COMPENG 3SK3 – Project 3: Color Demosaicing for Digital Cameras with Linear Regression

Professor: Dr. Xiaolin Wu

Khaled Hassan, hassak9, 400203796

Due: April 8, 2023

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by Khaled Hassan (hassak9, 400203796).

Pseudocode

The pseudocode used to implement this project can be found below:

1. Read training and testing image, instantiate relevant variables and clear workspace and command window.
2. Conduct training stage by generating the coefficient matrices. This is done using the training image.
    a. Split the image into R, G and B channels.
    b. Pad channels to eliminate edge distortions.
    c. calculate coefficient matrices using each color filter array (CFA) output
3. Conduct testing stage by reading in a Bayer Image (instructor's test files), or by converting the testing image into a Bayer Image.
4. Run the linear regression algorithm using the coefficient matrices from the training stage.
    a. Reconstruct RGB values by multiplying each Bayer pattern by the appropriate coefficient matrix.
    b. Using a nested for loop to iterate over each pixel, then
        i. if red pixel, use GRBG.
        ii. else if green pixel, use BGGR.
        iii. else if blue pixel, use RGGB or GBRG.
5. Establish benchmark with the built-in MATLAB function *demosaic()* and compare images and RMSE results.

Code

The script is included here and submitted alongside the report and video in the file titled '3SK3_P3_hassak9_main.m'.

```
clc; clear;
%% Image Inputs & Paramaters %%
training_image = double(imread("training.jpg"));

% Uncomment out 1 @ a time (1/4)
%%% Non-Bayer Input
% testing_image = im2double(imread("testing_image_1.jpg"));
% testing_image = im2double(imread("testing_image_2.jpg"));
% testing_image = im2double(imread("testing_image_3.jpg"));

%%% Bayer Image Input:
% testing_image = im2double(imread("test1.png"));
% testing_image = im2double(imread("test2.png"));

% note: im2double is better when the image will be displayed. TODO undo
% normalization to [0, 1]

patch_size = [1 1];
window_size = [5 5];
pad_size = [2 2];


%% Step 1: Generate Coefficient Matrices %%
```

```
% Extract red, green and blue channels, pad to acommodate for processing
% image edges
red = training_image(:,:,1);
red_col = im2col(red, patch_size);
red_pad = padarray(red, pad_size, 'symmetric', 'both');
red_win = im2col(red_pad, window_size);

green = training_image(:,:,2);
green_col = im2col(green, patch_size);
green_pad = padarray(green, pad_size, 'symmetric', 'both');
green_win = im2col(green_pad, window_size);

blue= training_image(:,:,3);
blue_col = im2col(blue, patch_size);
blue_pad = padarray(blue, pad_size, 'symmetric', 'both');
blue_win = im2col(blue_pad, window_size);

% acquire coefficient matrices, 2 for each possible type of mosaic patch
[green_A_RGGB, blue_A_RGGB] = RGGB(red_win, green_win, blue_win,
green_col,blue_col);
[green_A_BGGR,red_A_BGGR] = BGGR(red_win, green_win, blue_win, green_col,
red_col);
[red_A_GRBG, blue_A_GRBG] = GRBG(red_win, green_win, blue_win,
red_col,blue_col);
[blue_A_GBRG, red_A_GBRG] = GBRG(red_win, green_win, blue_win,
blue_col,red_col);

%% Step 2: Testing
[x, y, z] = size(testing_image);

% Uncomment out 1 @ a time (2/4)
%%% Non-Bayer Input
% For RGGB Images:
% bayer_image = testing_image(:,:,2);
% bayer_image(1:2:x, 1:2:y) = testing_image(1:2:x, 1:2:y, 1);
% bayer_image(2:2:x, 2:2:y) = testing_image(2:2:x, 2:2:y, 3);
% greyscale = bayer_image;
% imwrite(bayer_image, "results/bayer_image.jpg")

%%% Bayer Image Input:
bayer_image = testing_image;
greyscale = testing_image;

if (rem(x, 2) ~= 0)% need to pad 1 row of size 510
    bayer_image(end+1, :) = zeros(1, 510);
end


%% Benchmarking
% Uncomment out 1 @ a time (3/4)
%%% Non-Bayer Input
% builtin_demosaic = demosaic(imread("results/bayer_image.jpg"), "rggb");
% RMSE_BI = get_RMSE(im2uint8(builtin_demosaic), im2uint8(testing_image));

%%% Bayer Image Input:
builtin_demosaic = demosaic(imread("test1.png"), "rggb");
% builtin_demosaic = demosaic(imread("test2.png"), "rggb");
```

```
imwrite(builtin_demosaic, "results/builtin_demosaic.jpg");

%% Step 3: Linear Regression
% reconstruct channels from Bayer Image
recon_r = repmat([1 0; 0 0], ceil(x/2), ceil(y/2)) .* bayer_image;
recon_g = repmat([0 1; 1 0], ceil(x/2), ceil(y/2)) .* bayer_image;
recon_b = repmat([0 0; 0 1], ceil(x/2), ceil(y/2)) .* bayer_image;

bayer_image = padarray(bayer_image, [3 3], 'symmetric', 'both');
bayer_image(end - 2, :) = [];
bayer_image(:, end - 2) = [];
bayer_image(3, :) = [];
bayer_image(:, 3) = [];

for i = 3:x + 2
    for j = 3:y + 2
        % extract and flatten a 5x5 submatrix of greyscale bayer_image
        flatten = bayer_image(i - 2:i + 2, j - 2:j + 2);
        flatten = flatten(:);

        if mod(i, 2) && ~mod(j, 2) % red, use GRBG to reconstruct
            recon_r(i - 2, j - 2) = red_A_GRBG'*flatten;
            recon_b(i - 2, j - 2) = blue_A_GRBG'*flatten;

        elseif ~mod(i, 2) && ~mod(j, 2) % green, use BGGR to reconstruct
            recon_g(i - 2, j - 2) = green_A_BGGR'*flatten;
            recon_r(i - 2, j - 2) = red_A_BGGR'*flatten;

        elseif mod(i, 2) && mod(j, 2) % blue, use RGGB to reconstruct
            recon_b(i - 2, j - 2) = blue_A_RGGB'*flatten;
            recon_g(i - 2, j - 2) = green_A_RGGB'*flatten;

        elseif ~mod(i, 2) && mod(j, 2) % blue, use GBRG to reconstruct
            recon_b(i - 2, j - 2) = blue_A_GBRG'*flatten;
            recon_r(i - 2, j - 2) = red_A_GBRG'*flatten;
        end
    end
end

reconstructed_image = cat(3, recon_r, recon_g, recon_b);
reconstructed_image = uint8(reconstructed_image.*255); % undoing
normalization
imwrite(reconstructed_image,
"results/linear_regression_reconstruction.jpg");

% Uncomment out 1 @ a time (4/4)
%%% Display RMSE Results %%%
%%% Non-Bayer Input
% RMSE_LR = get_RMSE(im2uint8(reconstructed_image),
im2uint8(testing_image));
% fprintf("RMSE b/w testing image and reconstructed: %.10f\n", RMSE_LR);
% fprintf("RMSE b/w testinag image and builtin: %.10f\n", RMSE_BI);

% subplot to view results
figure;
subplot(1,3,1);
```

```
imshow(greyscale);
title('Bayer Image');
subplot(1,3,2);
imshow(reconstructed_image);
title('Reconstructed using Linear Regression');
subplot(1,3,3);
imshow(builtin_demosaic);
title('Built In');


%% Utility Functions %%

function RMSE = get_RMSE(image_1, image_2)
    RMSE = sqrt(immse(image_1, image_2));
end

function [green_A_RGGB, blue_A_RGGB] =
RGGB(red_win,green_win,blue_win,green_col,blue_col)

    red_pat = repmat ([1 0;0 0], [3,3]);
    green_pat = repmat ([0 1;1 0], [3,3]);
    blue_pat = repmat ([0 0;0 1], [3,3]);

    red_pat(6,:) = [];
    red_pat(:,6) = [];

    green_pat(6,:) = [];
    green_pat(:,6) = [];

    blue_pat(6,:) = [];
    blue_pat(:,6) = [];

    X = (im2col(red_pat,[1 1])'.*red_win+im2col(green_pat,[1
1])'.*green_win+im2col(blue_pat,[1 1])'.*blue_win)';

    blue_A_RGGB = inv(X'*X)*X'*(blue_col');
    green_A_RGGB = inv(X'*X)*X'*(green_col');
end

function [green_A_BGGR,red_A_BGGR] =
BGGR(red_win,green_win,blue_win,green_col,red_col)

    blue_pat = repmat ([1 0;0 0], [3,3]);
    green_pat = repmat ([0 1;1 0], [3,3]);
    red_pat = repmat ([0 0;0 1], [3,3]);

    red_pat(6,:) = [];
    red_pat(:,6) = [];

    green_pat(6,:) = [];
    green_pat(:,6) = [];

    blue_pat(6,:) = [];
    blue_pat(:,6) = [];
```

```
    X =
(im2col(red_pat,[1,1])'.*red_win+im2col(green_pat,[1,1])'.*green_win+im2col(
blue_pat,[1,1])'.*blue_win)';

    red_A_BGGR = inv(X'*X)*X'*(red_col');
    green_A_BGGR = inv(X'*X)*X'*(green_col');
end

function [red_A_GRBG,blue_A_GRBG] =
GRBG(red_win,green_win,blue_win,red_col,blue_col)

    green_pat = repmat ([1 0;0 1], [3,3]);
    blue_pat= repmat ([0 0;1 0], [3,3]);
    red_pat= repmat ([0 1;0 0], [3,3]);

    red_pat(6,:) = [];
    red_pat(:,6) = [];

    green_pat(6,:) = [];
    green_pat(:,6) = [];

    blue_pat(6,:) = [];
    blue_pat(:,6) = [];

    X =
(im2col(red_pat,[1,1])'.*red_win+im2col(green_pat,[1,1])'.*green_win+im2col(
blue_pat,[1,1])'.*blue_win)';

    blue_A_GRBG = inv(X'*X)*X'*(blue_col');
    red_A_GRBG = inv(X'*X)*X'*(red_col');
end

function [blue_A_GBRG,red_A_GBRG] =
GBRG(red_win,green_win,blue_win,blue_col,red_col)

    green_pat= repmat ([1 0;0 1], [3,3]);
    blue_pat= repmat ([0 1;0 0], [3,3]);
    red_pat= repmat ([0 0;1 0], [3,3]);

    red_pat(6,:) = [];
    red_pat(:,6) = [];

    green_pat(6,:) = [];
    green_pat(:,6) = [];

    blue_pat(6,:) = [];
    blue_pat(:,6) = [];

    X = (im2col(red_pat,[1 1])'.*red_win+im2col(green_pat,[1
1])'.*green_win+im2col(blue_pat,[1 1])'.*blue_win)';

    red_A_GBRG = inv(X'*X)*X'*(red_col');
    blue_A_GBRG = inv(X'*X)*X'*(blue_col');
end
```

The MATLAB script above performs image demosaicing, which involves converting raw sensor data from the format a digital camera would store a digital image into a full-color image. The script takes as input a training image and a testing image. The testing image can either be a non-Bayer pattern image (full RGB color) a Bayer pattern image (raw greyscale mosaic). The script requires modifications in four separate line blocks depending on which category the input image falls under. Then, the image demosaicing process follows goes through 2 main steps to reconstruct the full-color image:

1. Generate Coefficient Matrices:

This step constitutes the algorithm training phase of the image reconstruction process. First, the red, green, and blue channels are extracted from the training image and pads them to accommodate for image edge processing, since the individual pixel reconstruction process works in 5 x 5 blocks. The padded matrices are then inverted into columns to form windows, which are then used to generate the coefficient matrices for each of the 4 possible types of mosaic patch (RGGB, BGGR, GRBG, and GBRG). At this stage, the model has recognized the necessary patterns from the training data, which would then be used to reconstruct the testing image in the subsequent stages.

2. Apply Learned Pattern on Testing image:

If the testing image is a non-Bayer, fully RGB image, the script needs to isolate the Bayer image representation from that input to use in the reconstruction process. As the input image follows the RGGB pattern, the green channel is isolated from the image, and by indexing and slicing the input image, the Bayer image representation is extracted from the non-Bayer input. The next major step of the process involving the Linear Regression algorithm requires a Bayer pattern image, and so this step is skipped if that is the case. The next step involves applying the pattern that was trained on each pixel. Therefore, a nested for loop was used to iterate over every pixel in every row and every column. To accommodate for the padding that is introduced to the testing image, we iterate from the 3rd index until 2 indices after the last real pixel. This is because of the aforementioned nature of the reconstruction process, and how it requires operating on no smaller than a 5x5 submatrix of pixel data blocks. After padding and extracting, this submatrix is flattened, and depending on the type of the mosaic patch, the appropriate coefficient matrix is multiplied by the flattened submatrix. Finally, the red, green and blue channels are fully reconstructed at the end of the nested for loop. The last step in the reconstruction process is to concatenate the channels together before writing to an image filetype. The root mean square error (RMSE) is also calculated between the input testing image and the reconstructed image. For benchmarking purposed, the built-in MATLAB *demosaic()* function is also used and the RMSE of using it is also documented.

Experimental Results & Discussion

The script was run 5 times, including 3 of my own test cases in addition to the 2 test cases provided by the instructor. The image seen in Figure 1 was used to train the model. This image was chosen as it possessed a wide variety of different colors with plenty of sharply contrasting edges between them. As such, this should train the model fairly rigorously.

Figure 1: Training Image Used [1]

Links to 3 demonstration videos can be found below:

1. Part 1: https://drive.google.com/file/d/1OV9zp6o7iiamJeH7pQn9dZE9Mac9VGW5/view?usp=share_link
2. Part 2: https://drive.google.com/file/d/1OTJ3Jq8faf_ys6J8z26DzjxHhKNGIc83/view?usp=share_link
3. Part 3: https://drive.google.com/file/d/1ORCpO3xk1la2DzNyoyasg82QKQsXuoyx/view?usp=share_link

Experimental results are summarized and discussed as follows:

My first test case: Motorcycle.

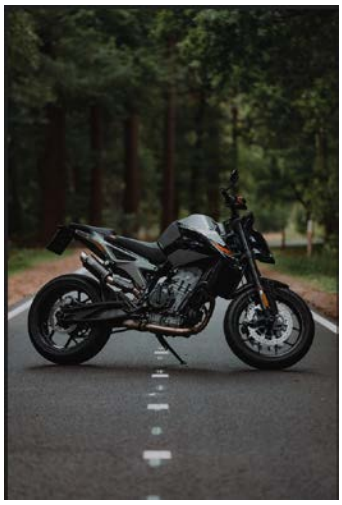

Figure 2: Original Testing Image [2]

Figure 3: Bayer Pattern Image

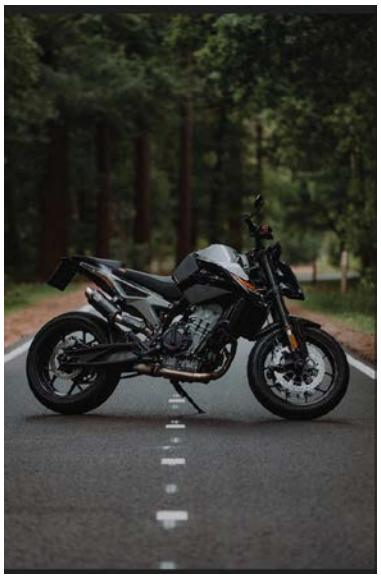

Figure 4: Linear Regression Reconstruction

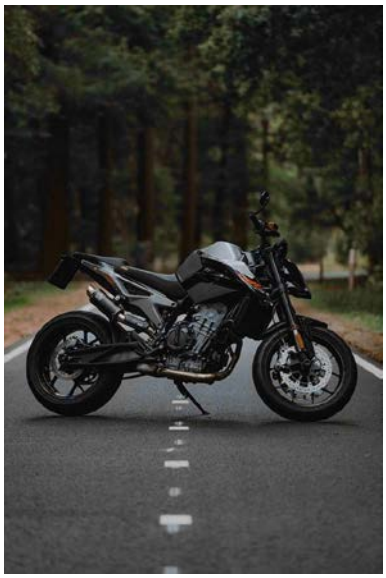

Figure 5: Built-in *demosaic()* Reconstruction

Figures 2 through 5 demonstrate the implementation of my color demosaicing script using an image found online and comparing the results with the built-in *demosaic()* function. The training image used can be seen in Figure 1 above. Figure 3 shows the Bayer Pattern representation of the image in Figure 2, which is then fed into the model to be reconstructed in Figure 4. The image looks extremely similar to the original, with very minor discrepancies noticeable when colors closely mix together. The experimental RMSE is found to be 2.982. In contrast, the RMSE of calling the built-in function, whose output can be observed in Figure 5, has been found to be 4.607. Therefore, it can be concluded that my implementation has outperformed the benchmark, fulfilling the bonus component of the project.

My second test case: Sunrise Image

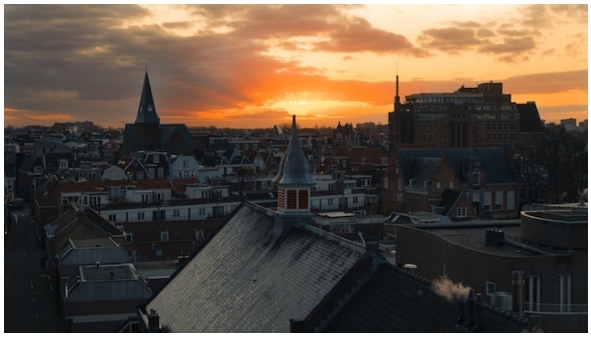

Figure 6: Original Testing Image [3]

Figure 7: Bayer Pattern Image

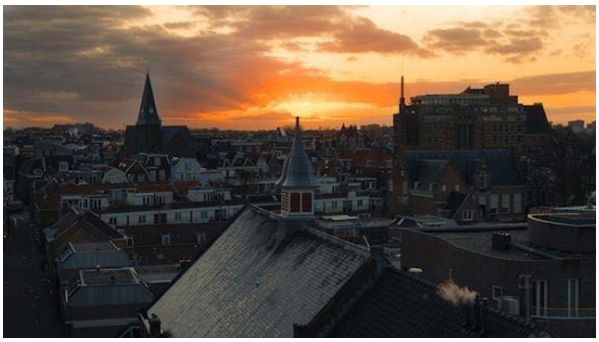

Figure 8: Linear Regression Reconstruction

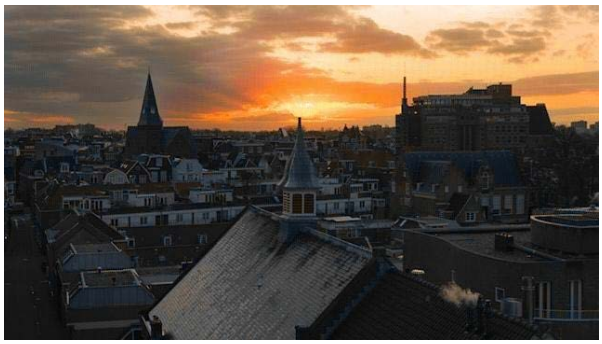

Figure 9: Built-in *demosaic()* Reconstruction

Figures 6 through 9 demonstrate the second test run of the script using another testing image found online. As with the first test case, the image outputted by the reconstruction model is identical to the input image. Once again, the RMSE resulting from running the model is less than the benchmark function, with the experimental RMSE being 2.922 compared to 4.561.

My third test case: Dirt Bikes

Figure 10: Original Testing Image

Figure 11: Bayer Pattern Image

Figure 12: Linear Regression Reconstruction

Figure 13: Built-in *demosaic()* Reconstruction

Figures 11 through 13 demonstrate the third test run of the script using another testing image found in the Project instructions Document. As with the first and second test cases, the image outputted by the reconstruction model is identical to the input image. The RMSE resulting from running the model is slightly higher, reaching 6.347 for this test case. However, the model still outperforms the benchmark, which results in an RMSE of 8.254.

Instructor's test case 1:

Figure 14: Bayer Pattern

Figure 15: LR Reconstruction

Figure 16: Built-in Reconstruction

Instructor's test case 2:

Figure 17: Bayer Pattern

Figure 18: LR Reconstruction

Figure 19: Built-in Reconstruction

For the next two test cases, the script was modified to allow for the use of the Bayer pattern input image provided by the instructor. As such, an RMSE comparison could not be generated for these test cases. An examination of the image quality should be sufficient to compare the script with MATLAB's benchmark function. As can be seen in Figures 14 to 16 and Figures 17 to 19 above, the reconstructed images are sufficiently clear, and a fairly close reconstruction to what the original images are predicted to be. This indicates the accuracy of the reconstruction model in general and compared to the MATLAB benchmark. Additionally, the RMSE values obtained above indicate the successful completion of the project bonus requirement.

References

[1] D. Chan, “assorted-color petaled flowers,” *Unsplash.com*, Jun. 09, 2015. https://unsplash.com/photos/pXmbsF70ulM (accessed Apr. 08, 2023).

[2] G. Coolen, “black and silver naked motorcycle on road during daytime,” *Unsplash.com*, Jun. 08, 2020. https://unsplash.com/photos/-5rcxih1e44 (accessed Apr. 08, 2023).

[3] M. Kleen, “white and brown concrete buildings during sunset,” *Unsplash.com*, Nov. 27, 2020. https://unsplash.com/photos/eRwmBvhfvG8 (accessed Apr. 08, 2023).