EE 046200 - Technion - Image Processing and Analysis

Computer Homework 3

Due Date: 30.06.22

Submission guidelines

READ THIS CAREFULLY

- **Please notice:** Some of the exercises contain questions on topics that are yet to be taught in the lecture or the frontal exercises. You may consider them as background or preparation questions to the topic before learning about it in class, or you may wait until the topic is taught, and solve only the questions on the topics you already learned.
- **Avoid unethical behavior.** This includes plagiarism, not giving credit to source code you decide to use, and false reporting of results. Consulting with friends is allowed and even recommended, but you must write the code on your own, independently of others. The staff will treat unethical behavior with the utmost severity. **!אנא המנעו מהתנהגות שאינה אתית והעתקות**
- Code submission in **Python only**. You can choose your working environment:
  - You can work in a Jupyter Notebook , locally with Anaconda (the course's computer HW will not require a GPU).
  - You can work in a Python IDE such as PyCharm or Visual Studio Code. Both also allow opening/editing Jupyter Notebooks.
- The exercise must be submitted **IN PAIRS** (unless the computer homework grader approved differently) until **Thursday 30.06.2022 at 23:55**.
- The exercise will be submitted via Moodle in the following form: You should submit two **separated** files:
  - A report file (visualizations, discussing the results and answering the questions) in a `.pdf` format, with the name `hw3_id1_id2.pdf` where `id1`, `id2` are the ID numbers of the submitting students.
    - Be precise, we expect on point answers. But don't be afraid to explain you statements (actually, we expect you to).
    - Even if the instructions says "Show/Display...", you still need to explain what are you showing and what can be seen.
    - No other file-types (`.docx` , `.html` , ...) will be accepted
  - A compressed `.zip` file, with the name: `hw3_id1_id2.zip` which contains:
    - A folder named `code` with all the code files inside (`.py` or `.ipynb` ONLY!)
      - The code should be reasonably documented, especially in places where non-trivial actions are performed.
      - Make sure to give a suitable title (informative and accurate) to each image or graph, and also to the axes. Ensure that graphs and images are displayed in a sufficient size to understand their content (and maintain the relationship between the axes - do not distort them).
    - A folder named `my_data` , with all the files required for the code to run (your own images/videos) and all the files you created. make sure to refer to your input files in the code locally. i.e. (if the code is in 'code' directory, and the input file is in a parallel 'my_data' directory): `img = cv2.imread('../my_data/my_img.jpg')`
    - **DO NOT** include the given input data in the zip. The code should refer to the given input data as it is located in a folder named `given_data` . i.e.: `img = cv2.imread('../given_data/given_img.jpg')`
- If you submit your solution after the deadline, 4 points will be reduced automatically for each of the days that have passed since the submission date (unless you have approved it with the course staff before the submission date). Late submission will be done directly to the computer homework grader via mail, and not via Moodle.
- Several Python, numpy, openCV reference files are attached in the Moodle website, and you can of course also use the Internet's help.
- Questions about the **computer** exercise can be directed to the computer homework grader through the relevant Moodle forum or by email.

General Notes:

In [1]:
```python
# imports for the HW
import numpy as np
import matplotlib.pyplot as plt
import cv2
import glob
```

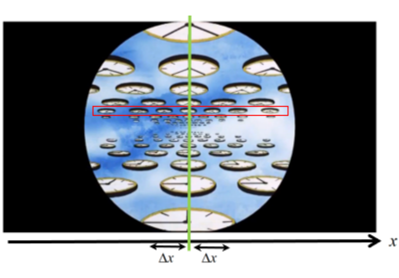
Part 1 - Sampling and Aliasing in Images

In This part we will sample images **in space and in time**, and examine the phenomenon of **aliasing** in both types of sampling.

In order to do so, we will use the music video of the song Time by Pink Floyd. You are encouraged to open the file `Time - Pink Floyd.mp4` , to watch the video and enjoy the music. :)

1.a - Spatial Sampling:

1. Sample the first frame from the 33rd second of the `Time - Pink Floyd.mp4` video and display it. You may use the `video_to_frames` function from your first python HW. For the spatial sampling in this section we will use the region in the frame which contains a row of 7 clocks, as seen in the red rectangle in the following figure:



2. Let us examine the `292` nd row of the image. Create a copy of the original image in which this row is marked in red and display it. In addition, create and display a graph containing the gray levels of the `292` nd row of one of the color channels (your choice) as a function of the column index.

3. Now, for the spatial sampling: sample the image with sampling interval of $\Delta x = 64$ (meaning - sample only the columns of the image, with no sampling on the y-axis). The sampling will start at the middle of the image (the green line in the above figure) and continue towards both directions (i.e., the central column of the image is sampled, and then all the columns in the image that are of $n \cdot \Delta x$ distance from it, for $n \in \mathbb{Z}$ (within image borders, of course). Display the sampled image and a copy of the original image in which the sampled columns are marked in red.

4. In order to evaluate the result of sampling we would like to return the image to its original dimensions. We will do so by interpolating on the column dimension, using `cv2.resize` that uses bilinear interpolation by default. Return the sampled image to its original dimensions and display the result.

5. Examine our region of interest (the red rectangle in section 1.a.1) in your new image. How many "smeared" clocks can you identify now? Create and display a graph containing the gray levels of the `292` nd row in one of the color channel you previously chose as a function of the column index for the new image. What are the differences between this graph and the graph from section 1.a.2? Explain.

1.b - Temporal Sampling:

1. In this part we would like to perform a temporal sample on a time section from the video. Watch the time section between the seconds `30-45` . In this part we will use only this time section. In what direction do the clocks in the time section progress in space? In what direction do the clocks hands turn? Load all the frames from this time section. You may use the `video_to_frames` function from your first python HW.

2. Sample the time section with sampling interval of $\Delta p = 16$ (meaning - the frames indexed 0, 16, 32, etc...). In order to examine the influence of the temporal sampling, create a new video having the duration of the original time section (15 seconds) and the same FPS rate. In order to do so, use Zero-Order Hold interpolation: every frame in your sampled video will be translated in the new video into $\Delta p = 16$ consecutive frames. Create the video and save it in `mp4` format. **Attach the video to your submission in the my_data folder**. (Note that the video does not need to contain sound).

3. Watch the video you created - in what direction do the clocks progress now? In what direction do the clocks hands turn? Explain.

Part 2 - PCA Compression

One of the ways to compress images is by using dimensionality reduction and saving the image representaion in the lower dimension. One of the classic methods for dimensionality reduction is Principal Component Analysis (PCA).

In This part we will examine this method and its performance on a set of images containing faces of people: Labelled Face In the Wild (LFW). The set consists of `13233` gray-scale images of size `64x64` .

- **Note:** You ar not allowed to use PCA functions already implemented in Python packages in this part.

2.a - Pre-processing & covariance matrix

1. Uncompress the dataset in a sub-folder named `LFW` in the `given_data` folder. Load all of the images into a 3D numpy array of size `64x64x13233` (after converting them to grayscale). Display 4 images from the data set (remember their indices - these are the images you will restore later in the exercise).
   - **Note: DO NOT upload the data set as part of your submission!**

2. Define numpy array `X` of size `4096x13233` in which every column represents one of the images in column-major representation (don't forget the `'F'` argument in `np.reshape`).

3. Calculate the mean of every pixel in the `X` array (results in a `4096` vector) and display it as a `64x64` image (don't forget the `'F'` argument in `np.reshape`). Now, transform the mean vector into a `4096x1` array and call it `mu` . Subtract `mu` from `X` to get an array of data centered around 0 - call it `Y` .

4. Calculate the covariance matrix of `Y` using `np.cov` . Note that the size of the covariance matrix is depended on the size of each sample, and in our case we should get a matrix of size `4096x4096` .

2.b - Principal Components

The eigenvectors of the covariance matrix are called the **Principal Components**. If one would calculate all of the principal components of our covariance matrix, they will get a spanning set of the $\mathbb{R}^{4096}$ space. In practice, in order to perform dimensionality reduction, we want to project our data into a lower dimension. In PCA we do so by projecting every image into a space with a spanning set of k eigenvectors (principal components) corresponding to the k largest eigenvalues of the covariance matrix.

Calculate the `k=10` largest eigenvalues of the covariance matrix and their corresponding principal components.

- The parameter `eig_vals` will contain the `k=10` eigenvalues in descending order (largest eigenvalue first).
- The parameter `eig_vecs` will contain a matrix in which the columns are the corresponding principal components in order respective to `eig_vals` .
- The covariance matrix is symmetric, so you can use `np.linalg.eigh` in order to find them.

Display a plot of the `eig_vals` vector. In addition, display the first 4 principal components in `eig_vecs` as images of size `64x64` (don't forget the `'F'` argument in `np.reshape`).

2.c - Compression by projection

Find the projection of every image in the space spanned by the `k=10` principal components by calculating:

$$P = V^T Y$$

where V is the `eig_vecs` matrix and Y is the data matrix `Y` .

Note that now each column of the matrix P is actually a representation of one image from the data set (after subtracting the mean), in the lower dimension.

2.d - Restoration

For the 4 images you presented in section 2.a.1, extract the appropriate columns in the P matrix. We will denote each column as p_i where $i \in \{1, 2, 3, 4\}$. Find $\hat{x}_i$, the restoration of each column by applying:

$$\hat{x}_i = V p_i + \mu$$

Where μ is the calculated mean `mu` . Display the 4 restored images (sized `64x64`). In the title of each image display the MSE between the original image and the restoration. Reminder:

$$MSE_i = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} (x_i[m,n] - \hat{x}_i[m,n])^2$$

Where x_i is the original image corresponding to column p_i in X.

2.e - Changing k value

Now, compress `Y` using `k=570` principal components, and restore and display again the 4 images you chose. What do you think of the restored results now? Compare to the `k=10` case. Note that although we have enlarged the dimension of the images in the low dimesional space, it is still less than 7 times (!) smaller than the original image.

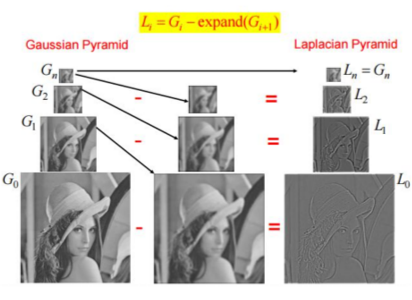
Part 3 - Multiple Resolutions

Multi-resolution pyramids are a very useful tool in the field of image processing. In this part you will learn how to calculate these pyramids and you will implement a useful app using them.

More extensively, you will:

- construct a Gaussian pyramid and a Laplacian pyramid out of an image
- reconstruct the image from the Laplacian pyramid
- implement image blending using multiple resolutions

The construction of a Gaussian pyramid and a Laplacian pyramid is depicted in the following illustration:



Gaussian pyramid: each level of the Gaussian pyramid (the left column in the above illustration) is created by filtering of the previous level using a Gaussian filter, and then sampling one pixel from every two pixels in each axis (reduce operation). The 0th level is the original image.

Laplacian pyramid: each level of the Laplacian pyramid (the right column in the above illustration) is created by calculating the difference between the corresponding Gaussian level and the next Gaussian level after expansion is applyed on it (expansion - upsampling the source image by injecting even zero rows and columns and then convolving the result with the same Gaussian filter as in the Gaussian pyramid, multiplied by 4). For Gaussian and Laplacian pyramids with n levels, the nth level of the Laplacian pyramid will be equal to the nth level of the Gaussian pyramid. In fact, the Laplacian pyramid divides the image into different frequency ranges.

3.a - Pyramids creation

Implement the function `pyr_gen` that constructs Gaussian and Laplacian pyramids out of an image.

In [ ]:
```python
def pyr_gen(n, m, img, gauss_pyr, laplace_pyr):
    """
    Constructs Gaussian and Laplacian pyramids out of an image.
    :param n: number of pyramid levels
              (excluding the 0th level - total number of levels - n+1).
    :param m: current pyramid level
    :param img: The input gray-scale image.
                The m-1 level of the Gaussian pyramid.
                np array of of type uint8.
    :param gauss_pyr: The Gaussian pyramid so far.
                      Python list of length [m-1] containing the pyramid levels
                      as np arrays of type uint8.
    :param laplace_pyr: The Laplacian pyramid so far.
                        Python list of length [m-1] containing the pyramid levels
                        as np arrays of type uint8.
    :return:
        gauss_pyr: The Gaussian pyramid.
                   Python list of length [n-m+1] containing the pyramid levels
                   as np arrays of type uint8.
        laplace_pyr: The Laplacian pyramid.
                     Python list of length [n-m+1] containing the pyramid levels
                     as np arrays of type uint8.
    """
    assert m>=0 and m<=n
    # ====== YOUR CODE: ======

    # ========================

    return gauss_pyr, laplace_pyr
```

Guidance:

- You may use the `cv2.pyrUp` and `cv2.pyrDown` functions in your implementation.
- You should implement the function recursively.

Now, load the images `Ironman.jpg` and `Downey.jpg` and convert them to grayscale. Build for both of them Gaussian and Laplacian pyramids with `n=4` levels using `pyr_gen` . Display the results: two figures, one for every image. The top row of each figure will contain the Gaussian pyramid, and the bottom row will contain the Laplacian pyramid (5 images in each row). The images can be displayed in the same size, or in different sizes (your choice).

3.b - Reconstruction using Laplacian pyramid

Implement the function `laplace_recon` which reconstructs an image by summing all of its Laplacian pyramid levels. The summation will start from the top of the pyramid (level n - smallest image) and will continue iteratively: Perform expansion on the ith level and sum it with the $(i-1)$th level, until reaching the bottom of the pyramid.

In [ ]:
```python
def laplace_recon(laplace_pyr):
    """
    Image reconstruction from Laplacian pyramid.
    :param laplace_pyr: The Laplacian pyramid.
                        Python list containing the pyramid levels
                        as np arrays of type uint8.
    :return:
        recon_img: The reconstructed image.
                   2D np array of the same shape as laplace_pyr[0].
    """
    # ====== YOUR CODE: ======

    # ========================

    return recon_img
```

Now, reconstruct both of the images using `laplace_recon` and display the reconstructed images and the subtraction images between the original and reconstructed images. Calculate and display the MSE between the original images and the reconstructions.

3.c - Image blending

Create a mask of the same size as the original images. The mask will contain `1` s in its left half and `0` s in its right size. Create a Gaussian pyramid out of this mask.

Create a new Laplacian pyramid. let us define $L^{(i)}_{Ironman}$ as the ith level of the Laplacian pyramid of the `Ironman` image, $L^{(i)}_{Downey}$ as the ith level of the Laplacian pyramid of the `Downey` image, and $G^{(i)}_{mask}$ as the ith level of the Gaussian pyramid of the mask. The ith level of the new pyramid, $L^{(i)}_{blend}$, will be defined as follows:

$$L^{(i)}_{blend} = G^{(i)}_{mask} L^{(i)}_{Ironman} + (1 - G^{(i)}_{mask}) L^{(i)}_{Downey}$$

Now use `laplace_recon` on the new pyramid and display the reconstructed image. Explain the result.

Credits

- Icons from Icon8.com - https://icons8.com