Digital images

Øyvind Ryan

Mar 9, 2017

Basic facts

Light is electromagnetic radiation with wavelengths in the range 400–700 nm (1 nm is 10^{-9} m): Violet has wavelength 400 nm and red has wavelength 700 nm. White light contains roughly equal amounts of all wave lengths.

The resolution of a medium is the number of dots per inch (dpi). The number of dots per inch for monitors is usually in the range 70–120, while for printers it is in the range 150–4800 dpi. The horizontal and vertical densities may be different. On a monitor the dots are usually referred to as *pixels* (picture elements).

The resolution of a scanner usually varies in the range 75 dpi to 9600 dpi, and the color is represented with up to 48 bits per dot.

The number of pixels recorded by a digital camera usually varies in the range 320×240 to 6000×4000 with 24 bits of color information per pixel. The total number of pixels varies in the range 76 800 to 24 000 000 (0.077 megapixels to 24 megapixels).

Digital image

A digital image P is a matrix of *intensity values* $\{p_{i,j}\}_{i,j=1}^{M,N}$. For grey-level images, the value $p_{i,j}$ is a single number, while for color images each $p_{i,j}$ is a vector of three or more values. If the image is recorded in the rgb-model, each $p_{i,j}$ is a vector of three values,

$$p_{i,j} = (r_{i,j}, g_{i,j}, b_{i,j}),$$

that denote the amount of red, green and blue at the point (i, j).

Figure: Our test image.

Figure: Black and white (left), and grey-level (right) versions of the image in Figure 9.1.

In these notes the intensity values $p_{i,j}$ are assumed to be real numbers in the interval $[0, 1]$. For color images, each of the red, green, and blue intensity values are assumed to be real numbers in $[0, 1]$.

The pixels of an image are assumed to be square with sides of length one, with the pixel with value $p_{i,j}$ centered at the point (i, j).

Reading, writing and displaying images

```
X = double(imread('filename.fmt', 'fmt'))
imshow(uint8(X))
imwrite(uint8(X), 'filename.fmt', 'fmt')
```

Use the python module `images`.

Normalising the intensities

The simple linear function

$$g(x) = \frac{x - a}{b - a}, \quad a < b,$$

maps the interval $[a, b]$ to $[0, 1]$. In particular $g(x) = x/255$ maps $[0, 255]$ to $[0, 1]$. More generally, we perform computations that result in intensities outside the interval $[0, 1]$. We can then compute the minimum and maximum intensities p_{min} and p_{max} and map the interval $[p_{\text{min}}, p_{\text{max}}]$ back to $[0, 1]$.

```
function Z=mapto01(X)
  minval = min(min(min(X)));
  maxval = max(max(max(X)));
  Z = (X - minval)/(maxval-minval);
end
```

Extracting the different colors

```
img = double(imread('lena.png'));
```

```
X1 = zeros(size(img));
X1(:,:,1) = img(:,:,1);

X2 = zeros(size(img));
X2(:,:,2) = img(:,:,2);

X3=zeros(size(img));
X3(:,:,3) = img(:,:,3);
```

Figure: The red, green, and blue components of the color image.

Converting from color to grey-level

We replace the three color values (r, g, b) by a single value p that represent the grey level. Several possibilities:

1. Use the largest of the three color components:

2. Use the average of the three color components.

3. Use the length of the component vector (needs to be normalised).

```
X1 = max(img, [], 3);

X2 = (img(:, :, 1) + img(:, :, 2) + img(:, :, 3))/3;

X3 = sqrt(img(:,:,1).^2 + img(:,:,2).^2 + img(:,:,3).^2);
X3 = 255*mapto01(X3);
```

Figure: Alternative ways to convert a color image to a grey level image. The result is mapped to $(0, 1)$.

Computing the negative image

Replace an intensity p by its 'mirror value' $1 - p$.

Figure: The negative versions of an image.

Increasing the contrast

A common problem with images is that the contrast often is not good enough: a large proportion of the grey values are concentrated in a rather small subinterval of $[0, 1]$.

Solution: spread out the values by applying a function to the intensity values. This function should have a large derivative in the areas where the intensity values are concentrated.

The functions in the left plot are all on the form

$$f_n(x) = \frac{\arctan\big(n(x-1/2)\big)}{2\arctan(n/2)} + \frac{1}{2}.$$

The functions in the right plot are all on the form

$$g_\epsilon(x) = \frac{\ln(x+\epsilon) - \ln\epsilon}{\ln(1+\epsilon) - \ln\epsilon},$$

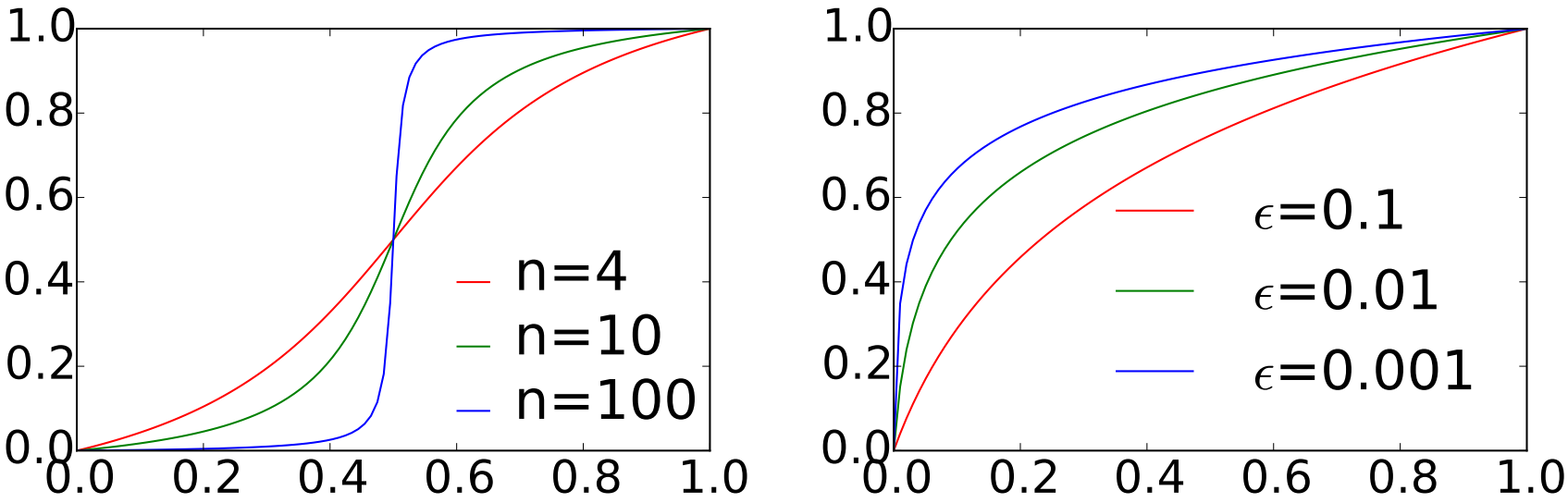

Figure: Some functions that can be used to improve the contrast of an image.

Figure: The middle functions have been applied to a grey-level version of the test image.

```
function Z=contrastadjust(X,epsilon)
  Z = X/255; % Maps the pixel values to [0,1]
  Z = (log(Z+epsilon) - log(epsilon))/...
           (log(1+epsilon)-log(epsilon));
  Z = Z*255; % Maps the values back to [0,255]
end
```

Computational molecules

We say that an operation S on an image X is given by the *computational molecule*

$$A = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & a_{-1,-1} & a_{-1,0} & a_{-1,1} & \cdots \\ \cdots & a_{0,-1} & \underline{a_{0,0}} & a_{0,1} & \cdots \\ \cdots & a_{1,-1} & a_{1,0} & a_{1,1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

if we have that

$$(SX)_{i,j} = \sum_{k_1,k_2} a_{k_1,k_2} X_{i-k_1,j-k_2}.$$

In the molecule, indices are allowed to be both positive and negative, we underline the element with index $(0,0)$ (the center of the molecule), and assume that $a_{i,j}$ with indices falling outside those listed in the molecule are zero (as for compact filter notation).

Filtering and computational molecules, Theorem 9.9

Let S_1 and S_2 be filters with compact filter notation $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, respectively, and consider the operation S where S_1 is first applied to the columns in the image, and then S_2 is applied to the rows in the image. Then S is an operation which can be expressed in terms of the computational molecule $a_{i,j} = (\boldsymbol{t}_1)_i(\boldsymbol{t}_2)_j$.

the combined filtering operation, denoted S, takes the form

$$S(X) = S_1 X (S_2)^T,$$

Applying S_1 to the columns of X is what we call a *vertical filtering operation*. Applying S_2 to the rows of X is what we call a *horizontal filtering operation*. The order of vertical and horizontal filtering of an image does not matter.

Computing $S(X)$

Assume that the image is stored as the matrix `X`. In Exercise 9.13 you will implement `tensor_impl` which computes the transformation $S(X) = S_1 X (S_2)^T$, where `X`, `S1`, and `S2` are input.

```
S1 = @(x) filterS(t1, x, 1);
S2 = @(x) filterS(t2, x, 1);

Y = tensor_impl(X, S1, S2)
```

Tensor product of vectors

If $\boldsymbol{x}, \boldsymbol{y}$ are vectors of length M and N, respectively, their tensor product $\boldsymbol{x} \otimes \boldsymbol{y}$ is defined as the $M \times N$-matrix defined by $(\boldsymbol{x} \otimes \boldsymbol{y})_{i,j} = x_i y_j$. In other words, $\boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{x}\boldsymbol{y}^T$.

Observation: Let $\mathcal{E}_M = \{\boldsymbol{e}_i\}_{i=0}^{M-1}$ $\mathcal{E}_N = \{\boldsymbol{e}_i\}_{i=0}^{N-1}$ be the standard bases for $\mathbb{R}^M$ and $\mathbb{R}^N$. Then

$$\mathcal{E}_{M,N} = \{\boldsymbol{e}_i \otimes \boldsymbol{e}_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$$

is a basis for $L_{M,N}(\mathbb{R})$, the set of $M \times N$-matrices. This basis is often referred to as the standard basis for $L_{M,N}(\mathbb{R})$.

Tensor product of matrices, Definition 9.13

If $S_1 : \mathbb{R}^M \to \mathbb{R}^M$ and $S_2 : \mathbb{R}^N \to \mathbb{R}^N$ are matrices, we define the linear mapping $S_1 \otimes S_2 : L_{M,N}(\mathbb{R}) \to L_{M,N}(\mathbb{R})$ by linear extension of $(S_1 \otimes S_2)(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) = (S_1 \boldsymbol{e}_i) \otimes (S_2 \boldsymbol{e}_j)$. The linear mapping $S_1 \otimes S_2$ is called the tensor product of the matrices S_1 and S_2.

Compact filter notation and computational molecules, Theorem 9.14

If $S_1 : \mathbb{R}^M \to \mathbb{R}^M$ and $S_2 : \mathbb{R}^N \to \mathbb{R}^N$ are matrices of linear transformations, then $(S_1 \otimes S_2)X = S_1 X (S_2)^T$ for any $X \in L_{M,N}(\mathbb{R})$. In particular $S_1 \otimes S_2$ is the operation which applies S_1 to the columns of X, and S_2 to the resulting rows. In other words, if S_1, S_2 have compact filter notations $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, respectively, then $S_1 \otimes S_2$ has computational molecule $\boldsymbol{t}_1 \otimes \boldsymbol{t}_2$.

Composing tensor products, Corollary 9.15

We have that $(S_1 \otimes T_1)(S_2 \otimes T_2) = (S_1 S_2) \otimes (T_1 T_2)$.

Smoothing an image, Example 9.9

Let us consider computational molecules where both filters are lowpass. $S = \frac{1}{4}\{1, \underline{2}, 1\}$ (row 2 from Pascal's triangle) gives the computational molecule

$$A = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & \underline{4} & 2 \\ 1 & 2 & 1 \end{pmatrix}.$$

This means that we compute the new pixels by

$$\hat{p}_{i,j} = \frac{1}{16}\big(4p_{i,j} + 2(p_{i,j-1} + p_{i-1,j} + p_{i+1,j} + p_{i,j+1}) + p_{i-1,j-1} + p_{i+1,j-1} + p_{i-1,j+1} + p_{i+1,j+1}\big).$$

If we instead use the filter $S = \frac{1}{64}\{1, 6, 15, \underline{20}, 15, 6, 1\}$ (row 6 from Pascal's triangle), we get the computational molecule

$$\frac{1}{4096}\begin{pmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 20 & 120 & 300 & \underline{400} & 300 & 120 & 20 \\ 15 & 90 & 225 & 300 & 225 & 90 & 15 \\ 6 & 36 & 90 & 120 & 90 & 36 & 6 \\ 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{pmatrix}.$$

Implementations

```
X1=CreateExcerpt();
imshow(uint8(X1))
```

```
shortmolecule = @(x) filterS([1 2 1]/4, x, 1);
X2 = tensor_impl(X1, shortmolecule, shortmolecule);
imshow(uint8(X2))
```

```
S2 = conv([1 3 3 1]/8, [1 3 3 1]/8);
longmolecule = @(x) filterS(S2, x, 1);

X3 = tensor_impl(X1, longmolecule, longmolecule);
imshow(uint8(X3))
```

Figure: The two right images show the effect of smoothing the left image.

Smoothing a simple image

Figure: The results of smoothing the simple image to the left with the filter $\frac{1}{4}\{1, \underline{2}, 1\}$ horizontally, vertically, and both, respectively.

Partial derivative in x-direction

We use the familiar symmetric Newton quotient approximation for the partial derivative:

$$\frac{\partial P}{\partial x}(i,j) \approx \frac{p_{i,j+1} - p_{i,j-1}}{2},$$

This corresponds to applying the bass-reducing filter $S = \frac{1}{2}\{1, \underline{0}, -1\}$ to all the rows (alternatively, applying the tensor product $I \otimes S$ to the image).

Observation: The partial derivative $\partial P/\partial x$ of the image can be computed with the computational molecule

$$\frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ 1 & \underline{0} & -1 \\ 0 & 0 & 0 \end{pmatrix}.$$

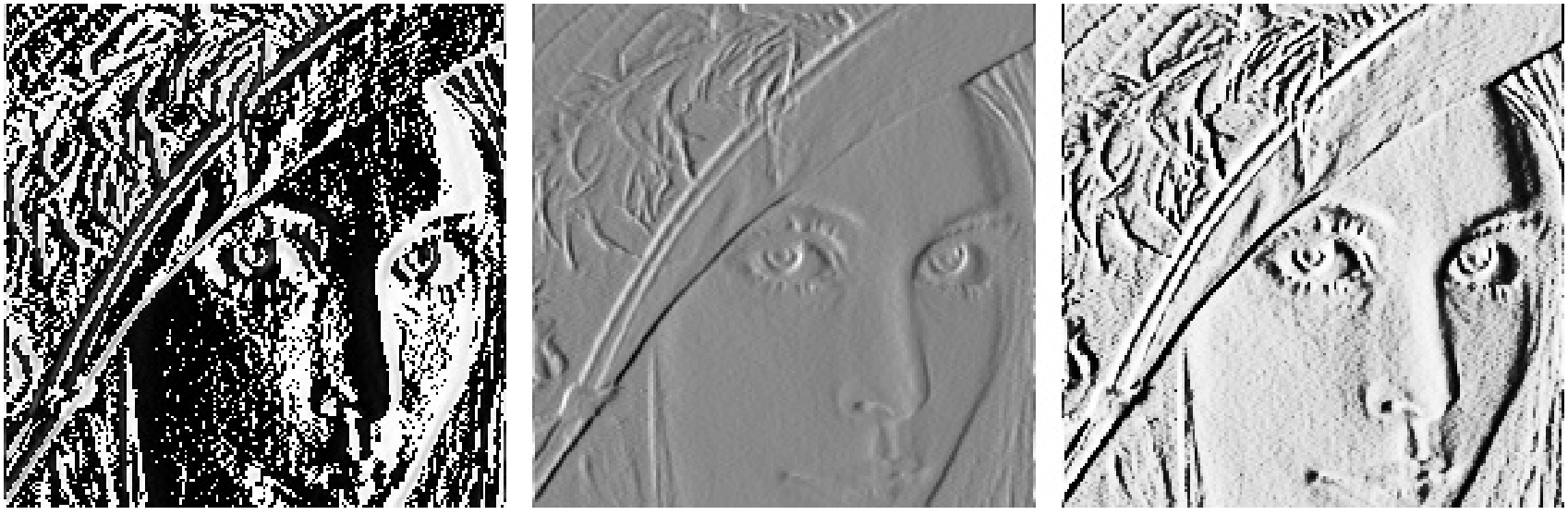

Figure: Experimenting with the partial derivative in the x-direction. The left image has artefacts, since the pixel values are outside the legal range. We therefore normalize the intensities to lie in $[0, 1]$ (middle), before we increase the contrast (right).

Partial derivative in y-direction

Observation: The partial derivative $\partial P/\partial y$ of the image can be computed with the computational molecule

$$\frac{1}{2}\begin{pmatrix} 0 & 1 & 0 \\ 0 & \underline{0} & 0 \\ 0 & -1 & 0 \end{pmatrix}.$$

The gradient

The gradient and it length are

$$\nabla P = \left(\frac{\partial P}{\partial x}, \frac{\partial P}{\partial y} \right)$$

$$|\nabla P| = \sqrt{\left(\frac{\partial P}{\partial x} \right)^2 + \left(\frac{\partial P}{\partial y} \right)^2}.$$

Figure: The computed gradient (left). In the middle the intensities have been normalised to the $[0, 255]$, and to the right the contrast has been increased.

Figure: The first-order partial derivatives in the x- and y-direction, respectively. In both images, the computed numbers have been normalised and the contrast enhanced.

Second-order derivatives

We use the three point approximation to the second derivative

$$\frac{\partial P}{\partial x^2}(i,j) \approx p_{i,j+1} - 2p_{i,j} + p_{i,j-1}$$

Observation: The second order derivatives of an image P can be computed by applying the computational molecules

$$\frac{\partial^2 P}{\partial x^2}: \quad \begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix},$$

$$\frac{\partial^2 P}{\partial y \partial x}: \quad \frac{1}{4}\begin{pmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{pmatrix},$$

$$\frac{\partial^2 P}{\partial y^2}: \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

Figure: The second-order partial derivatives in the xx-, xy-, and yy-directions, respectively. In all images, the computed numbers have been normalised and the contrast enhanced.

Applying to a simple image

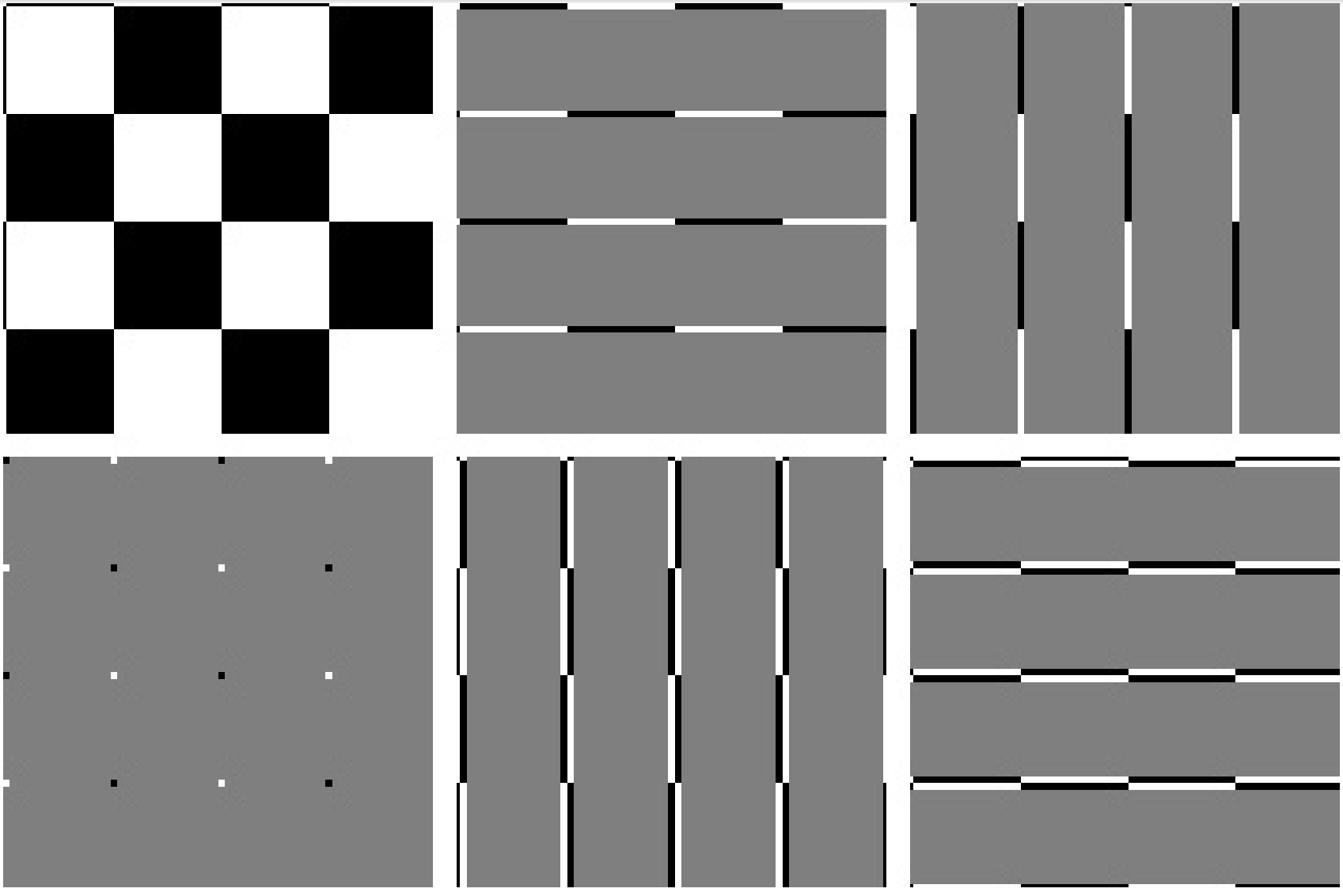

Figure: Different tensor products representing partial derivatives applied to a simple chess pattern example image (upper left). The tensor products are $S \otimes I$, $I \otimes S$, $S \otimes S$, $I \otimes S^2$, and $S^2 \otimes I$.

Bases for tensor products, Theorem 9.16

If $\mathcal{B}_1 = \{\boldsymbol{v}_i\}_{i=0}^{M-1}$ is a basis for $\mathbb{R}^M$, and $\mathcal{B}_2 = \{\boldsymbol{w}_j\}_{j=0}^{N-1}$ is a basis for $\mathbb{R}^N$, then $\{\boldsymbol{v}_i \otimes \boldsymbol{w}_j\}_{(i,j)=(0,0)}^{(M-1,N-1)}$ is a basis for $L_{M,N}(\mathbb{R})$. We denote this basis by $\mathcal{B}_1 \otimes \mathcal{B}_2$.

Coordinate matrix

Let $\mathcal{B} = \{\boldsymbol{b}_i\}_{i=0}^{M-1}$, $\mathcal{C} = \{\boldsymbol{c}_j\}_{j=0}^{N-1}$ be bases for $\mathbb{R}^M$ and $\mathbb{R}^N$, and let $A \in L_{M,N}(\mathbb{R})$. By the coordinate matrix of A in $\mathcal{B} \otimes \mathcal{C}$ we mean the $M \times N$-matrix X (with components X_{kl}) such that $A = \sum_{k,l} X_{k,l}(\boldsymbol{b}_k \otimes \boldsymbol{c}_l)$.

Change of coordinates in tensor products, Theorem 9.18

Assume that

- $\mathcal{B}_1, \mathcal{C}_1$ are bases for $\mathbb{R}^M$, and that S_1 is the change of coordinates matrix from $\mathcal{B}_1$ to $\mathcal{C}_1$,
- $\mathcal{B}_2, \mathcal{C}_2$ are bases for $\mathbb{R}^N$, and that S_2 is the change of coordinates matrix from $\mathcal{B}_2$ to $\mathcal{C}_2$.

Both $\mathcal{B}_1 \otimes \mathcal{B}_2$ and $\mathcal{C}_1 \otimes \mathcal{C}_2$ are bases for $L_{M,N}(\mathbb{R})$, and if X is the coordinate matrix in $\mathcal{B}_1 \otimes \mathcal{B}_2$, and Y the coordinate matrix in $\mathcal{C}_1 \otimes \mathcal{C}_2$, then the change of coordinates from $\mathcal{B}_1 \otimes \mathcal{B}_2$ to $\mathcal{C}_1 \otimes \mathcal{C}_2$ can be computed as

$$Y = S_1 X (S_2)^T.$$

The change of coordinates from $\mathcal{B}_1 \otimes \mathcal{B}_2$ to $\mathcal{C}_1 \otimes \mathcal{C}_2$ can thus be implemented as follows:

- For every column in the coordinate matrix in $\mathcal{B}_1 \otimes \mathcal{B}_2$, perform a change of coordinates from $\mathcal{B}_1$ to $\mathcal{C}_1$.
- For every row in the resulting matrix, perform a change of coordinates from $\mathcal{B}_2$ to $\mathcal{C}_2$.

Change of coordinates with the DFT

We zero out small DFT coefficients:

```
X = X.*(abs(X) >= threshold);
```

Figure: The effect on an image when it is transformed with the DFT, and the DFT-coefficients below a certain threshold are zeroed out. The threshold has been increased from left to right, from 100, to 200, and 400. The percentage of pixel values that were zeroed out are 76.6, 89.3, and 95.3, respectively.

Change of coordinates with the DCT

We zero out small DCT coefficients in the same way.

Figure: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold are zeroed out. The threshold has been increased from left to right, from 30, to 50, and 100. The percentage of pixel values that were zeroed out are 93.2, 95.8, and 97.7, respectively.

Drop splitting into blocks

Figure: The effect on an image when it is transformed with the DCT, and the DCT-coefficients below a certain threshold are zeroed out. The image has not been split into blocks here, and the same thresholds as above were used. The percentage of pixel values that were zeroed out are 93.2, 96.6, and 98.8, respectively.