SIP: Signal and Image Processing Course 2019

Week 1

Herein you will find both the hands-on exercises to help you learn the week's material and the week's assignments for which you will need to submit GROUP reports.

Report Guidelines For All Assignments

- Your answers should be complete, but concise. As an example, this week's assignment should not require more than 10 pages.
- Make sure you actually answer all parts of the questions.
- If you are asked to comment about the result, this usually means we are looking for 1 - 2 sentences that show you have understood the outcome.
- Any figures (images, graphs etc) should be at a size suitable for their purpose. For example, if you want to show pixel-level differences, then the pixels should be large enough to be seen. Use a zoomed portion of the image if necessary.
- ALL figures, images, graphs, tables etc should be numbered, (e.g. Figure 1, Table 3), and include at least a minimal caption (e.g. "Figure 1: Original image rotated by 45 degrees").
- All graph axes need to be labelled, and axes scales included.
- Images should include a labelled colour scale (a 'colorbar') where appropriate.
- All text in legends, labels, titles, scales etc should be readable, e.g. be set in a readable font size.
- When asked to show your code, this usually means code 'snippets'. That is, the few important lines of code that are the most relevant for the question. Unless specified otherwise, exclude the 'book-keeping' code (e.g. setting paths, loading/saving data etc).
- You should always submit your report as a separate PDF file and your complete code (.py files) in a compressed archive (we read .zip or .tar.gz - NOT .rar format).

Exercises

In order to ensure that you have understood the reading material, follow the code examples below for all three chapters.

THESE EXERCISES ARE NOT TO BE SUBMITTED AND WILL NOT BE ASSESSED - THEY ARE FOR YOUR OWN BENEFIT TO HELP YOU UNDERSTAND THE TOPICS.

NOTE: You will learn more if you type the code into the Python interpreter or a script, instead of just cut-and-pasting it. You may find it best to enter all the code into a single '.py' file. Use the help function to read about those functions which you have not used previously. Write down or save any results, and save any figures or resultant images on disk - it will help you remember and you can go back to it in the future.

Finally, if you don't understand something, ASK!

1. Representation

Example 1.1

Type in the following code:

```
from skimage.io import imread, imsave

A = imread('cameraman.tif') # Read in an image
print(type(A)) # The type of an image is a numpy.ndarray
print(A.shape) # Print the dimensions (in pixels) of the image
print(A.dtype) # The data type of each pixel (element in the numpy.ndarray)
imsave('cameraman.jpg', A) # Write an image
```

Where is the JPEG file saved?

What coding method was used to save the JPEG file?

Why is the file size different for the PNG and JPEG files?

Example 1.2

Type in the following code and notice the changes occurring to the image:

```
from matplotlib.pyplot import axis, imshow, show
from skimage.io import imread

# Read in intensity image
A = imread('cameraman.tif');
# Display image using imshow - but the colors are wrong!
imshow(A) # Prepare to show the image in A
show() # This opens a window for each imshow call and displays the content

# Remove axes including tick marks and labels
axis('off')
# Display intensity image in gray-scale
imshow(A, cmap='gray') # Tell matplotlib to interpret A as a gray scale image
show()
```

Example 1.3

Type in the following code and notice the changes occurring to the image:

```
from matplotlib.pyplot import axis, colorbar, imshow, show, figure
from numpy.random import rand
from skimage.io import imread

# Generate random image array in range 0-125
```

```
B = rand(256, 256)*125;
figure(1) # Open a new figure window
axis('off')
imshow(B, cmap='gray');
colorbar()
show()

# We might expect that we should not have seen intensities above 125 (an average gray color),
# but we do because imshow automatically rescales the image.
# The colorbar shows the mapping between intensities being show and the values in B.

figure(2) # Open a new figure window
axis('off')
imshow(B, vmin=0, vmax=255, cmap='gray');
colorbar()
show()
# if we specify range of data explicitly by vmin and vmax, then imshow
# displays the correct image contrast
```

Example 1.4

Type in the following code and notice the changes occurring to the image:

```
from matplotlib.pyplot import axis, imshow, show, subplot
from skimage.io import imread

# Read in 8-bit unsigned integer gray intensity image of cell
B = imread('cell.tif');
print(B.shape, B.dtype)
# Read in 16-bit unsigned integer gray intensity image of spine
C = imread('spine.tif');
print(C.shape, C.dtype)
# Read in 8-bit unsigned integer (per channel) colour image.
D = imread('onion.png');
print(D.shape, D.dtype)

# Creates a 3 by 1 mosaic of plots (first 2 parameters)
subplot(3,1,1)
# and display 1st image at sub-plot 1 (3rd parameter)
# Set colourmap to jet (false colours)
imshow(B, cmap='jet')
axis('off')

subplot(3,1,2)
# Display 2nd image at sub-plot 2
imshow(C, cmap='gray')

axis('off') # Which of the subplots does this method affect?

subplot(3,1,3)
```

```
# Display 3rd (colour) image at sub-plot 3
imshow(D)
show()

# Image of C looks wrong and we get a warning in the python interpreter.
# We will see what to do about this later
```

Try resizing the figure window to tall-and-thin and short-and-wide. Notice what happens to the different images.

Example 1.5

Type in the following code and notice the changes occurring to the image:

```
from matplotlib.pyplot import imshow, show
from skimage.io import imread
from pylab import ginput

# Read in 8-bit intensity image of cell
B = imread('cell.tif');

# Use the mouse cursor to examine the grayscale image in the interactive viewer
imshow(B, cmap='gray');
show()

# Read in 8-bit colour image.
D = imread('onion.png');

# Use the mouse cursor to examine the RGB image in the interactive viewer
imshow(D);
show()

# Use the mouse to get coordinate input to your program
imshow(D)
print('Click on 3 points in the image')
coord = ginput(3)
print('You clicked: ' + str(coord))
show()

# print pixel value at location (25,50)
print(B[25,50])
# set pixel value at (25,50) to white
B[25,50] = 255;
# view resulting changes in image
imshow(B, cmap='gray')
show()


# print RGB pixel value at location (25,50)
print(D[25,50,:])
```

```
# print only the red value at (25,50)
print(D[25,50,1])

D[25,50,:] = [255, 255, 255]; # set pixel value to RGB white
imshow(D)
# view resulting changes in image
show()
```

After changing the pixel value of B or D, does the image in the viewer update or not?

Where is the image origin? Is the order of (X, Y) the same in the interactive window as when addressing the image array in Python?

Example 1.6

Type in the following code:

```
from matplotlib.pyplot import imshow, show, subplot, title
from skimage.io import imread
# Read in 8-bit RGB colour image.
D = imread('onion.png');
# convert it to a grayscale image
D_read_as_gray = imread('onion.png', as_gray=True)
D_naive_gray = 1/3 * D[:,:,0] + 1/3 * D[:,:,1] + 1/3 * D[:,:,2]
D_better_gray = 0.2989 * D[:,:,0] + 0.5870 * D[:,:,1] + 0.1140 * D[:,:,2]

# Display all images in a 2x2 figure
subplot(2,2,1)
title('Original')
imshow(D)

subplot(2,2,2)
title('read as gray')
imshow(D_read_as_gray, cmap='gray')

subplot(2,2,3)
title('naive gray')
imshow(D_naive_gray, cmap='gray')

subplot(2,2,4)
title('better gray')
imshow(D_better_gray, cmap='gray')
show()
```

Example 1.7

Type in the following code:

```
from matplotlib.pyplot import imshow, show, subplot, title
```

```
from skimage.io import imread

# Read in 8-bit RGB colour image.
D = imread('onion.png');

# extract red channel (1st channel)
Dred = D[:,:,0]
# extract green channel (2nd channel)
Dgreen = D[:,:,1]
# extract blue channel (3rd channel)
Dblue = D[:,:,2]

# Display all images in a 2x2 figure
subplot(2,2,1)
imshow(D)

subplot(2,2,2)
imshow(Dred, cmap='gray')
title('red');

subplot(2,2,3)
imshow(Dgreen, cmap='gray')
title('green');

subplot(2,2,4)
imshow(Dblue, cmap='gray')
title('blue');

show()
```

Example 1.8

Type in the following code:

```
from matplotlib.pyplot import imshow, show, subplot, title, plot
from skimage.io import imread

I = imread('cameraman.tif') # Read in an image

imshow(I, cmap='gray') # Plot the image

# Define some points
x=[100, 100, 230]
y=[100, 200, 200]

# Plot red star markers
plot(x, y, 'r*')

# Plot green solid lines between points
plot(x[::2], y[::2], 'g-')
```

```
show() # Notice every thing is plotted in the same figure window
```

Exercise 1.1

Using the examples presented for displaying an image in Python together with those for accessing pixel locations, investigate adding and subtracting a scalar value from an individual location, i.e. I(i,j) = I(i,j) + 25 or I(i,j) = I(i,j) - 25. Start by using the grey-scale 'cell.tif' example image and pixel location (100, 20). You will need to ensure that your program does not try to create a pixel value that is larger or smaller than the pixel can hold. For instance, an 8-bit image can only hold the values 0–255 at each pixel location. What is the effect on the grey-scale colour of adding and subtracting?
Expand your technique to RGB colour images by adding and subtracting to all three of the colour channels in a suitable example image. Also try just adding to one of the individual colour channels whilst leaving the others unchanged. What is the effect on the pixel colour of each of these operations?

Exercise 1.2

Based on your answer to Exercise 1.1, use the for-loop construct in Python to loop over all the pixels in the image and brighten or darken the image.
You will need to ensure that your program does not try to create a pixel value that is larger or smaller than the pixel can hold. For instance, an 8-bit image can only hold the values 0–255 at each pixel location and similarly for each colour channel for a 24-bit RGB colour image.
NOTE: Using a for-loop to access pixels in Python is not the most efficient approach, but sometimes it is necessary and cannot be avoided.

Exercise 1.3

Using the grey-scale 'cell.tif' example image, investigate using different false colour maps to display the image. The imshow function in Python can take a range of values to specify different false colour maps: Read https://matplotlib.org/examples/color/colormaps_reference.html to get a full list of supported color maps. What different aspects and details of the image can be seen using these false colourings in place of the conventional grey-scale display?

Exercise 1.4

Load the cameraman.tif example image into Python and using the functions introduced in Example 1.1 save it once as a JPEG format file (e.g. sample.jpg) and once as a PNG format image (e.g. sample.png). Next, reload the images from both of these saved files as new images in Python, 'Ijpg' and 'Ipng'.
We may expect these two images to be exactly the same, as they started out as the same image and were just saved in different image file formats. If we compare them by computing the absolute difference of one image from the other at each pixel location we can check whether this assumption is correct.
Use the fact that images are represented as numpy arrays and that + and - operators are implemented to work as pixel-wise operations. You can compute the pixel-wise absolute value with 'numpy.fabs(x)' or vectorize the computation by using 'vectorabs = np.vectorize(lambda x: abs(x))'. Use this to create a difference image between 'Ijpg' and 'Ipng'.
The difference between these two images is not all zeros as one may expect, but a noise pattern related to the difference in the images introduced by saving in a lossy compression format (i.e. JPEG) and a lossless compression format (i.e. PNG). The difference we see is due to the image information removed in the JPEG version of the file which is not apparent to us when we look at the image.

2. Convolution

Example 2.1

Type in the following code and try to understand what is going on:

```
from matplotlib.pyplot import axis, imshow, plot, show, subplot
from skimage.io import imread
from scipy.signal import convolve
import numpy as np

# Define rectangle signal f and normalize
f = np.ones(64)
f = f / sum(f)

# Convolve f with itself to give g and normalize
g = convolve(f,f)
g = g / sum(g)

# Convolve g with itself to give h and normalize
h = convolve(g,g)
h = h / sum(h)

# Convolve h with itself to give j and normalize
j = convolve(h,h)
j = j / sum(j)

subplot(2,2,1)
plot(f,'k-')
axis('off')

subplot(2,2,2)
plot(g,'k-')
axis('off')

subplot(2,2,3)
plot(h,'k-')
axis('off')

subplot(2,2,4)
plot(j,'k-')
axis('off')

show()
```

Python function: scipy.signal.convolve
This example illustrates the repeated convolution of a 1-D uniform signal with itself. The resulting signal quickly approaches the form of the normal distribution, illustrating the central limit theorem of statistics.

Example 2.2

Type in the following code and try to understand what is going on:

```
from matplotlib.pyplot import imshow, show, subplot
from skimage.io import imread
from skimage.transform import rotate
from skimage import img_as_float
from scipy.signal import gaussian, convolve2d, correlate2d
import numpy as np

A = img_as_float(imread('trui.png')) # Read the image into an array of floats

# Define Gaussian convolution kernel
gaussian1d = gaussian(5, 2) # Construct a 1D Gaussian filter
PSF = np.outer(gaussian1d, gaussian1d) # Outer product to create 2D filter

# Define a filter for simulating motion blur
motionWidth = 21
h = np.zeros((motionWidth, motionWidth))
h[motionWidth//2, :] = 1
h = rotate(h, 45).astype(float)
h = h / np.sum(h)

# Convolve image with Gaussian filter kernel (or PSF)
B = convolve2d(PSF, A)

# Convolve with motion blur PSF
C = convolve2d(A, h)

# Self-convolution
D = convolve2d(A, A)

# Auto-correlation
E = correlate2d(A, A)

# Display original image
subplot(2,3,1)
imshow(A, cmap='gray')

# Display filtered image
subplot(2,3,2)
imshow(B, cmap='gray')

# Display filtered image
subplot(2,3,3)
imshow(C, cmap='gray')

# Display convolution image with itself
subplot(2,3,4)
imshow(D, cmap='gray')
```

```
# Display autocorrelation of the image
subplot(2,3,5)
imshow(E, cmap='gray')

show()
```

Python functions: convolute2d, gaussian, rotate, correlate2d
This example illustrates one way to do 2-D convolutions in Python (the package scipy.ndimage has other functions). The example show convolution of the image with a Gaussian kernel and a filter that simulate motion blur. The final plots shows the autoconvolution and autocorrelation of the image given by the convolution and correlation of the function with itself.

Notice the function img_as_float - scikit-image functions assumes that input images either have integer datatypes (dtype) or float in the ranges [-1;1] or [0;1]. Scikit-image provides function for converting images to specific datatypes. For more details see http://scikit-image.org/docs/stable/user_guide/data_types.html.

matplotlib.pyplot.imshow is a bit different it assumes that image datatypes are unsigned 8-bit integer of float in the range [0;1]. If this is not the case imshow performs a scaling of the intensities before plotting (the scaling can be overruled as we will see later).

Example 2.3

Type in the following code and try to understand what is going on:

```
from matplotlib.pyplot import axis, imshow, show, subplot, title
from skimage.io import imread
from scipy.signal import gaussian, convolve2d
import numpy as np

A = imread('cameraman.tif') # Read in an image
rows, cols = A.shape

# Get image dimensions
Abuild = np.zeros(A.shape)

# Construct zero image of equal size
# Randomly sample 1# of points only and convolve with gaussian PSF
sub = np.random.rand(rows, cols) < 0.01
# Generate a vector of random sample points
Abuild[sub] = A[sub]

h = gaussian(10, 2)
h = np.outer(h, h)

B10 = convolve2d(h, Abuild)

subplot(1,2,1)
axis('off')
imshow(Abuild, cmap='gray')
```

```
title('Object points')

subplot(1,2,2)
axis('off')
imshow(B10, cmap='gray')
title('Response of LSI system')

show()
```

Exercise 2.1

Using Example 2.3 we can investigate the construction of a coherent image from a limited number of point samples via the concept of the PSF.
Here, we randomly select 1% (0.01) of the image pixels and convolve them using a Gaussian-derived PSF applied as a 2-D image filter via the convolve2d() function. Experiment with this example by increasing the percentage of pixels randomly selected for PSF convolution to form the output image. At what point is a coherent image formed? What noise characteristics does this image show? Additionally, investigate the use of the gaussian() function and experiment with changing the parameters of the Gaussian filter used to approximate the PSF. What effect does this have on the convergence of the image towards a coherent image (i.e. shape outlines visible) as we increase the number of image pixels selected for convolution?

3. Pixel Processing

Example 3.1

Type in the following code and notice the changes occurring to the image:

```
from matplotlib.pyplot import imshow, show, subplot
from skimage.io import imread
import numpy as np

A=imread('cameraman.tif'); # Read in image
subplot(1,3,1), imshow(A, cmap='gray');    # Display image
B = A + 100;                    # Add 100 pixel values to image A
subplot(1,3,2), imshow(B, cmap='gray');    # Display result image B
D = np.clip(A.astype('uint16') + 100, 0, 255).astype('uint8');    # Add 100 pixel values to image A
subplot(1,3,3), imshow(D, cmap='gray');    # Display result image D

show()
```

What is the advantage of using the complicated operation in D instead of simple addition in the computation of B?

Example 3.2

Type in the following code and notice the changes occurring to the image. You will need to use the images downloaded from the course homepage (cola1.png and cola2.png):

```
from matplotlib.pyplot import imshow, show, subplot
from skimage.io import imread
from skimage import img_as_ubyte
import numpy as np

# Read in 1st image
A = imread('cola1.png')

# Read in 2nd image
B = imread('cola2.png')

subplot(2,4,1)
imshow(A) # Display 1st image

subplot(2,4,2)
imshow(B) # Display 2nd image

# subtract images and clip the result to the range of an uint8
Output = np.clip(A.astype('int16') - B, 0, 255).astype('uint8')

# subtract images and clip the result to the range of an uint8 the skimage way
Output2 = img_as_ubyte(np.clip(A.astype('int16') - B, 0, 255))
```

```
# Display result
subplot(2,4,3)
imshow(Output)
subplot(2,4,4)
imshow(Output2)

show()
```

Example 3.3

Replace the last line of Example 3.2 with the following:

```
Output2 = img_as_ubyte(np.clip(np.absolute(A.astype('int16') - B), 0, 255))

# Display result
subplot(2,4,5)
imshow(Output2)

show()
```

Compare the two subtraction methods - which is best if you want to find changes in the image?

Example 3.4

Replace the last line of Example 3.2 with the following:

```
from matplotlib.pyplot import figure

figure() # Open a new window
subplot(1,3,1)
imshow(A) # Display 1st image

# multiple image by 1.5
Output = A * 1.5

# Display result
subplot(1,3,2)
imshow(Output.astype('uint8')) # Be careful we are clipping the float part and the top range

# divide image by 4
Output2 = A / 4.0

# Display result
subplot(1,3,3)
imshow(Output2.astype('uint8')) # Be careful we are clipping the float part and the top range

show()
```

Are these results useful?

Example 3.5

Type in the following code and notice the changes occurring to the image.

```
from matplotlib.pyplot import axis, imshow, show, subplot
from skimage.io import imread

# Remember our problem with the 16 bit unsigned integer gray image from Example 1.4
# Read in 16-bit unsigned integer gray intensity image of spine
C = imread('spine.tif');
print(C.shape, C.dtype)

# In order to show the image we need to scale the intensities either to be in floating point
# in the range [0;1] or unsigned 8 bit integer in the range [0;255].
Cscaled = C.astype('float32') / C.max()

Cscaled2 = img_as_float(C)

subplot(1,3,1)
imshow(C, cmap='gray') # Previous version

subplot(1,3,2)
imshow(Cscaled, cmap='gray') # Correctly scaled

subplot(1,3,3)
imshow(Cscaled2, cmap='gray') # Correctly scaled

show()
```

Example 3.6

Type in the following code and notice the changes occurring to the image.

```
from matplotlib.pyplot import imshow, show, subplot
from skimage.io import imread
import numpy as np

# read in image
A = imread('cameraman.tif')

subplot(1,2,1), imshow(A, cmap='gray') # display image

# invert the image
B = np.invert(A)

# display result image B
subplot(1,2,2), imshow(B, cmap='gray')
```

```
show()
```

Is this method useful?

Example 3.7

Type in the following code and notice the changes occurring to the image. You will need to use the images downloaded from the course homepage (toycars1.png and toycars2.png):

```
from matplotlib.pyplot import imshow, show, subplot
from skimage.io import imread
from skimage.color import rgb2gray
from skimage import img_as_uint
import numpy as np

A = imread('toycars1.png') # Read in 1st image
B = imread('toycars2.png') # Read in 2nd image

subplot(4,2,1), imshow(A)
subplot(4,2,2), imshow(B)

subplot(4,2,3)
D = np.clip(np.abs(A.astype('int16') - B), 0, 255).astype('uint8') # Conversion 1
imshow(D)

subplot(4,2,4)
E = img_as_uint(np.abs(A.astype('int16') - B)) # Conversion 2
imshow(E)


Abw = np.where(rgb2gray(A) <= 0.5, 0, 1) # convert to binary image
Bbw = np.where(rgb2gray(B) <= 0.5, 0, 1)  # convert to binary image

# We need to convert to float for imshow to work
subplot(4,2,5), imshow(Abw.astype(float), cmap='gray')
subplot(4,2,6), imshow(Bbw.astype(float), cmap='gray')

Output = Abw ^ Bbw # Perform XOR of images
subplot(4,2,7), imshow(Output.astype(float), cmap='gray') # Display result

show()
```

Notice the following:

- the images are first converted to binary using thresholding.
- the operators for AND is '&' whilst the operator for OR is '|' and are applied in infix notation form as A & B, A | B. Similarly the operator for the XOR operation is '^'.

Is the result as you would have expected?

Example 3.8

Type in the following code and notice the changes occurring to the image.

```
from matplotlib.pyplot import imshow, show, subplot
from skimage.io import imread
from skimage.color import rgb2gray
import numpy as np

I=imread('trees.tif') # Read in 1st image
thresratio = 0.1
T=np.where(rgb2gray(I) <= thresratio, 0, 1) # perform thresholding

subplot(1,2,1), imshow(I.astype('uint8')) # Display original image
subplot(1,2,2), imshow(T.astype('float'), cmap='gray') # Display thresholded image

show()
```

Try a few different thresholds and see how the output changes.

Example 3.9

Type in the following code and notice the changes occurring to the image.

```
from matplotlib.pyplot import imshow, show, subplot
from skimage.io import imread
from skimage import img_as_float
import numpy as np

I = imread('cameraman.tif')

subplot(2,2,1), imshow(I, vmin=0, vmax=255, cmap='gray')

Id = img_as_float(I)
Output1 = 2*np.log(1 + Id)
Output2 = 3*np.log(1 + Id)
Output3 = 5*np.log(1 + Id)

subplot(2,2,2), imshow(Output1, vmin=0, vmax=3.5, cmap='gray')
subplot(2,2,3), imshow(Output2, vmin=0, vmax=3.5, cmap='gray')
subplot(2,2,4), imshow(Output3, vmin=0, vmax=3.5, cmap='gray')

show()
```

Which areas of the image gain more visible detail, and which lose it?

Example 3.10

Type in the following code and notice the changes occurring to the image.

```
from matplotlib.pyplot import imshow, show, subplot
from skimage.io import imread
from skimage import img_as_float
import numpy as np

I = imread('cameraman.tif')

subplot(2,2,1), imshow(I, vmin=0, vmax=255, cmap='gray')

Id = img_as_float(I)
Output1 = 4*(((1+0.3)**(Id)) - 1)
Output2 = 4*(((1+0.4)**(Id)) - 1)
Output3 = 4*(((1+0.6)**(Id)) - 1)

subplot(2,2,2), imshow(Output1, vmin=0, vmax=2.5, cmap='gray')
subplot(2,2,3), imshow(Output2, vmin=0, vmax=2.5, cmap='gray')
subplot(2,2,4), imshow(Output3, vmin=0, vmax=2.5, cmap='gray')

show()
```

Which areas of the image gain more visible detail, and which lose it?

Example 3.11

Type in the following code and notice the changes occurring to the image.

```
from matplotlib.pyplot import imshow, show, subplot
from skimage.io import imread
from skimage import img_as_float
import numpy as np

I = imread('cameraman.tif')

subplot(2,2,1), imshow(I, vmin=0, vmax=255, cmap='gray')

Id = img_as_float(I)
Output1 = 2*(Id**0.5)
Output2 = 2*(Id**1.5)
Output3 = 2*(Id**3.0)

subplot(2,2,2), imshow(Output1, cmap='gray')
subplot(2,2,3), imshow(Output2, cmap='gray')
subplot(2,2,4), imshow(Output3, cmap='gray')

show()
```

Which areas of the image gain more visible detail, and which lose it?

Assignment

Complete all questions. Include your own code and resultant images in your report. Make sure you adhere to the report guidelines (see first page), and the deadline as specified on the course homepage.

1.1

Write a program to display a random 20-by-20 pixel gray scale image. In the figure, make sure that the axes are labelled 'X' and 'Y', the tick marks are visible and also labelled 0-19. Use the Python function ginput (see Example 1.5) to record the selection of a pixel using the mouse, and display that pixel's X, Y co-ordinates, and then change the colour of the selected pixel to black. Include your code and a screenshot of the figure in your report.

1.2

Select one of the greyscale images from the "Test Images" folder on the course homepage. Implement a program to perform the bit-slicing technique as described in the lecture slides, and extract/display the resulting bit-plane images (see illustration in lecture slides) as separate images. Include your Python code in your report, along with a figure showing the 8 bit-planes (hint: use the subplot function).

1.3

Pick one of your own photos and import it into Python (use 'png', 'tif' or 'jpg' format) and check that it is in RGB format. Using the skimage.color.rgb2hsv function, write a program to display the individual hue, saturation and value channels of your (or any other) RGB colour image. You may wish to refer to the reading material (Gonzales & Woods) Figure 6.16. Include the Python code and resultant images in your report.

1.4

Using a combination of multiplication and addition of numpy arrays implement a function for blending two images A and B into a single image with corresponding blending weights w_A and w_B such that output image C is
C = w_A . A + w_B . B
Experiment with different example images and also with blending information from a sequence of images (e.g. the toycars images found in the "Test images" folder i Absalon). How could such a technique be used in a real application?

1.5

Write a Python function that takes an RGB image as input and converts it to a gray scale image (do not use the skimage.color.rgb2gray function). Use the perceptually pleasing transformation mentioned in the lecture slides. Write a program that reads in a RGB color image of your choice and convert it to gray scale using your function. Plot both the color and gray scale versions of the image. Include your code and a screenshot of the figure in your report.

1.6

The engineering of image formation poses limits on the spatial resolution of an image through the digitization process. At the limits of spatial resolution certain image details are lost. Using your own image from above (Assignment see 1.3), experiment with the use of the skimage.transform.resize() function using the output_shape parameter to specify the image width and height (rows and columns). As you reduce the image size, identify details/structures that deteriorate beyond recognition/

comprehension. Save 3 examples of resizing and highlight the relevant changes in your report. Then investigate the use of resize() for enlarging your image, using all of the available interpolation (see the description of the order parameter in the documentation). Save the results for your report, and comment on the differences between them. Also report the time each of the different resizing interpolation options takes (Hint: Use the Python package timeit).

1.7

The function imshow() allows us to display an image and perform a number of different interactive operations upon it. Use this function to load the 'railway.png' example image (available in the "Test Images" folder on the course homepage) and measure the length in pixels of the railway sleepers in the foreground (front) and background (rear) of the image. Based on the discussion from the lectures, what can we determine about the relative distance to the camera of the two sleepers you have measured? If we assume the standard length of a railway sleeper to be 2.5 m, what additional information would be needed to determine the absolute distance of these items from the camera? How could this be applied to other areas, such as estimating the distance of people or cars from the camera?

1.8

Using the examples that you tried earlier on arithmetic operations (Examples 3.1 – 3.4) we can investigate the use of these operators in a variety of ways. First, load the example images 'cell.tif' and 'cameraman.tif' (available in the "Test Images" folder on the course homepage). Investigate the combined use of the skimage.transform.resize() function and shape attribute of the numpy array to resize one of the images to be the same size as the other.
Try adding both of these images together using the addition operator '+' and displaying the result. What happens? Remember from the Examples that you need to handle over- and underflow by clipping the output to fit the range of the used datatype. Repeat this assignment using the subtraction operator '-'. Include your code and a screenshot of the figure in your report.

1.9

Use the sequence of cell images ('AT3_1m4_01.tif', 'AT3_1m4_02.tif, . . . 'AT3_1m4_09.tif', 'AT3_1m4_10.tif') and write a Python program that display a mosaic of the differences (Hint: Compute the absolute difference) between consecutive images in the sequence (9 images in total). Include your code and figure in your report. You may wish to use an additional enhancement approach to improve the dynamic range of difference images. What is the result of this differencing operation? How could this be useful?
Hint. You may wish to set up an array containing each of the image file names 01 to 10.