Assignment 1
Pixel Magic: Mastering Some Basic Image Processing Techniques

Please note:

1. What you must hand in includes the assignment report (.pdf) and – if necessary – source codes (.py). Please zip them all together into an archive file named according to the following template: XXXXXXXX.zip
   Where XXXXXXXX must be replaced with your student ID.
2. Some problems are required to be solved *by hand* (shown by the icon), some need to be implemented (shown by the icon), and some need to be dockerized (shown by the icon).
3. As for the first type of the problems, you are free to solve them on a paper and include the picture of it in your report. Here, cleanness and readability are of high importance.
4. Your work will be evaluated mostly by the quality of your report. Don't forget to explain what you have done, and provide enough discussions when it's needed.
5. 5 points of each homework belongs to compactness, expressiveness and neatness of your report and codes.
6. By default, we assume you implement your codes in Python. If you're using Matlab, you have to use equivalent functions when it is asked to use specific Python functions.
7. Your codes must be separated for each question, and for each part. For example, you have to create a separate .py file for part b. of question 3. Please name it like p3b.py.
8. Problems with bonus points are marked by the icon.
9. You may submit your work up to 10 days late for all assignments. After that, a 5% penalty will be applied per day.
10. **Please upload your work in Courses, before the end of 14 March.**
11. If there is ***any*** question, please don't hesitate to contact us through the following email addresses:
    - Atghaei@aut.ac.ir
    - fardin.aiar@gmail.com
    - minoo.dolatabadi75@gmail.com
    - aidin.khalili@ymail.com
    - b.roshanfekr@aut.ac.com
12. Unfortunately, it is quite easy to detect copy-pasted or even structurally similar works, no matter being copied from another student or internet sources. Try to send us your own work, without being worried about the grade! ;)

1. Exploring Color Spaces and Image Transformations (15 Pts.)

Digital images, as you are probably very well aware, are represented as matrices of numerical values in mathematical terms. Since images are structured as matrices, various mathematical operations can be applied to them. This problem set is designed to help you explore the relationship between digital images and mathematical operations while enhancing your understanding of different color spaces. You will perform fundamental image transformations, analyze their effects, and compare different color representations commonly used in image processing.

a. Load the RGB image "Q1.jpg" as the input image. Then, visualize and analyze each of its color channels separately (Red, Green, and Blue).

b. Swap the Red and Blue channels of the RGB image and display the modified image. Explain the visual impact of this operation and how it alters the image's appearance.

c. Convert the RGB image to a grayscale image using the following formula:

$$I = 0.299R + 0.587G + 0.114B$$

Display the grayscale image and explain the significance of this weighted conversion. How does it differ from simply calculating the average of the different channels?

d. Explain whether the RGB to grayscale conversion is always possible. Provide insights into the implications of this transformation and any potential information loss.

e. Convert the RGB image to the HSV color space and display each channel separately (Hue, Saturation, and Value). Discuss the meaning of each channel and how the HSV representation benefits image analysis and manipulation.

f. Convert the RGB image to the YCbCr color space and display each channel separately. Explain the significance of the YCbCr model, particularly in applications like image compression and television broadcasting, and compare its characteristics to the RGB model.

g. Discuss the advantages and disadvantages of working with different color spaces (RGB, HSV, YCbCr) in various image processing applications. Provide examples of scenarios where each color space is preferable.

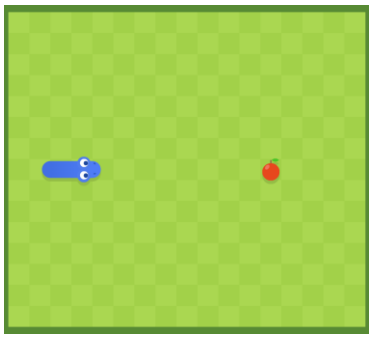
h. Load three separate grayscale images that represent different color channels (Q1h1.jpg, Q1h2.jpg, Q1h3.jpg), but without knowing which one corresponds to Red, Green, or Blue. Combine them properly to reconstruct an RGB image. Determine a method to correctly assign each grayscale image to its corresponding channel, and explain your approach.

2. The Snake Game is Here! (15 Pts.)

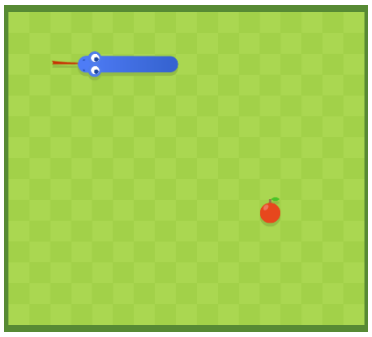
The Snake game is a classic and widely recognized arcade game that has been enjoyed across various platforms and generations. The game involves controlling a snake that moves across the board, consuming targets to grow longer while avoiding collisions with itself and the board boundaries.

In this exercise, you are provided with two images representing different states of the game, named "Q2_snake1.png" and "Q2_snake2.png".

a. Using these two images, extract the different components of the game, including:
   - The empty game board
   - The apple
   - The snake before eating the apple
   - The snake after eating the apple



b. Using the image "Q2_snake1.png" and the extracted components from part (a), simulate the movement of the snake as it eats the apple. Generate frames for each step of the game and create a GIF documenting the gameplay. The output GIF should start with "Q2_snake1.png" and end exactly after the snake eats the apple.



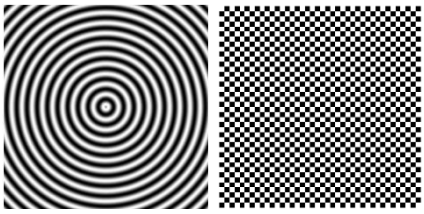
3. Digital Image Acquisition (20 Pts.)

The goal of this assignment is to understand the process of digital image acquisition by generating images from mathematical functions and analyzing the impact of different sampling resolutions and distortions.

You are provided with the following mathematical functions.

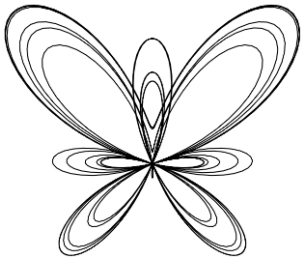
i. $I(x,y) = 255 \times \sin\left(20\pi\sqrt{x^2+y^2}\right)$

ii. $I(x,y) = \begin{cases} 255 & \sin(20\pi x)\sin(20\pi y) > 0 \\ 0 & otherwise \end{cases}$



a. Write a function that takes a mathematical function and image size as input, generates the corresponding image, and saves it to a predefined path.
b. Use your function to generate images for the given functions. (Consider images of size 512×512 pixels.)

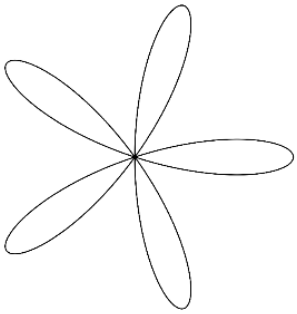
Now, consider the following parametric equations:



i.

$$x = \sin(t)\left(e^{\cos(t)} - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right)$$
$$y = \cos(t)\left(e^{\cos(t)} - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right)$$
$$0 \le t \le 12\pi$$

ii.



$$x = \cos(k\theta)\cos(\theta)$$
$$y = \cos(k\theta)\sin(\theta)$$
$$0 \le \theta \le 2\pi, \quad k = 5$$

c. For these functions generate images by plotting the equations.
d. Generate each function-based image at different resolutions (e.g., 100x100, 500x500, 1000x1000 pixels).
e. Compare how the resolution affects the quality and clarity of the generated images.

Submission Requirements: A report including A brief explanation of each function used, Generated images with different resolutions, and A comparison of image quality at different resolutions.

4. Extracting Relevant Colors from an Image (15 Pts.)

Image Segmentation is the process of partitioning an image into several segments, i.e. sets of pixels known as super-pixels. Segmented image is the simplified version of the image and could be more meaningful and easier to analyse, as can be seen in figure 1.

Figure 1: Example of image Segmentation

Most image segmentation methods are based on color information of image pixels. One primary step of these methods is to find a set of most relevant colors of an image, or an image "palette", and assign each area of the image to a color from this set. Finding this set is also called **Color Quantization**. Figure 2. Shows an example of an image and its dominant color and palette colors set.

Figure 2: Finding dominant color and palette sets of an example image

Your task is to do something similar in python, i.e. to write a function which finds the most frequent colors of an input image. Your function input is the image and a parameter which determines the size of the palette, and its output is a set of 100x100 boxes, each with a specific color.

Consider the data in the file "Q4_us_president.txt". This file contains a large number (210012) of length 3 vectors, each on one line. Each vector represents the red, green, and blue intensity values of one of the pixels in the image shown at the right. The image has 516 rows and 407 columns. The pixels in the file are listed row by row from top to bottom, and within each row from left to right. For example, the first pixel in the file is the uppermost left pixel in the image. The second line of the file contains the pixel to the right of that one, and so on.

a. Read this image and find its dominant color and palette set. Set the palette size to the following values: 5, 7, 9.
b. Apply image segmentation to the image using part a.

5. Creating an Image Mosaic using OpenCV (15+5 Pts.)

You are given a "folder of images" containing multiple small images of a single person. Additionally, you are given a "target image" of the same person in a well-positioned frame. Your task is to write a Python program using OpenCV to generate a mosaic image, where the target image is reconstructed using the smaller images from the given folder like the image below:

Requirements:

i. The target image should be divided into small tiles.
ii. Each tile should be replaced with the best-matching image from the given folder.
iii. The matching should be based on color similarity (e.g., by comparing the average color of the tile with the average color of each image in the folder).
iv. The final output should resemble the target image but be composed of the smaller images.

Input:

- A folder (images/) containing multiple small images of the person.
- A target image (target.jpg) of the same person.

Output:

- A mosaic image (mosaic.jpg), where the target image is formed using the images from the folder.

Additional Requirement:

- You must write a **Dockerfile** to build an image that installs all required dependencies and generates the output file.

Hints:

- Resize each small image in the folder to match the size of a tile.
- Divide the target image into a grid of tiles.
- For each tile, find the most visually similar image from the folder.
- Replace each tile with the best-matching image and reconstruct the mosaic.
- You can select any images you like and place them in the folder, but your algorithm must automatically determine the most suitable ones.
- It's important to note that there is no restriction on the number of images in the folder, and your code should be capable of handling any quantity and type of images.
- The choice of the target image and the images in the folder is entirely up to you, so feel free to select them as you prefer.

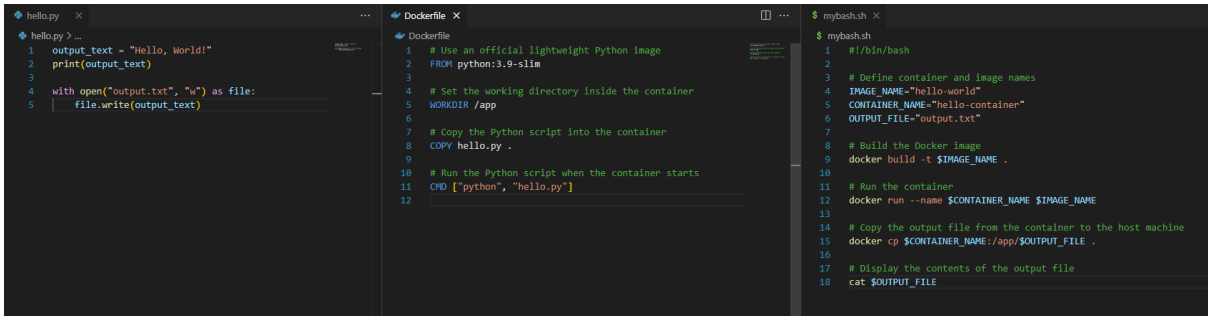
Bonus Challenge (Optional):

- Use more advanced similarity measures (e.g., histogram matching instead of just average color).
- Optimize performance for large images and large datasets.

Example:

Given a folder containing 100+ small images of a person and a target image, your program should generate an artistic mosaic in which the target image remains visible but is composed of the smaller images.

As an example, the figure below shows a Dockerfile running a Hello World Python script.



hello.py

```
output_text = "Hello, World!"
print(output_text)

with open("output.txt", "w") as file:
    file.write(output_text)
```

Dockerfile

```
# Use an official lightweight Python image
FROM python:3.9-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the Python script into the container
COPY hello.py .

# Run the Python script when the container starts
CMD ["python", "hello.py"]
```

mybash.sh

```
#!/bin/bash

# Define container and image names
IMAGE_NAME="hello-world"
CONTAINER_NAME="hello-container"
OUTPUT_FILE="output.txt"

# Build the Docker image
docker build -t $IMAGE_NAME .

# Run the container
docker run --name $CONTAINER_NAME $IMAGE_NAME

# Copy the output file from the container to the host machine
docker cp $CONTAINER_NAME:/app/$OUTPUT_FILE .

# Display the contents of the output file
cat $OUTPUT_FILE
```

6. Solving a Jigsaw Puzzle Using Basic Image Operations (20 Pts.)

It’s hard to find someone who has never played a jigsaw puzzle game. However, it’s even harder to find someone who has solved one using image processing techniques—a challenge you are about to tackle.

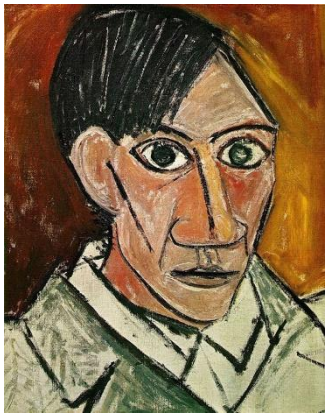
Beyond being a fun exercise, this task will help you practice fundamental image operations such as cropping, resizing, rotation, translation, and more.

The figure below shows an image containing 15 pieces of a 5 × 3 jigsaw puzzle depicting Self-Portrait, oil on canvas, 1907. Your task is to extract the pieces and then assemble them to reconstruct the complete image at a final size of 954 × 750 pixels. Although this may seem challenging, it can be accomplished using simple image operations on the input.

Now, let’s get started! First, load the image **"Q6.jpg"**.

a. Extract the pieces. Note that the background color is white, making it quite easier to extract the pieces. Save all the extracted images with a name according to the template [piece_01.jpg].
b. Now comes the main part. For your convenience, the piece in the top left corner is placed in its right location. Start with this piece and find its neighbors one by one using a similarity metric of your choice. Note that some pieces need to be rotated. Therefore, it is advised to consider all 3 states (0 ,90 and 180 degrees) when comparing pieces. The output of this step must be a 5 x 3 matrix specifying the correct position of pieces based on their numbers.
c. Use the matrix you obtained in the previous part to complete the puzzle. Display and save the resultant image.

Hint: To avoid getting mixed up, save the intermediate pieces into separate folders.



Good Luck!