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 gray-scale 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_';

CAPTION_START = 'Matrix A grayscale,';
CAPTION_END = 'SVD compression rank ';
CAPTION_NO_COMPRESSION = 'No compression';

RANKS = 1:3;

subplot(2,2,1)
intens(A, {CAPTION_START, CAPTION_NO_COMPRESSION});

imwrite(normalize_matrix(A), UNCOMPRESSED_PATH);
uncompressed_size = file_size(UNCOMPRESSED_PATH);

%Space allocation
errors = RANKS;
size_ratios = RANKS;

for k = RANKS
    CAPTION_RANK = int2str(k);
    FULL_CAPTION = {CAPTION_START, [CAPTION_END, CAPTION_RANK]};

    FILE_NAME = filename(FOLDER, strcat(FILE_NAME_START, CAPTION_RANK));
    [A_compressed, size_ratios(k), errors(k)] = ...
        compress_and_create_image(A, k, FILE_NAME, uncompressed_size);

    add_subplot(k+1, A_compressed, FULL_CAPTION);
end
```

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

I create a figure of `2×2` subplots where the uncompressed image is first plotted in the top-left corner,

corresponding to position `1`, using `intens`. That function does simply what was specified in the project description, i.e. converts the matrix to a gray-scale plot and adds a title specified as an input parameter.

```
function [] = intens(M, img_title)

imagesc(round(M));
axis image;
colormap(gray);

title(img_title)
xlabel 'Column index'
ylabel 'Row index'
```

To be able to save the gray-scale images with `imwrite` I had to *normalize* the matrix, that is divide each element with the max value of the entire matrix. This is done by the function `normalize_matrix`.

```
function M_norm = normalize_matrix(M)

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

In the `for` loop, I call `compress_and_create_image` for each of the ranks 1,2 and 3. That function looks like this,

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

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

ratio = compare_size(uncompressed_size, FILE_NAME);
error = norm(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 saved as a image, and the compression ratio is calculated as the difference in size between the compressed and uncompressed image files. The error is calculated as the norm of the difference between matrices.

Finally, in every loop iteration `add_subplot` is called.

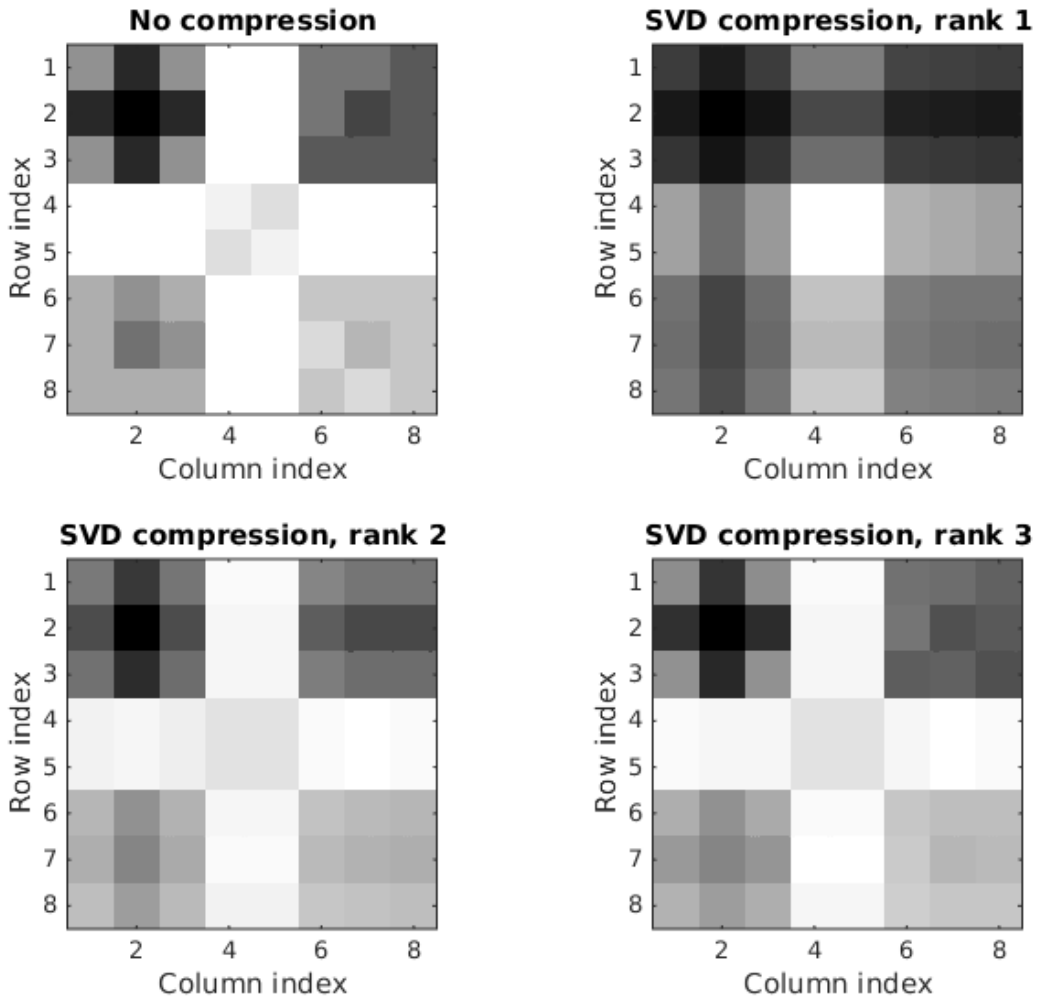
```
function[] = add_subplot(subplot_position, M, CAPTION)
```

```
subplot(2, 2, subplot_position);
intens(M, CAPTION);
```

The resulting `2×2` subplot is presented below, where it's clear that the compressed image resembles the original more when rank is increased from 1 to 2, and even more from 2 to 3.

Ranks 2 and 3 are *passable* as representations of the uncompressed image, whereas the rank 1 image is compressed to the point where 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 well over 90%, so nothing is really gained from compressing `8×8` pixel images like this. [Using SVD of rank > 3, defeats the purpose of the compression because]



SVC compresssion rank	Size ratio	Error
1	0.95592	152.85
2	0.98347	50.595
3	0.99174	18.553

For the the second part I started by downloading `nena.jpg` from the web page, and wrote a script for the gray-scale 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 then call functions `assignment_2_grayscale` and `assignment_2_color`.

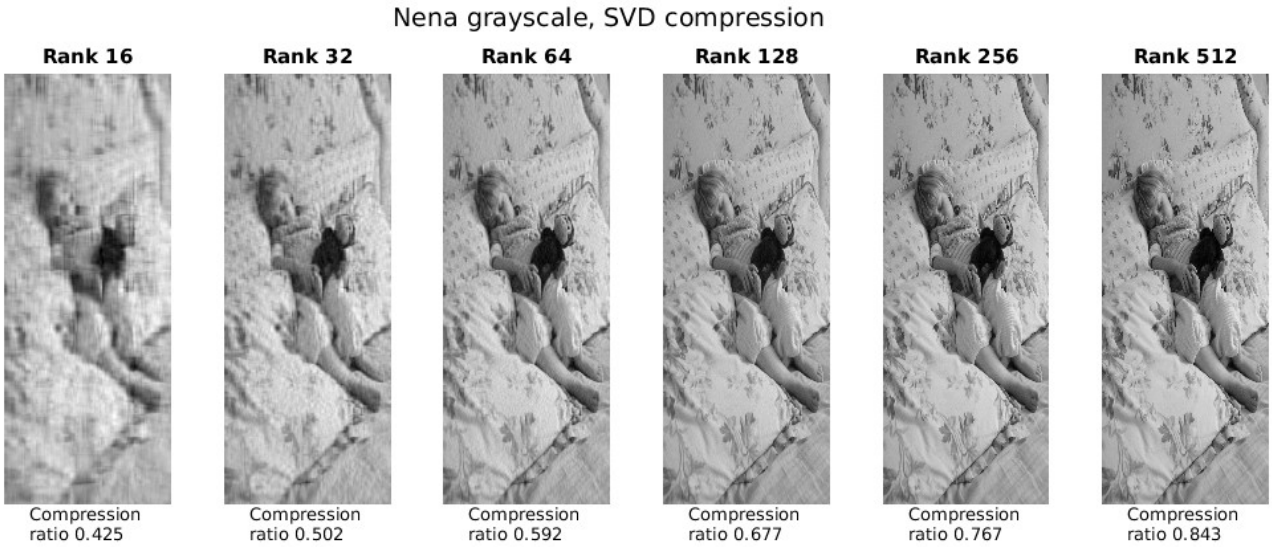
The first on of those looks as follows,

```
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_';

RANKS = 2 .^ (0:9);

nena_grayscale = double(rgb2gray(nena));
imwrite(normalize_matrix(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;

for k = rank_indices
    RANK_STRING = int2str(RANKS(k));
    FILE_NAME = filename(FOLDER, strcat( ...
        FILE_NAME_START, RANK_STRING));

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

where the exponential values of `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 file.

For compression ranks `2?` and below, images are mainly noise and basically useless for any

application. At `2?` pretty good quality is achieved with compression ratio `0.6774`, and for rank `2?` great quality is achieved with compression ratio `0.7667`. For even higher values, the increase in quality is hardly worth is since the image size quickly approaches the size of the uncompressed image.



The color image manipulation is performed by `assignment_2_color`.

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

FOLDER = 'nena_color';
RANKS = 1:3:100;

nena_double = double(nena);

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

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

for k = rank_indices
    [compressed_tensor, ratio, E] = compress_3_components( ...
        FOLDER, nena_double, nena_size, RANKS(k));

    errors(k) = E;
    size_ratios(k) = ratio;

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

```
assignment_2_video(F, RANKS);
```

The compression of a color image is more complicated than for gray-scale, 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.

```
function [compressed_tensor, size_ratio, error_median] = ...
    compress_3_components(FOLDER, uncompressed, uncompressed_size, rank)

COMPONENTS = 3;
FILE_NAME = '/nena_compressed_rank_';

%preallocation of space
compressed_tensor_normalized = uncompressed;
compressed_tensor = uncompressed;
error_sum = 0;

for i = 1:COMPONENTS
    component = uncompressed(:,:,i);
    compressed_component = svd_compression(component, rank);
    compressed_tensor_normalized(:,:,i) = ...
        normalize_matrix(compressed_component);
    compressed_tensor(:,:,i) = compressed_component;

    error_sum = error_sum + norm(component - compressed_component);
end

error_median = error_sum / COMPONENTS;

file_name = filename(FOLDER, strcat(FILE_NAME, int2str(rank)));
imwrite(compressed_tensor_normalized, file_name);

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

Finally, to create a video from compressed images of increasing, the function `assignment_2_video` is used. The particular values for `RANKS` specified in `assignment_2_color` produces probably the best illustration of the differences between ranks when presented in movie form. The variable `F` is assigned a frame for every compressed color image created, and then passed to the video function.

```
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 your data from Matlab, including into video. I got better acquainted with how images are represented internally, which was eye-opening.

I struggled initially, 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, although I finally got through it. Same thing with the video functionality where the Matlab help pages are not always intuitive for new users, although the Matlab Central forums are usually easier to use. Going through the painstaking process of finding the right function for a situation was still probably a worthwhile endeavor, and I feel much more comfortable with using Matlab as a proper scripting language now. It has lots of power.