\partial^2}{\partial y^2}I = I * h_{yy}.$$

where h_x is a filter which approximates the derivative in the x direction, h_y is a filter which approximates the derivative in the y direction, h_{xx} is a filter which approximates the second derivative in the x direction, and h_{yy} is a filter which approximates the second derivative in the y direction. You may want to blur[1] *I a little by convolving it with g_0, a Gaussian of small standard deviation, σ_0, before taking these derivatives, or, equivalently, convolve I with the derivatives of g_0.* ♢

A Python command you may find useful for the exercise above:
cv2.getDerivKernels.

Note that the output of cv2.getDerivKernels will in fact be two 1D filters, which you can use as input to cv2.sepFilter2D. Alternatively, you can take these two 1D filters and create a 2D filter from them:

```
h1,h2=cv2.getDerivKernels(1,0,3)
h1.shape is (3, 1)
h2.shape is (3, 1)
h2.dot(h1.T) then gives
array([[-1.,  0.,  1.],
       [-2.,  0.,  2.],
       [-1.,  0.,  1.]], dtype=float32)
```

Regarding how to compute the second derivative:
In principle, you can use cv2.sobel twice if you want. But it is better to do it in a single operation.E.g., suppose you are using 3x3 filters.
cv2.getDerivKernels(1,0,3) gives you the "d/dx" filter.
cv2.getDerivKernels(0,1,3) gives you the "d/dy" filter.
cv2.getDerivKernels(2,0,3) gives you the "d^2/dx^2" filter.
cv2.getDerivKernels(0,2,3) gives you the "d^2/dy^2" filter.

[1] To avoid amplifying noise and/or spatial "aliasing" – the latter is a topic covered in signal/image processing classes.