Introduction to Image Processing
– Assignment 1 –

Exercise 1 - Scanlines & Image Profiles (2 Pts.)

Compute the profile of the image `bloodcells.tif` (see below) for a user-defined scanline.

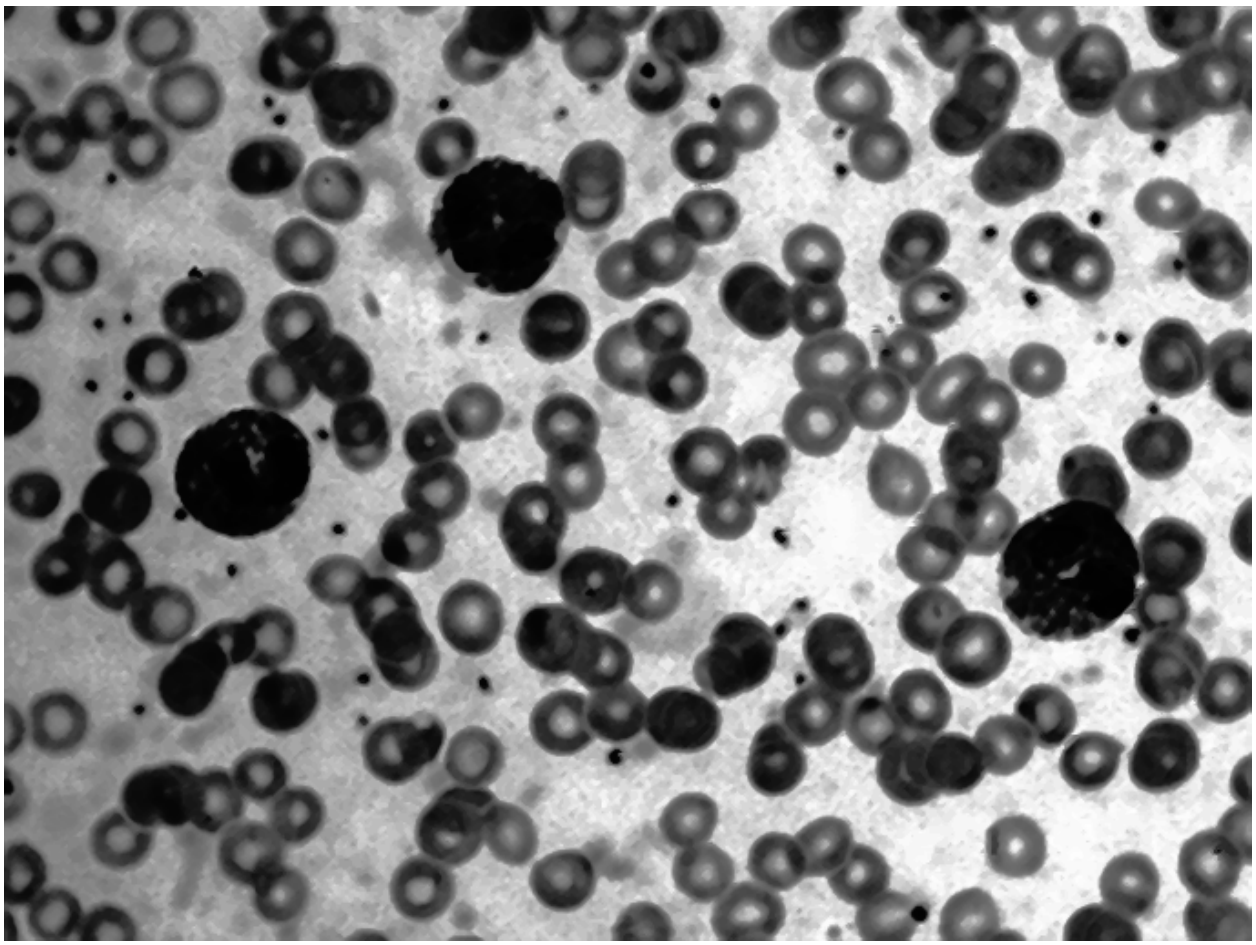
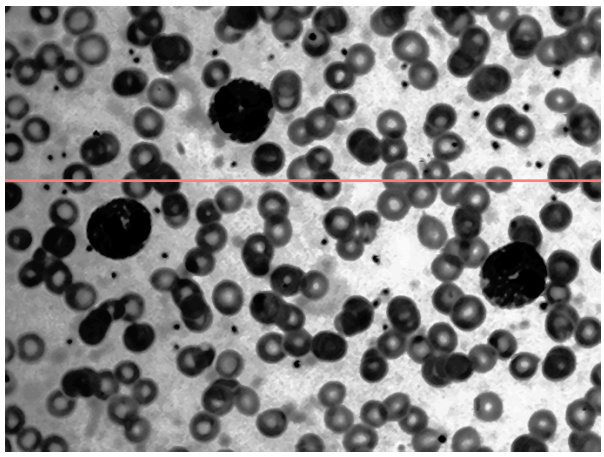


Fig. 1: `bloodcells` input image

For this, create a Java program that receives 2 command-line arguments: (1) the input image, and (2) the scanline number (i.e. row number). Use the template Java files for it and add your implementation to `computeProfile(BufferedImage input_img, int linenum)`. Test your implementation via Themis. The example output for the given scanline (plotted as graph) could look as in the following image:



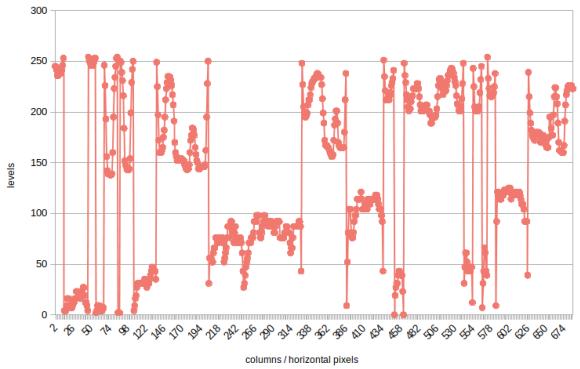


Fig. 2: Illustration of a scanline in the `bloodcells` image, and its corresponding profile.

Exercise 2 - Image Histogram (2 Pts.)

Write a function `computeHistogram(BufferedImage input_img)` that computes the histogram of a grayscale image like `bloodcells.tif` . For this create a Java program that receives 1 command-line argument: the input image and returns a `long[]` array. Use the template Java files and test your output via Themis. Make sure your function also works for other (greyscale) images, like `trui.tif`. The results could look like the example below:

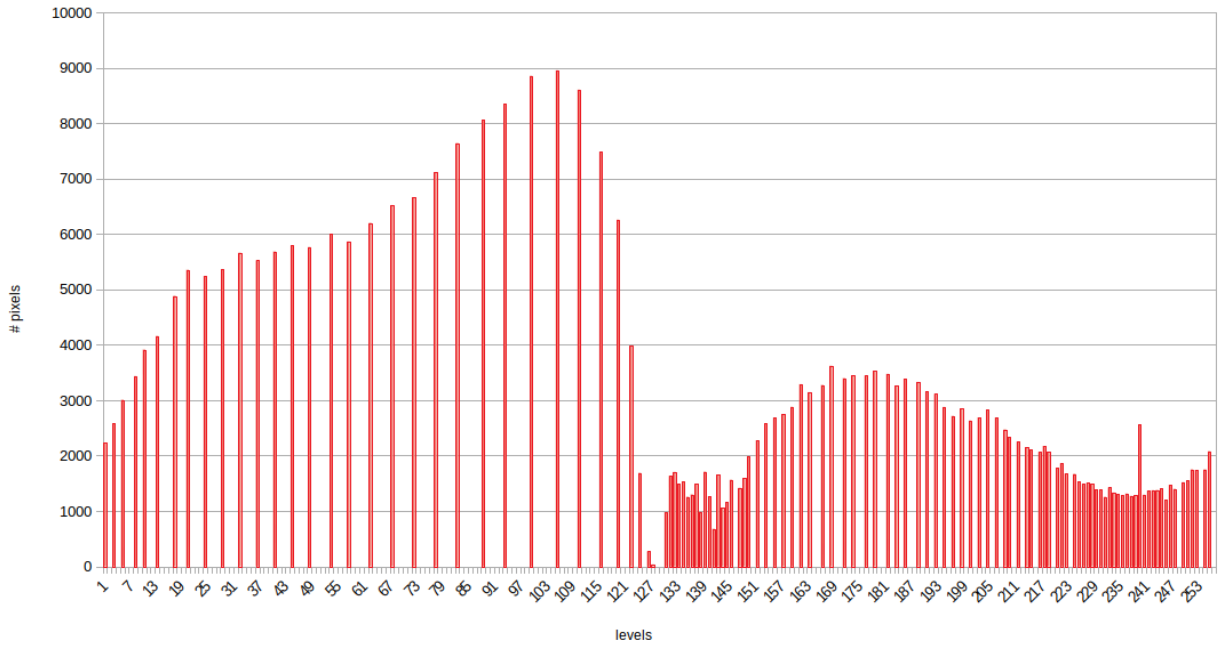


Fig. 3 Illustration of the histogram of the `bloodcells` image (Fig.1)

Exercise 3 - Image Entropy (2 Pts.)

Write a function `computeEntropy(BufferedImage input_img)` that computes the Entropy of a given greyscale input image. For this create a Java program that takes one command-line argument: the input image and returns the Entropy as a `float`. Use the template Java files and test your output via Themis.

Exercise 4 - Image Self-Information (2 Pts.)

Write a function `computeInformation(BufferedImage input_img)` that computes the self-information of a given greyscale input image. Here, compute the self-information for each data event *x*, with *x* representing the individual image intensities, and print out the related self-information to the console. Intensity values that do not occur in the image will be set to an information value of 0 (**definition** to maintain the fixed-length array output). For this, create a Java program that takes one command-line argument: the input image and returns the array of self-information as a sequence of `float`. Use the template Java files and test your output via Themis.

Exercise 5 - Noise Deterioration - Salt-and-Pepper Noise (4 Pts.)

Write a function `add_salt_n_pepper_simple(BufferedImage input_img, double noise_ratio)` that adds “salt and pepper” noise to a given grayscale input image. For this create a Java program that takes two command-line arguments: the input image and noise ratio and returns the updated image. Use the template Java files. For testing on Themis make sure to leave the given random seed untouched. Below you can see an example output.

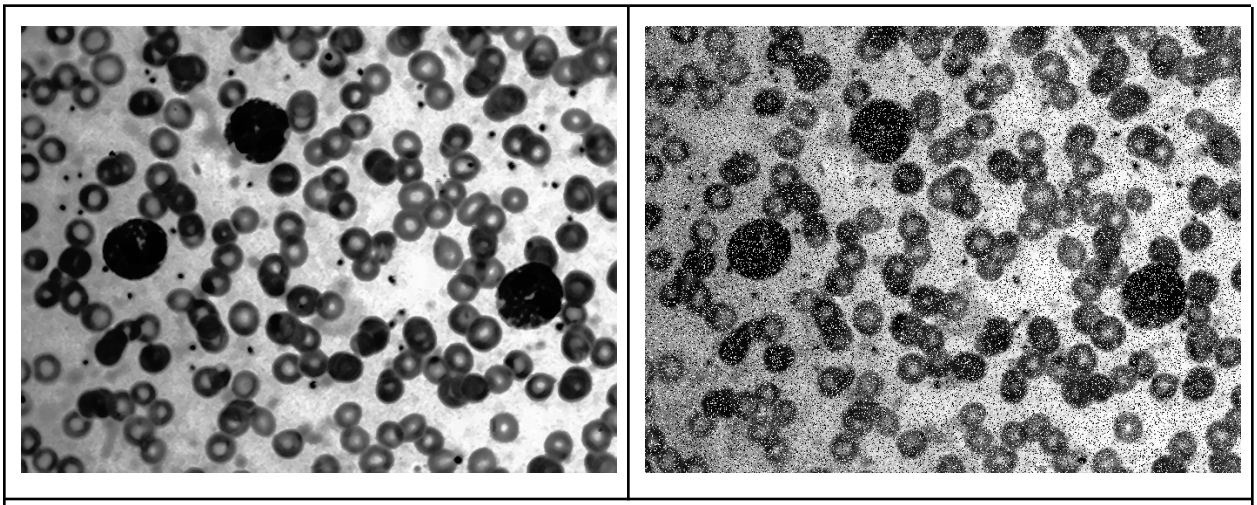


Fig. 4: Illustration of added salt and pepper noise in the `bloodcells` image.

Exercise 6 - Joint Histogram (6 Pts.)

Write a function `computeJointHistogram(BufferedImage input_img, BufferedImage ref_img)` that computes and returns a joint histogram between two given grayscale input images as 2D array `long[k][l]`. Here, the `k`-dimension represents the 1st image's histogram dimension, while the `l`-dimension stands for the 2nd image's histogram dimension. In order to accomplish this, create a Java program that takes two command-line arguments: the first- and second image's file paths. Use the template Java files and extend the above-mentioned function with your code. Test your output on Themis with the provided rocket-image and its distorted counterpart. On your own, you can also test other grayscale images.
For reasons of simplicity, we constrain ourselves to images of exact-same extents. This means, your code only needs to work for the smallest width-height subset shared between both images. Below, you see an illustration of the plotted output histogram.

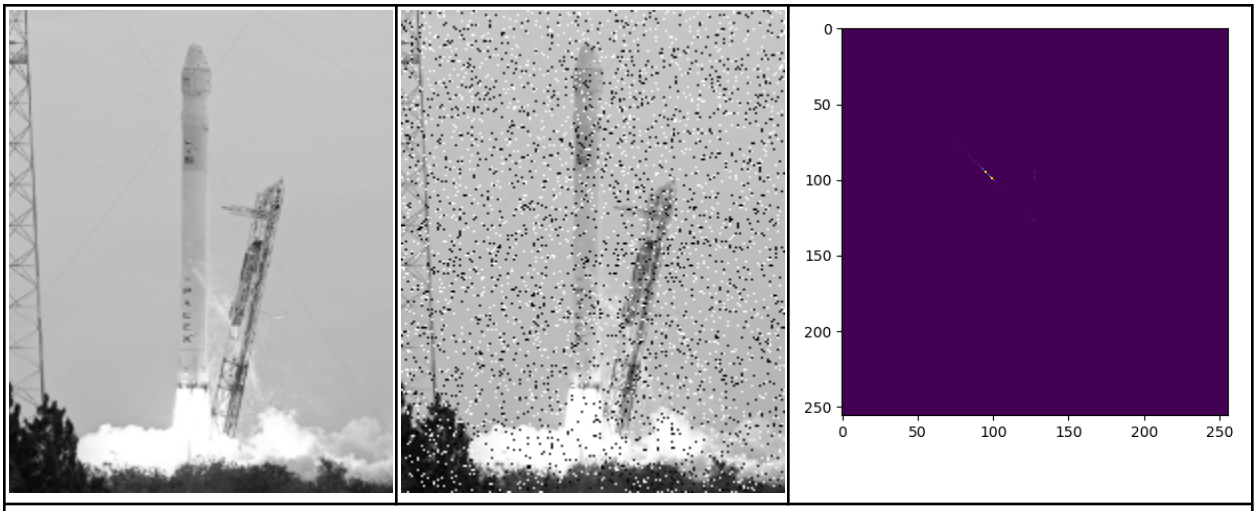


Fig. 5: Illustration of the resulting joint histogram computed between the `input` rocket image and its distorted `reference` counterpart.

Exercise 7 - Contrast Stretching (1 Pt.)

Write a function `stretchContrast(BufferedImage input_img)` that stretches the contrast of a given greyscale input image, returning the stretched image. For this create a Java program that takes one command-line argument: the input image. It outputs the stretched image. Use the template Java files and test your output via Themis.

Fig. 6: Illustration of contrast stretching on the `trui` image.

Exercise 8 - Gamma Correction (2 Pts.)

Write a function `gammaCorrection(BufferedImage input_img, float gamma)` that stretches contrast and brightness of a given greyscale input image with the unified γ parameter. For this, create a Java program that takes two command-line arguments: the input image and (optionally) a `float` gamma-value. The program returns the value-stretched image. Use the template Java files and test your output via Themis.

Fig. 7: Illustration of gamma-correction on the `trui` image. Here, γ=0.5.

Exercise 9 - (Global) Histogram Equalization (6 Pts.)

Write a function `equalizeHistogram(BufferedImage input_img)` that computes the histogram and equalizes the value distribution of a given greyscale input image according to the histogram. For this create a Java program that takes one command-line argument: the input image. The program returns the equalized image. Use the template Java files and test your output via Themis.

Fig. 8: Illustration of contrast stretching on the `trui` image.

Exercise 10 - Nearest-Neighbour Interpolation (2 Pts.)

Write a function `nearestInterpolate(BufferedImage input_img, float x, float y)` that interpolates the values of a given greyscale input image at a given position. For this create a Java program that takes three command-line arguments: the input image, and a floating-point x- and y-coordinate. The program returns the interpolated value as a `float`. Use the template Java files and test your output via Themis. Test your program with coordinates `(100.25, 59.75)`, which should return `138.0`. Use the `trui`-image as example input.

Exercise 11 - Bilinear Interpolation (2 Pts.)

Write a function `linearInterpolate(BufferedImage input_img, float x, float y)` that interpolates the values of a given greyscale input image at a given position. For this create a Java program that takes three command-line arguments: the input image, and a

floating-point x- and y-coordinate. The program returns the interpolated value as a `float`. Use the template Java files and test your output via Themis. Test your program with coordinates `(100.25, 59.75)`, which should return `143.1875`. Use the `trui`-image as example input.

Exercise 12 - RGB-to-HSL forward/backward transform (4 Pts.)

Write two functions – `rgb2hsl(int r, int g, int b)` and `hsl2rgb(float H, float S, float L)` – that compute the forward- and backward transformation of RGB-to-HSL for colour values of a given RGB input image.. For this, create a Java program that takes three command-line arguments: the input RGB colour image, and (optionally) the `integer`-valued x- and y-coordinates. The program returns the 3-value `float` triplet of the HSL values. Use the template Java files and test your output via Themis.
Locally test the forward transformation with the predefined coordinates, which should return the HSL triplet `(205.385, 96.296, 68.235)`.

Fig. 10: Example image provided for testing. The given test coordinate here is marked as a red dot.

Exercise 13 - RGB-to-YCbCr image split (ITU-R BT.601 profile) **(3 Pts.)**

Write a class containing the function `split_rgb2ycbcr()`. The class is envisaged to hold a given colour input image and store the resulting three pseudo-colour images. The class further implements the helping forward- and backward transform of RGB-to-YCbCr. The actual transformation can be done through the method explained in the lecture, or by employing the G-B and G-R deltas for generating Cb and Cr, which you can find online (e.g. Wikipedia). Either way, remember to use the actual ITU-R BT.601 profile for the transformation matrix (as shown in the lecture). For obtaining the individual pseudo-colour images, split the input in its Y, Cb and Cr components, then create three new pixels or images and store for each an image with its dedicated component being set from the input while neutral-setting the other colour components. Then, backward-transform the individual images for each channel to RGB for later storage. Think individually about the correct neutral-values judging from the lecture and the literature.
Create a Java program that takes one command-line argument, which is the input colour image. The program outputs the resulting pseudo-colour images. Use the template Java files and test your output via Themis.

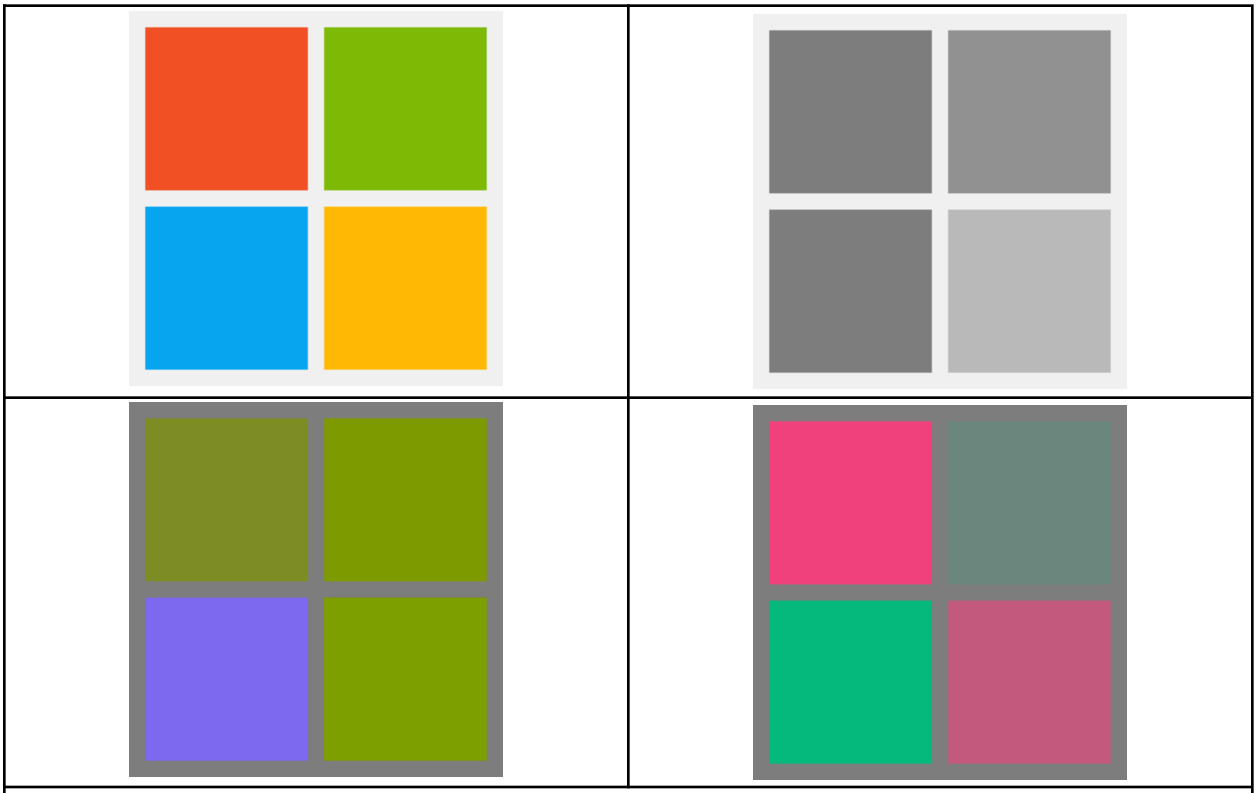


Fig. 11: Illustration of YCbCr pseudo-colouring on the replicated Microsoft logo-like image. top-left: original; top-right: Y (luma); bottom-left: Cb; bottom-right: Cr.