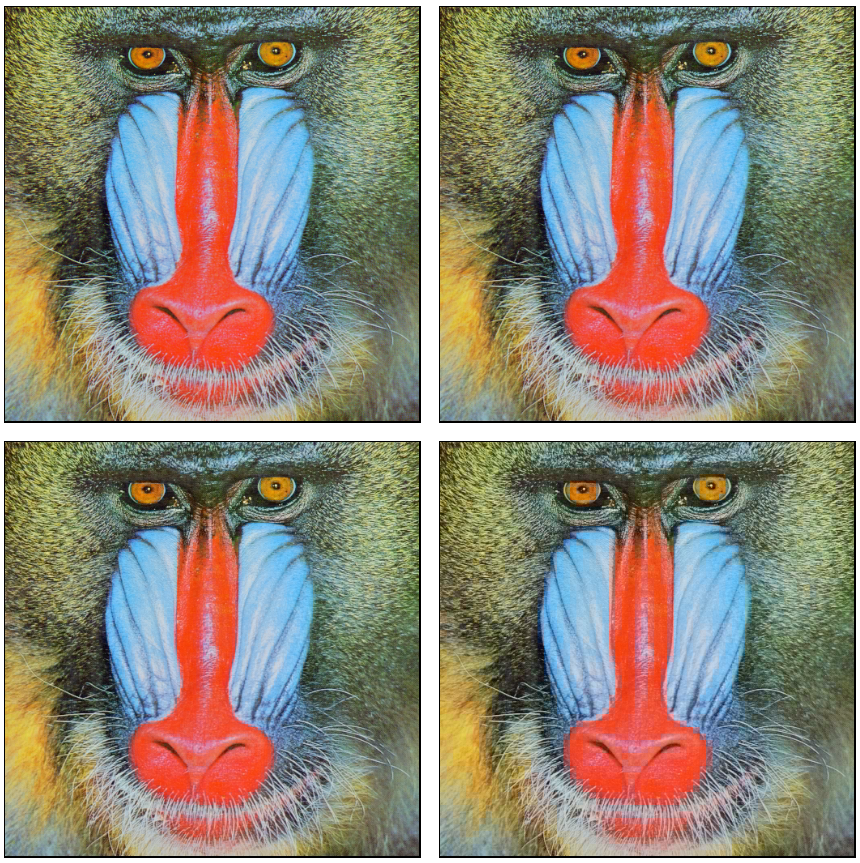
Image Processing and Computer Vision 2007/08
Example Solutions for Practical Assignments 2 (P2)

Problem 1 (Colour Spaces)

(a) The conversion from RGB to YCbCr images requires to implement the following code:

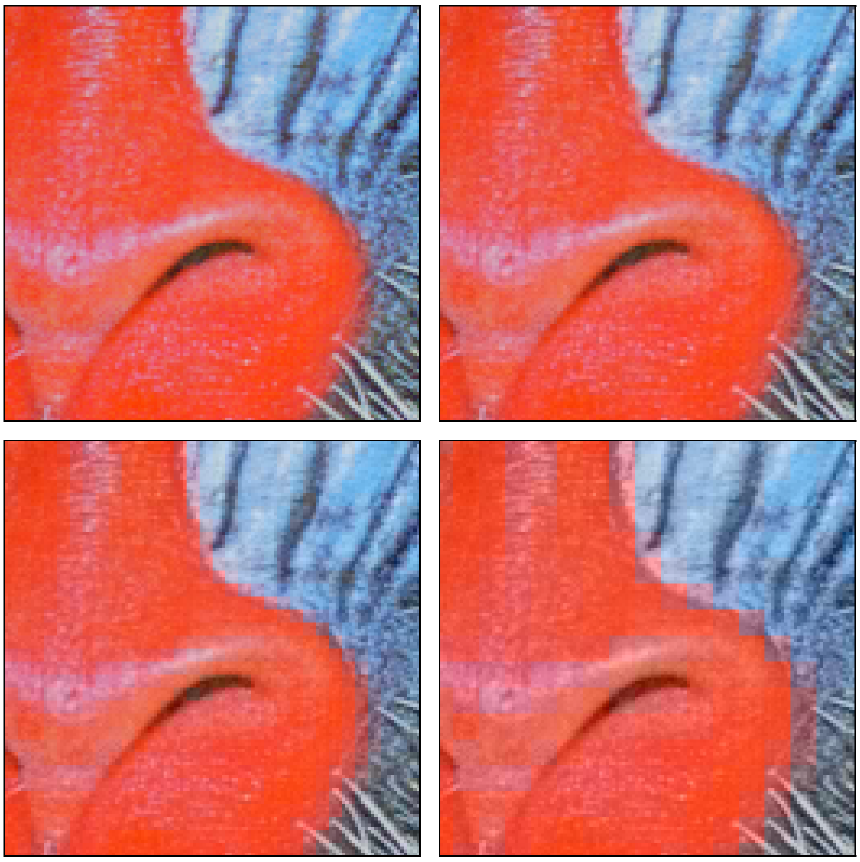
```
/* ------------------------------------------------------------ */

void RGB_to_YCbCr

     (float  **R,         /* Red channel of RGB image    */
      float  **G,         /* Green channel of RGB image  */
      float  **B,         /* Blue channel of RGB image   */
      float  **Y,         /* Y channel of YCbCr image    */
      float  **Cb,        /* Cb channel of YCbCr image   */
      float  **Cr,        /* Cr channel of YCbCr image   */
      long    nx,         /* pixel number in x-direction */
      long    ny)         /* pixel number in y-direction */
/*
  converts RGB to YCbCr colour space
*/

{
long    i, j;        /* loop variables */
float   help;        /* time saver */

help = 1.0/256.0;

/* sets higher frequencies to zero */
for (i=1;i<=nx;i++)
  for (j=1;j<=ny;j++)
  {
     Y[i][j] = 16  + help * (    65.738 * R[i][j]
                              + 129.057 * G[i][j]
                              +  26.064 * B[i][j]);

    Cb[i][j] = 128 + help * ( -  37.945 * R[i][j]
                              -  74.494 * G[i][j]
                              + 112.439 * B[i][j]);
    Cr[i][j] = 128 + help * (   112.439 * R[i][j]
                              -  94.154 * G[i][j]
                              -  18.285 * B[i][j]);
  }
 return;
}
/* ------------------------------------------------------------ */
```

(b) The compressed variants of the image `baboon.ppm` for subsampling factors of $S = 1, 2, 4$ and 8 are depicted below. As one can see, the variants for $S = 2$ and $S = 4$ provide still a quite good quality. This is due to the fact that the details (hairs, patterns) are preserved, since the Y-channel that contains these details is not compressed (downsampled). In the case of $S = 8$, however, slight block artifacts become visible.



Compressed variants of the image `baboon.ppm`. *(a) Top left:* Image for $S = 1$ (original image). *(b) Top right:* $S = 2$. *(c) Bottom left:* $S = 4$. *(d) Bottom right:* $S = 8$.

This is verified by the images below that depict a zoom of the nose region. Here also for $S = 2$ and $S = 4$ the loss of quality becomes obvious.


Zoom into the compressed variants of the image `baboon.ppm`. *(a) Top left:* Image for $S = 1$ (original image). *(b) Top right:* $S = 2$. *(c) Bottom left:* $S = 4$. *(d) Bottom right:* $S = 8$.

(c) While the original RGB image requires 24 bpp to store the information, the requiresments of the compressed (subsampled) images is given by

$$v = 8\left(1 + \frac{1}{S^2} + \frac{1}{S^2}\right) .$$

While the Y channels remain uncompressed, the Cb- and Cr-channel are reduced in both dimensions by a factor of S. Thus, 12 bpp for $S = 2$, 9 bpp for $S = 4$, and 8.25 bpp for $S = 8$ are needed. This is in the order of the memory consumption of grey value images.

Problem 2 (Discrete Cosine Transform / JPEG)

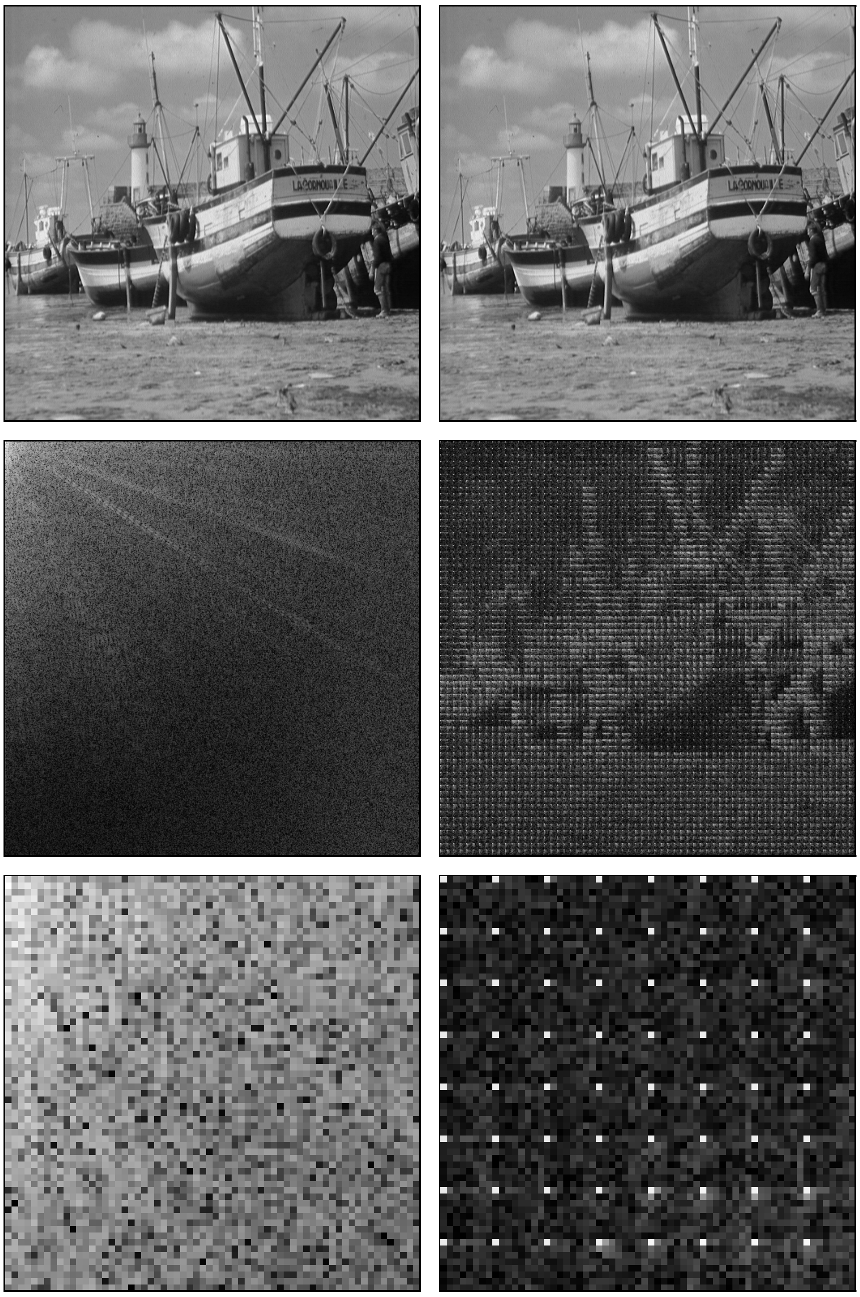
(a) The code for the discrete cosine transform (DCT) is given by

```
/* ------------------------------------------------------------ */
void DCT_2d

     (float   **u,         /* image, unchanged */
      float   **c,         /* coefficients of the DCT */
      long    nx,          /* pixel number in x-direction */
      long    ny)          /* pixel number in y-direction */
/*
  calulates DCT of input image
*/

{
long    i, j, m, p;  /* loop variables */
float   nx_1;        /* time saver */
float   ny_1;        /* time saver */
float   pi;          /* variable pi */
float   **tmp;       /* temporary image */
float   *cx,*cy;     /* arrays for coefficients */

/* ---- derive pi ---- */
pi = 2.0 * asinf(1.0);

/* ---- set time savers ---- */
nx_1 = pi / (2.0 * nx);
ny_1 = pi / (2.0 * ny);

/* ---- allocate memory ---- */
alloc_matrix (&tmp, nx, ny);
alloc_vector (&cx, nx);
alloc_vector (&cy, ny);

/* ---- set up coefficients ---- */
cx[0] = sqrt(1.0/nx);
for (p=1;p<nx;p++)
  cx[p] = sqrt(2.0/nx);

cy[0] = sqrt(1.0/ny);
for (p=1;p<ny;p++)
  cy[p] = sqrt(2.0/ny);
/* ---- DCT in y-direction ---- */
for (i=0; i<nx; i++)
 for (p=0; p<ny; p++)
     {
     tmp[i][p]=0;

     for (m=0; m<ny; m++)
         tmp[i][p] += cy[p] * u[i][m] * cosf((2.0*m+1)*p*ny_1);
     }

/* ---- DCT in x-direction ---- */

for (p=0; p<nx; p++)
 for (j=0; j<ny; j++)
     {
     c[p][j]=0;

     for (m=0; m<nx; m++)
         c[p][j] += cx[p] * tmp[m][j] * cosf((2*m+1)*p*nx_1);
     }

/* ---- free memory ---- */
disalloc_matrix (tmp, nx, ny);
disalloc_vector (cx, nx);
disalloc_vector (cy, ny);

return;
}
/* ---------------------------------------------------------- */
```

In contrast, the code for the inverse discrete cosine transform (IDCT) is given by

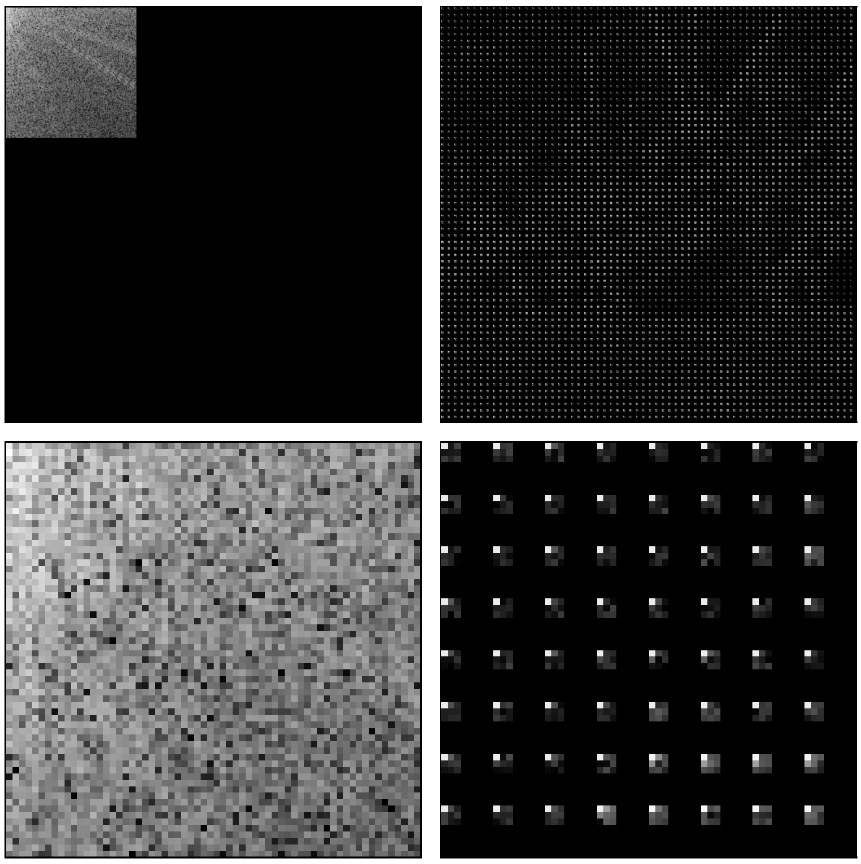
```
/* ------------------------------------------------------------ */
void IDCT_2d

     (float   **u,         /* image, unchanged */
      float   **c,         /* coefficients of the DCT */
      long    nx,          /* pixel number in x-direction */
      long    ny)          /* pixel number in y-direction */
/*
  calulates inverse DCT of input image
*/

{
long    i, j, m, p;  /* loop variables */
float   nx_1;        /* time saver */
float   ny_1;        /* time saver */
float   pi;          /* variable pi */
float   **tmp;       /* temporary image */
float   *cx,*cy;     /* arrays for coefficients */

/* ---- derive pi ---- */
pi = 2.0 * asinf(1.0);

/* ---- set time savers ---- */
nx_1 = pi / (2.0 * nx);
ny_1 = pi / (2.0 * ny);

/* ---- allocate memory ---- */
alloc_matrix (&tmp, nx, ny);
alloc_vector (&cx, nx);
alloc_vector (&cy, ny);

/* ---- set up coefficients ---- */
cx[0] = sqrt(1.0/nx);
for (m=1; m<nx; m++)
    cx[m] = sqrt (2.0 / nx);

cy[0] = sqrt(1.0/ny);
for (m=1; m<ny; m++)
    cy[m] = sqrt (2.0 / ny);
/* ---- DCT in y-direction ---- */
for (i=0; i<nx; i++)
 for (m=0; m<ny; m++)
     {
     tmp[i][m]=0;

     for (p=0; p<ny; p++)
         tmp[i][m] += cy[p] * c[i][p] * cosf((2*m+1) * p * ny_1);
     }

/* ---- DCT in x-direction ---- */
for (m=0; m<nx; m++)
 for (j=0; j<ny; j++)
     {
     u[m][j] = 0;

     for (p=0; p<nx; p++)
   u[m][j] += cx[p] * tmp[p][j] * cosf ((2*m+1) * p * nx_1);
     }

/* ---- free memory ---- */
disalloc_matrix (tmp, nx, ny);
disalloc_vector (cx, nx);
disalloc_vector (cy, ny);

return;
}
/* ------------------------------------------------------------ */
```

(b) Let us now compare the original DCT and the 8 × 8 DCT with respect to the DCT spectrum and the required runtime. To this end, we used the test image `boats.pgm`. This image and the corresponding spectra are depicted on the next page. While the spectrum of the normal DCT shows a concentration of low frequencies in the upper left corner, the 8 × 8 DCT shows essentially the same but for each block separately. Since the position of the blocks in the DCT corresponds to the position of these blocks in the image, you can still recognize some lines describing the original boats. This in turn means, that in contrast to the original DCT, at least the spatial information on the location of the different blocks is preserved. With respect to the runtimes, you notice that the 8 × 8 DCT is about 64 times faster. This is not surprising, since the DCT basis functions are only of size 8 instead of size 512.

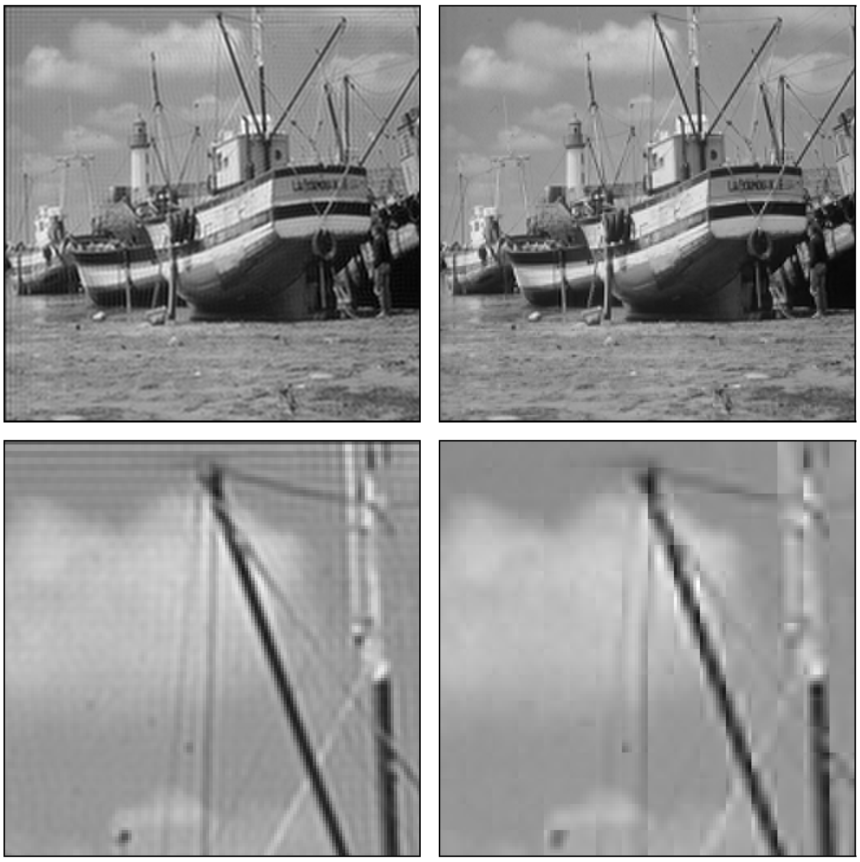


Comparison of the DCT and the 8 × 8 DCT for the image `boats.pgm`. *(a) Top left:* Original image. *(b) Top right:* Back-transformed image (no change). *(c) Centre left:* Spectrum of the DCT. *(d) Centre right:* Ditto for the 8 × 8 DCT. *(e) Bottom left:* Spectrum of the DCT (Zoom in the upper left corner). *(f) Bottom right:* Ditto for the 8 ×8 DCT.

(c) Let us now remove about 90% of all frequencies. This yields the spectra

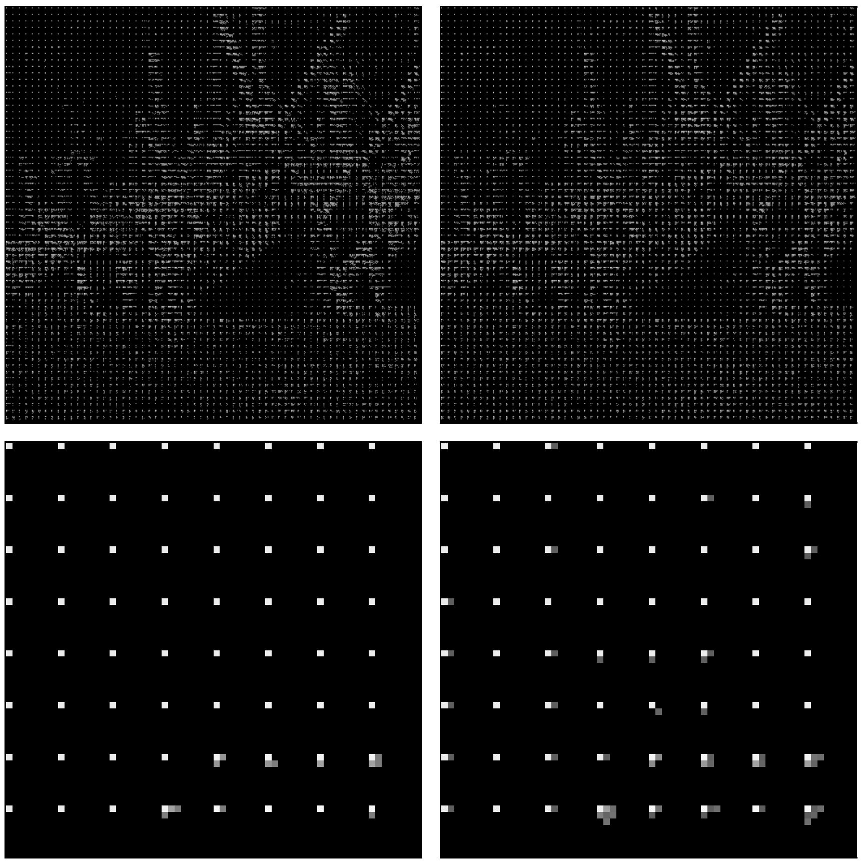

Comparison of the DCT and 8 × 8 DCT under the removal of high frequencies. *(a) Top left:* Spectrum of the DCT. *(b) Top right:* Ditto for the 8 × 8 DCT. *(c) Bottom left:* Zoom of DCT spectrum. *(d) Bottom right:* Ditto for the 8 × 8 DCT.

As one can see from the spectra, the frequencies that have been removed were exclusively high frequencies. Both, in the spectrum of the normal DCT and the spectrum of the 8 × 8 DCT only the low frequencies in the upper left corner of the spectrum/block spectrum remain. The compressed (backtransformed) images are shown on the next page. While the DCT is sharper but shows ringing artifacts due to the removal of global high frequencies, the 8 × 8 DCT image shows slight block artifacts.

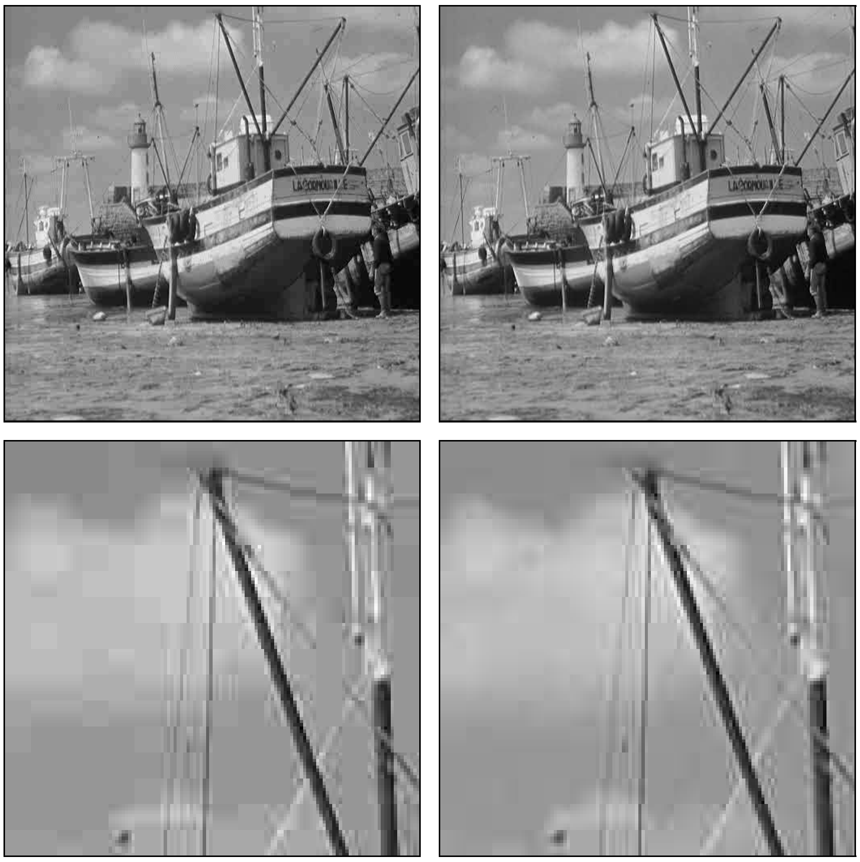


Comparison of the DCT and 8 × 8 DCT under the removal of high frequencies. *(a) Top left:* Compressed by DCT (backtransformed image). *(b) Top right:* Ditto for the 8 × 8 DCT. *(c) Bottom left:* Zoom of DCT image. *(d) Bottom right:* Ditto for the 8 × 8 DCT.

(d) Instead of removing simply the highest frequencies, one could try a more adaptive approach that removes the lowest coefficients of the 8 × 8 DCT spectrum. This can be done in terms of a quantisation step. Here, two different approaches are compared: (i) a simple approach that treats the coefficients of all frequencies equal and (ii) an approach that uses ideas from the previous task and gives more weight to low frequencies, since they are more important to the human eye. Please note that strategy (ii) is actually used by JPEG. The corresponding results that show similar compression rates as the ones in the previous task are presented on the next page.


Comparison of different quantisation strategies for the 8 × 8. *(a) Top left:* Equal treatment of all frequencies. *(b) Top right:* Giving more weight to low frequencies. *(c) Bottom left:* Zoom of (a). *(d) Bottom right:* Zoom of (b).

While the spectra look very similar, the corresponding compressed (backtransformed) images on the next page show quite different results. Although both images are relatively sharp, the one that gives more weight to low frequencies shows much less block artifacts (see e.g. the sky). This is due to the fact that storing a few more coefficients for the low frequencies significantly improves the overall result, while a few missing details (removed coefficients for the high frequencies) do not change the sharpness to much.



Comparison of different quantisation strategies for the 8 × 8. *(a) Top left:* Compressed with equal treatment of all frequencies. *(b) Top right:* Ditto for the approach that gives more weight to low frequencies. *(c) Bottom left:* Zoom of (a). *(d) Bottom right:* Zoom of (b).