Solution on Homework 1

In this assignment, we were tasked with converting a RAW image to a JPG/PNG format using a six-step process.

In the initial step, we converted the values of the raw image, originally represented as 16-bit unsigned integer array of pixel arranged in the bayer pattern, into floats ranging from 0 to 1. This involved saving the array as floats and then normalizing it by dividing it with (2^16)-1. (max. positive value of uint16)

Following that, the array was padded using the numpy pad method to later be able to compute the edge values of each channel.

Moving on to step two, we demosaiced the mosaiced image. Starting with the green channel, we initially made a copy of the still mosaiced image. Then, we iterated through the rows and columns, considering every second pixel in a step 2 manner. We had to differentiate between even and odd rows, but the main task was to calculate the missing values with the bilinear interpolation as discussed in the lecture. (Average Value of surrounding pixels). The calculation process is best observed in the code.

Next, we tackled the red channel. Again, starting with a copy of the mosaiced image, we computed the missing values in the columns (every second pixel in the row) since the given red pixels were not arranged in a checkerboard pattern. Then, we used these values to compute the missing values in the rows (every pixel in the row).

Similar operations were performed for the blue channel, but with an offset of one row and column.

After each channel was computed we had to cut off the padded edges we added before, so the channels had the same WxH as the input image.

The first function was considered complete after returning the image as a 3-D image using the numpy dstack method.

In the subsequent step, we applied white balancing. Initially, we split the provided 3-D image into its three channels. We then multiplied each channel by 0.5 divided by the mean of the channel. This operation ist just a rearrangement of the given formula, ensuring that each channel's mean became 0.5. Suppose the scalar is denoted as y; then, y * mean(channel) = 0.5. Therefore, y = 0.5/mean(channel). After applying this adjustment to each channel, we once again returned the image as a 3-D Array using the numpy dstack method.

In Step 4 we are simply applying the gamma curve using the np.power()-function on the image (it applies the power to each pixel in the image array).

In Step 5 we finally are clipping the image (in range 0 to 1), scale it with 255, and save it as a unit8 format and return it.

Saving the image was straightforward, using the imageio.imwrite() function, much like in HW0.

Note the comments in the code and the code itself for further clarification.