Harald Nordgren
900404-1936
ada09hno@student.lth.se

FMN011 – Project 2

I copied the `A` matrix from the `pdf` to a simple text editor for basic reformatting and then into `Matlab`. I saved it to file as `A.mat` with the command `save A`.

I then wrote a script for creating the grayscale models of assignment 1. This code works from a cleared session, given the file `.mat`-file containing matrix `A`:

```
load A;

FOLDER = 'A_matrix';
UNCOMPRESSED_PATH = [FOLDER, '/Uncompressed.jpg'];

FILE_NAME_START = 'A_matrix_grayscale_SVD_compression_rank_';
FIGURE_TITLE_START = 'Matrix A grayscale,';
FIGURE_TITLE_MIDDLE = 'SVD compression rank ';
CAPTION_START = 'Compression ratio ';

RANKS = 1:3;

figure;
set(gcf, 'Position', [0 0 634 638]);

ranks_size = size(RANKS);
subplot_width = (ranks_size(2) + 1) / 2;
subplot(2, subplot_width, 1)

intens(round(A), {FIGURE_TITLE_START, 'No compression'});
imwrite(divide_all_by_largest_element(A), UNCOMPRESSED_PATH);
uncompressed_size = file_size(UNCOMPRESSED_PATH);

%Space allocation
errors = RANKS;
size_ratios = RANKS;

for k = RANKS
    RANK_STRING = int2str(k);
    FULL_TITLE = {FIGURE_TITLE_START, [FIGURE_TITLE_MIDDLE, RANK_STRING]};

    FILE_NAME = create_image_path(FOLDER, [FILE_NAME_START, RANK_STRING]);
    [A_compressed, size_ratios(k), errors(k)] = ...
        compress_and_create_image(A, k, FILE_NAME, uncompressed_size);

    subplot(2, subplot_width, k+1);
    intens(round(A_compressed), FULL_TITLE);

    FULL_CAPTION = [CAPTION_START, num2str(size_ratios(k), 3)];
    text(0.15, -0.1, FULL_CAPTION, 'Units', 'normalized');
```

```
end

generate_table(FOLDER, RANKS, size_ratios, errors);
```

Variables in capital letters are used for numerical constants and strings. To achieve a logical and automatic naming of images I do lots of string manipulation throughout the code, starting here with the first few lines and further in the `for` loop.

I create a figure which will host the `2×2` subplots, and specify a size, which I extracted using the `get(gcf,'Position')` function after manually resizing the resultant figure to appropriate size. A similar process is later repeated for each of the assignments.

I plot the uncompressed image is in the top-left corner, corresponding to position `1`. This done by `intens`, which does simply what was specified in the project description, i.e. converts the matrix to a grayscale plot and adds a title specified as an input parameter.

```
function [] = intens(M, img_title)
imagesc(M);
axis off;
colormap(gray);
title(img_title);
```

I save the images with `imwrite` , which first required me to divide each element with the max value of the entire matrix. This is done by the function `divide_all_by_largest_element`.

```
function M_norm = divide_all_by_largest_element(M)
max_val = max(M(:));
M_norm = M / max_val;
```

In the loop I call `compress_and_create_image` for ranks 1:3,

```
function [A_compressed, ratio, error] = ...
    compress_and_create_image(A, k, FILE_NAME, uncompressed_size)

A_compressed = svd_compression(A, k);
imwrite(divide_all_by_largest_element(A_compressed), FILE_NAME);

ratio = compression_ratio(uncompressed_size, FILE_NAME);
error = matrix_relative_error(A, A_compressed);
```

where the actual compression is performed by `svd_compression`

```
function y = svd_compression(M, k)
[U,S,V] = svd(M);
y = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';
```

which I based on https://inst.eecs.berkeley.edu/~ee127a/book/login/l_svd_apps_image.html.

The compressed matrix is then saved as a image and the compression ratio is computed using the difference in file sizes between the uncompressed and compressed `.jpg` images. The error is calculated using `matrix_relative_error`.

```
function result = matrix_relative_error(A, A_compressed)
result = norm(A - A_compressed) / norm(A);
```

I then create a new subplot and plot the intensity matrix for the compressed matrix. After the loop is finished I put all the error data into a table using `generate_table`, which also saves it to file for easier importing to this report.

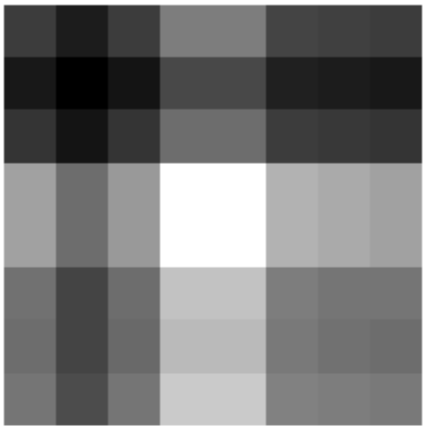
```
function [] = generate_table(FOLDER, RANKS, size_ratios, errors)

SIZE_RATIO_STRINGS = arrayfun(@(x) num2str(x, 3), ...
    size_ratios, 'uniformoutput',false);
ERROR_STRINGS = arrayfun(@(x) num2str(x, 2), ...
    errors, 'uniformoutput',false);

data_table = table(RANKS', SIZE_RATIO_STRINGS', ERROR_STRINGS', ...
    'VariableNames',{'SVD_compression_rank' ...
    'Compression_ratio' 'Relative_error'})

data_table_path = ['data_tables/', FOLDER, '_compression.csv'];
writetable(data_table, data_table_path);
```

The resulting `2×2` subplots are presented below, where it's clear that the compressed image resembles the original more when rank is increased. Ranks 2 and 3 are *passable* as representations of the uncompressed image, whereas the rank 1 image is compressed to the point where the important contrasts of the image are essentially gone. Maybe if used as a very small thumbnail it could be acceptable.

The compression ratio for each rank is very close to `1`, so nothing is really gained from compressing `8×8` pixel images like this, and especially not for ranks above `3`. Dividing every matrix element by the largest element before calling `imagesc` has no effect as far as I can tell, which seems to indicate the function automatically scales the input data.

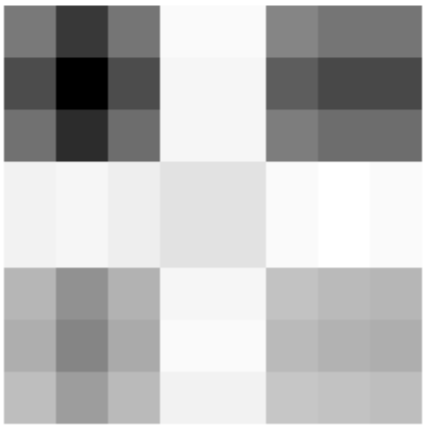
**Matrix A grayscale,
No compression**

**Matrix A grayscale,
SVD compression rank 1**



Compression ratio 1.05

**Matrix A grayscale,
SVD compression rank 2**



Compression ratio 1.02

**Matrix A grayscale,
SVD compression rank 3**

Compression ratio 1.01

For the the second assignment I started by downloading `nena.jpg` from the web page, and then wrote a script for the grayscale and color image compressions.

```
COLOR_UNCOMPRESSED_PATH = 'nena.jpg';

nena = imread(COLOR_UNCOMPRESSED_PATH);
file = dir(COLOR_UNCOMPRESSED_PATH);
nena_size = file.bytes;

[grayscale_ranks, grayscale_size_ratios, grayscale_errors] = ...
    assignment_2_grayscale(nena);
[color_ranks, color_size_ratios, color_errors] = ...
    assignment_2_color(nena, nena_size);
```

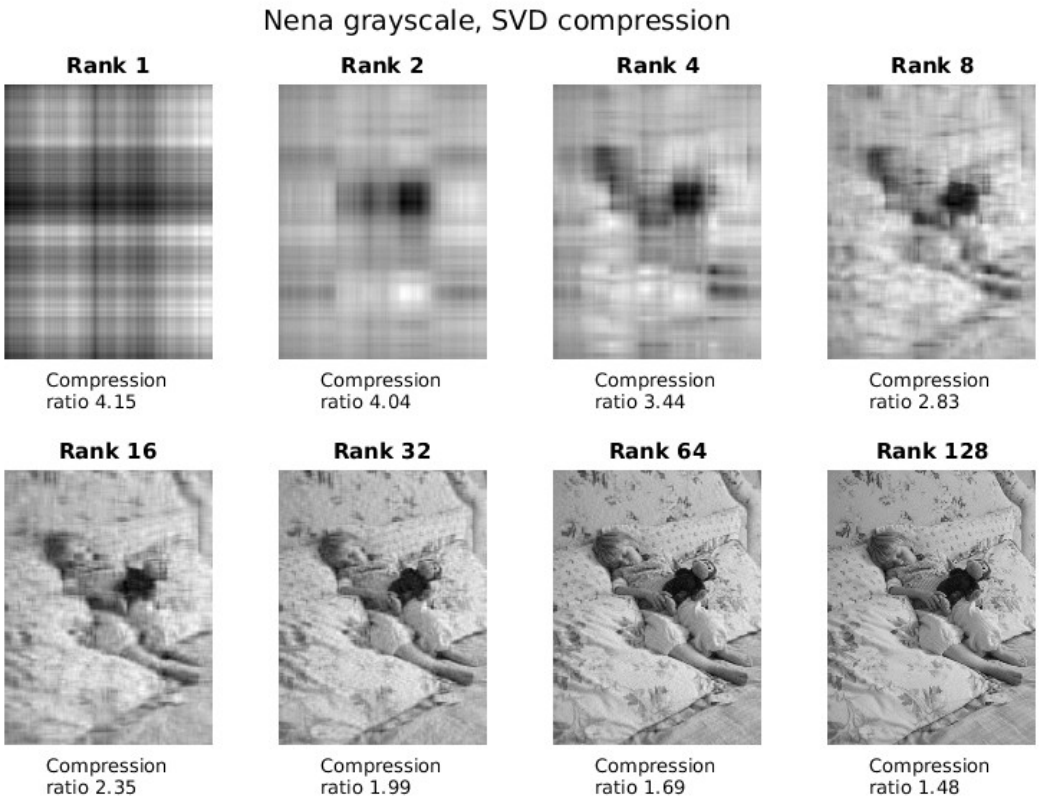
I calculate the file size for use in computing the compression ratios later on, and call functions `assignment_2_grayscale` and `assignment_2_color`.

```
function [RANKS, size_ratios, errors] = assignment_2_grayscale(nena)

FOLDER = 'nena_grayscale';
UNCOMPRESSED_IMAGE_PATH = [FOLDER, ...
    '/Nena_grayscale_uncompressed.jpg'];
FILE_NAME_START = 'Nena_grayscale_SVD_compressed_rank_';

FIGURE_TITLE = 'Nena grayscale, SVD compression';
SUBPLOT_TITLE_START = 'Rank ';
SUBPLOT_CAPTION_START = 'Compression';
SUBPLOT_CAPTION_MIDDLE = 'ratio ';

RANKS = 2 .^ (0:7);

nena_grayscale = double(rgb2gray(nena));
imwrite(divide_all_by_largest_element(nena_grayscale),
UNCOMPRESSED_IMAGE_PATH);
uncompressed_size = file_size(UNCOMPRESSED_IMAGE_PATH);

%Space allocation
rank_size = size(RANKS);
rank_indices = 1:rank_size(2);
size_ratios = RANKS;
errors = RANKS;

figure;
set(gcf, 'Position', [0 0 926 597]);

for k = rank_indices
    RANK_STRING = int2str(RANKS(k));
    FILE_NAME = create_image_path(FOLDER, strcat( ...
        FILE_NAME_START, RANK_STRING));
    FULL_TITLE = [SUBPLOT_TITLE_START, RANK_STRING];

    [compressed_image, size_ratios(k), errors(k)] = ...
        compress_and_create_image( nena_grayscale, ...
        RANKS(k), FILE_NAME, uncompressed_size);
```

```
    subplot(2, rank_size(2)/2, k);
    intens(compressed_image, FULL_TITLE);

    FULL_CAPTION = {SUBPLOT_CAPTION_START, ...
        [SUBPLOT_CAPTION_MIDDLE, num2str(size_ratios(k), 3)]};
    text(0.2, -0.12, FULL_CAPTION, 'Units', 'normalized');
end

suptitle(FIGURE_TITLE)

generate_table(FOLDER, RANKS, size_ratios, errors);
```

I found that exponential values for `RANKS` gives a good span and brings out the clear difference between different compression ranks. For each rank specified, `compress_and_create_image` is once again called to compress the matrix and save it as an image. Most of the other code resembles that of the first assignment.

For compression ranks below `2^4`, images are mainly noise and basically useless for any application. At around `2^5`, reasonable quality is achieved for the first time at a compression ratio close to `2`. Great quality is given at `2^8` but the compression ratio has now dipped. For even higher ranks, the compressed image looks very much like the original, but at the cost of any real decrease in image size.



The color image manipulation is performed by `assignment_2_color`.

```
function [RANKS, size_ratios, errors] = ...
    assignment_2_color(nena, nena_size)

FOLDER = 'nena_color';

FIGURE_TITLE = 'Nena color, SVD compression';
SUBPLOT_TITLE_START = 'Rank ';
SUBPLOT_CAPTION_START = 'Compression';
SUBPLOT_CAPTION_MIDDLE = 'ratio ';

RANKS = 2 .^ (0:7);

nena_double = double(nena);

%Space allocation
ranks_size = size(RANKS);
rank_indices = 1:ranks_size(2);
size_ratios = rank_indices;
errors = rank_indices;

loops = ranks_size(2);
F(loops) = struct('cdata',[],'colormap',[]);

figure;
set(gcf, 'Position', [0 0 926 597]);

for k = rank_indices
    [compressed_tensor, compressed_tensor_divided_by_largest, ...
        size_ratios(k), errors(k)] = compress_3_components(FOLDER, ...
        nena_double, nena_size, RANKS(k));

    movie_image = imresize(uint8(round(compressed_tensor)), 0.3);
    F(k) = im2frame(movie_image);

    subplot(2, ranks_size(2) / 2, k);
    imshow(imresize(compressed_tensor_divided_by_largest, 0.1));

    axis image;
    title([SUBPLOT_TITLE_START, int2str(RANKS(k))]);

    FULL_CAPTION = {SUBPLOT_CAPTION_START, ...
        [SUBPLOT_CAPTION_MIDDLE, num2str(size_ratios(k), 3)]};
    text(0.2, -0.12, FULL_CAPTION, 'Units', 'normalized');
end

suptitle(FIGURE_TITLE);
generate_table(FOLDER, RANKS, size_ratios, errors);

assignment_2_video(F, RANKS);
```

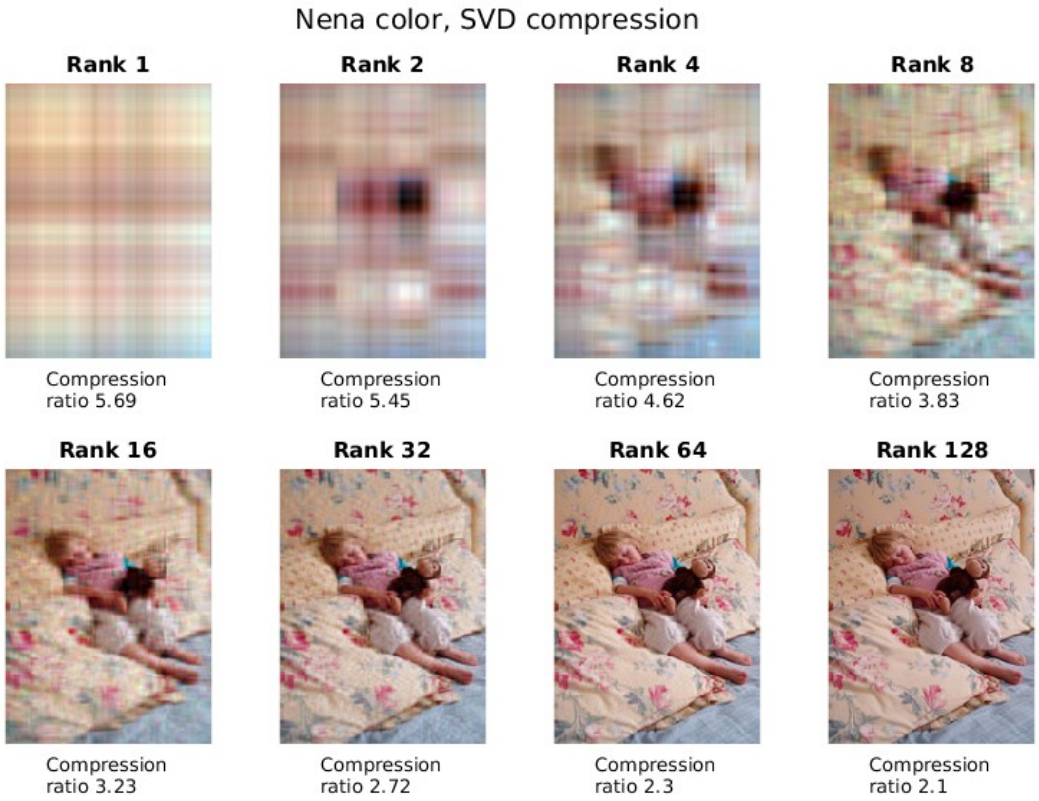
The compression of a color image is more complicated than for grayscale, and is handled by the function `compress_3_components` which uses `svd_compression` on each component

separately and then recombines the compressed components. After which the image is saved to file, and compression ratios and errors are returned. The errors are calculated as the mean between the errors of each individual component matrix.

```
function [compressed_image, compressed_image_divided_by_largest, ...
    size_ratio, error_median] = compress_3_components(FOLDER, ...
    uncompressed_image, uncompressed_size, rank)

COMPONENTS = 3;
FILE_NAME = '/nena_compressed_rank_';

%Space allocation
compressed_image = uncompressed_image;
compressed_image_divided_by_largest = uncompressed_image;
error_sum = 0;

for i = 1:COMPONENTS
    uncompressed_component = uncompressed_image(:,:,i);
    compressed_component = svd_compression(uncompressed_component, rank);
    compressed_image_divided_by_largest(:,:,i) = ...
        divide_all_by_largest_element(compressed_component);
    compressed_image(:,:,i) = compressed_component;

    error_sum = error_sum + matrix_relative_error( ...
        uncompressed_component, compressed_component);
end

error_median = error_sum / COMPONENTS;

file_name = create_image_path(FOLDER, strcat(FILE_NAME, int2str(rank)));
imwrite(compressed_image_divided_by_largest, file_name);

size_ratio = compression_ratio(uncompressed_size, file_name);
```

Looking at the table below, it appears that SVD gives better results for color images than for grayscale, with higher compression ratios at every rank and similar errors. Compression ratio `2` is now achived even at rank `2^7`, which has great quality even when zoomed in. A huge improvement over grayscale.

Otherwise, the relationship between ranks and quality matches that of the grayscale images pretty closely, a fact verified by the relative error of different compressed matrices. My naked-eye evaluation tells me that relative errors of `0.005` and below are generally to be preferred.



SVD compression rank	Nena grayscale realtive error	Nena color realtive error
1	0.14	0.14
2	0.099	0.10
4	0.057	0.061
8	0.036	0.038
16	0.024	0.024
32	0.013	0.014
64	0.0073	0.0077
128	0.0036	0.0038
256	0.0016	0.0017
512	0.00065	0.00069
1024	0.00016	0.00017

Finally, to create a video from compressed images the function `assignment_2_video` is called. I experimented with different values for `RANKS` and found that linear values of `1:3:100` gave the most interesting flow of images. The variable `F` is assigned a frame for every compressed color image created in `assignment_2_color`, and then passed to the video function along with `RANKS`.

```
function [] = assignment_2_video(F, RANKS)

FRAME_RATE = 6;

RANKS_FIRST = int2str(RANKS(1));
RANKS_DIFF = int2str(RANKS(2) - RANKS(1));
RANKS_LAST = int2str(RANKS(end));
RANKS_FOR_TITLE = [RANKS_FIRST, ':', RANKS_DIFF, ':', RANKS_LAST];
VIDEO_PATH = [VIDEO_FOLDER, TITLE_START, RANKS_FOR_TITLE, ...
    '_framerate_', int2str(FRAME_RATE), '.avi'];

myVideo = VideoWriter(VIDEO_PATH);
myVideo.FrameRate = FRAME_RATE;

open(myVideo);
writeVideo(myVideo, F);
close(myVideo);
```

From this project I learned huge amounts about Matlab, like how to transform data for your needs, how to manipulate strings in a useful manner and how to work with files and folders in a script. I learned how to write better Matlab functions, processing images and how to export data from Matlab, including into videos. I got better acquainted with how images are represented internally, which was quite eye-opening.

Working with some functions that only accept 8-bit integer matrices, and others that only accept numbers in the range [0,1] was frustrating at first. Same thing with the video functionality where the Matlab help pages are not always easy to read, although the Matlab Central forums are usually more accommodating.

Going through the painstaking process of finding the right function and syntax for each situation was still probably a worthwhile endeavor, and I already feel much more comfortable with using Matlab as a proper scripting language. It has tons of power.