# Homework 5:

## Operations with Images (100 points)

Please read important information first.

## Description

In this homework, you will write a Python program to make some calculations on 3D pixels of some images.
We will use the **ppm** image format for reading image files. You are provided with two functions:

1. `read_ppm_file(f)` which (is a function you are familiar with from the lectures) will enable you to read .ppm files to a 3d list.
2. `img_printer(img)` which is a function that prints 3d lists in a readable manner. You will use this and only this to output your results.

Two of the inputs you need are taken for you, `filename` and `operation`.
filename is the name of the .ppm file you will read, and operation is a specific operation which you will apply to the image you read.
The rest of the inputs you take will depend on the operation. filename will always belong to files in src folder, so you can directly use them with the filename variable.
operation will always be an integer between 1-7, both inclusive. The details of the operations are as follows:

**operation == 1**

When operation input is 1, you will apply min-max normalization.
In this case you will require two additional inputs: `minimum` and `maximum`, which you will also get from the user once this option is chosen.
Each input will be given in separate lines. You can assume that these two values are integers.
The formula for min-max normalization is as follows:

Make sure to round your normalized result to 4 decimal places.

**operation == 2**

When operation input is 2, you will apply z-score normalization.
This case does not require additional inputs. You will find **channelwise** means and standard deviations, and output the normalized image.
The required formulas are provided below:

There are two additional details:

1. Add the number `1e-6` to your standard deviation result to make sure that if there is an input which makes the standard deviation equal to 0, you don't get an error.
2. Round your normalized result to 4 decimal places.

**operation == 3**

When operation input is 3, you will convert the image to black and white. This case does not require additional inputs.
This can easily be done by taking the average of each pixel's channel values, and assigning to each channel the average value.
For example, if a pixel's channel values are [12, 13, 14], you can do (12 + 13 + 14) / 3 = 13 and assign to that pixel's channels [13, 13, 13] and that's it.

**operation == 4**

When operation input is 4, you will apply convolution to the image.
You will be provided two additional inputs: one is the filename of a filter which will be located under the src folder and the other is a stride parameter which denotes how many steps the filter will move after each summation is complete.
**Important:** If the weighted sum exceeds the image maximum color value or becomes less than 0, you must clip it such that it is always between 0 and maximum color value.
Each input will be given in separate lines. An example of the operation is provided below where stride is 1:

**operation == 5**

When operation input is 5, you will again apply convolution to the image, but this time you
will pad zeros to the edges of your input image so that your output image has the same dimensions as your input image.
You will once more be provided two additional inputs: one is the filename of a filter which will be located under the src folder
and the other is a stride parameter which denotes how many steps the filter will move after each summation is complete.
**Important:** If the weighted sum exceeds the image maximum color value or becomes less than 0, you must clip it such that it is always between 0 and maximum color value.
Each input will be given in separate lines.

**operation == 6**

When operation input is 6, you will apply color quantization to the image.
You will get a range input which you will use to compare whether two **pixels** are similar enough to be grouped together or not. Input will be given in a separate line.
The rule for this quantization is simple, if **all** the channel values between two pixels differ by less than the range, then they will be made equal.
**Important:** This quantization part is **required** to be a recursive implementation.
You will start from the pixel at (0,0) coordinates and check each of its valid neighbors.

**operation == 7**

When operation input is 7, you will again apply color quantization to the image but this time it is a 3d quantization.
You will get a range input which you will use to compare whether two **channel values** are similar enough to be grouped together or not.
The rule for this quantization is if the channel values between two neighbours differ by less than the range, then they will be made equal.
Note that this time, different channels of the same pixel are considered as neighbors as well as the same channel on different pixels. **Important:** This quantization part is **required** to be a recursive implementation.
You will start from the channel at (0,0,0) coordinates and check each of its valid neighbors.

Check the examples for further clarification. Keep in mind that we will be grading your code not just based on these examples, but other cases as well, so try to write code which can handle all possible cases.

**Warning:** You are not allowed to use any imports and any topics that haven't been covered this semester.

You will be provided example operation input and outputs on Moodle. src folder contains certain input files you will need. If you alter these files by accident, you can also find the original versions on Moodle.
The recursion depth limit is assumed to be 1000, and inputs are provided accordingly.