Project

Deadline: 15.11.2024 23:59 SGT

The project is an assignment that has to be completed in groups of 3 persons maximum (the persons **have to be** in the same lab group). This project will test your ability to use Python in a more concrete and complex setting than what you were used to during the tutorials/lab sessions. Also, it will improve your ability to work as a team when having a project to complete. The goal of the project is to program a picture editor in Python, allowing a few functionalities such as brightness/contrast change, blur effect, etc. and with a simple user interface.

Very Important!!! Please follow the instructions below. Projects that will not **strictly follow** these instructions will risk getting big grade penalty.

- The Python script to execute the program must be in your project folder and named exactly `picedit.py` (no uppercase letter). For submission of your project on NTU Learn, please create a zip archive from your project folder, name the archive according to your matric number (i.e. `<YourMatricNumber>.zip`), and upload it on NTU Learn. If two or three people are in your group, simply separate the matric numbers with an underscore character.
- The submission system automatically closes exactly at the deadline. Hence, after that, if you didn't submit, you will get 0 point for your project. You can update your submission as many times as you want on NTU Learn, only the last submission will be taken into account. Thus, I advise you to submit an earlier version much before the deadline, to be sure you won't end up with no submission at all.
- Only one person of your group submits the project on NTU Learn. Do not submit the same project for all your group members.
- Make sure your project works properly (i.e. the program doesn't output errors). If errors are output during the test of a functionality of your project, this functionality will be considered as not working at all.
- Do not copy any code (or part of) from other groups or from implementations available online. Special software will be run to check for such cheating cases. If you are caught copying code, you will risk serious consequences (see course guidelines). Also, do not let other groups copy your code, as both groups with similar code will get 0 mark for the project (regardless on who copied on who). Note that every year several groups get 0 mark (or worse: academic misconduct) because of plagiarism. To avoid any possible issues, I would strongly advise you to not collaborate among different groups.
- To have full grades, your Python script must implement the functions `change_brightness`, `change_contrast`, `grayscale`, `blur_effect`, `edge_detection`, `embossed`, `rectangle_select`, `magic_wand_select` and `menu` (see below). Make sure you don't miswrite the functions names, or make an error in the input/output that are expected for these functions. The grading will be mostly be based on these functions. You can find on NTU Learn a skeleton of the `picedit.py` file. You are required to start from this file.
- You can also find on NTU Learn the Python file `test.py` and the tiny image file `mini_test.jpg` to pre-test your functions (simply copy the files in your folder and run it). It should output OK for all tests (note that getting a OK does not necessarily mean your function is working perfectly). You can also find on NTU Learn a skeleton of the picedit.py file, please start from this file.
- You should not have any Python code that does not belong to a function (i.e. all your Python code must be inside a function), except the module imports at the very beginning, and except the call to the `menu` function, which will start your program. You can define and use your own functions in addition to the existing ones.

1 Picture Editor

1.1 Objectives

The objective of the project is to write in Python a **working** picture editor program. The program must allow the user to load a picture, save it, and apply various filters/transformations on the loaded picture (see below). All this via a simple menu interface through the console. You have to use the skeleton file `picedit.py` provided on NTU Learn.

Note that everytime you add a feature to your program, you should test it thoroughly before continuing. Testing your program only at the very end is the best way to render the bug hunting close to impossible ! In order to not make the program too slow, I advise you to only use pictures with at most 800 pixels of width and height.

2 Three steps to complete the project

2.1 1st step: implementing the skeleton of the project and the user interface

The first step in a programming project is perhaps the most important one: before writing any Python code, you should think about the functions you will need to implement, their input/output, their goal, the overall structure of the entire program. This step has already be done for you and you have to use the skeleton file `picedit.py` provided on NTU Learn. Make sure you understand what each function is supposed to do and try to get an idea of how the entire program will be organised. In short, you have a `menu` function to handle the interface with the user, and one function for each filter/transformation functionality to be added.

2.1.1 Data Structure for the picture editor.

In order to represent a picture in Python, we will use a Numpy three-dimensional array: actually a two dimensional array (for row/column positions in the picture) and another dimension to hold the three Red-Green-Blue (RGB) integers for that pixel. Note that each RGB value is an integer between 0 and 255 included (the first/second/third one represents the amount of red/green/blue color respectively). Here is an example of an image of 2 by 2 pixels:

```
np.array([ [[91,0 ,0], [155,32,113]], [[74,0,177], [139,255,33]] ]
```

First, import the Numpy module using:

```
import numpy as np
```

For example, you can create an all-grey picture `img` (all RGB values filled to 145 for example) of r rows and c columns using:

```
img = np.full((r, c, 3), 145)
```

Finally, the RGB values of the pixel in the picture located at row r and column c can be accessed using `img[r,c]`. Its green component can then be accessed using `img[r,c,1]`. Slicing for Numpy arrays works similar to lists.

In addition to image arrays, we will be manipulating selection masks. A selection mask will define the set of pixels from the image that have currently been selected by the user (typically when the user would like to apply a filter only on some portion of the image). This is simply represented by a two dimensional Numpy array with the same number of rows and columns as the corresponding picture array. An entry `mask[r,c]` will be equal to 0 if the pixel located at row r and column c has not been selected by the user (equal to 1 otherwise).

2.1.2 Loading/Saving a RGB picture.

You can load a picture very easily in Python. First, import the following libraries by typing at the beginning of your script file:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

Assuming that `filename` is a string containing your file name, you can run

```
img = mpimg.imread(filename)
```

which will load the picture's RGB values in a variable named `img`. A function `load_image(filename)` has already been implemented for you in the skeleton file. It takes as input a string `filename` (the file from which you want to load, it can be a file with ".jpg" or ".png" extension) and will output two objects: the image loaded and a selection mask set by default to the entire image size (i.e. a two-dimension array full of 1: all pixels are selected by the user).

Similarly, you can save a picture from an array named `img` in a file named `filename` by calling:

```
mpimg.imsave(filename, img)
```

Again, a function `save_image(filename, img)` has already been implemented for you in the skeleton file. It takes as input a string `filename` (the file name to which you want to save, don't forget to add the ".jpg" extension) and an image `img` (the image you want to save). It doesn't output anything.

2.1.3 Displaying a RGB picture.

A function `display_image(img, mask)` has already been implemented for you in the skeleton file. It takes as input an image array `img` and a selection mask `mask` and does not output anything. It will simply display the picture in the console and show the shape of the set of selected pixels (according to `mask`) using red color. If using Spyder, please go to "Tools → Preferences → IPython console → Graphics → Graphics Backend" and select "inline" ... you can also adjust the picture display size in the console by changing the width and height values in "Inline Backend".

2.1.4 Interacting with the user.

You will have to implement the `menu` function. This function's role is to handle the menu interface with the user via the console. It is basically the director function that will interact with the user and distribute the work to all functions, depending on the user's choices. Note that **all** the calls to the `display_image()` and `input()` functions should be in the `menu` function (you should not have any `display_image()` or `input()` calls anywhere else). It is the main function that is called in your Python script. It takes no input and doesn't output anything.

The menu should keep asking the user to select an option (see options below) until he requests to quit (typing character "`e`"):

```
What do you want to do ?
e - exit
l - load a picture
s - save the current picture
1 - adjust brightness
2 - adjust contrast
3 - apply grayscale
4 - apply blur
5 - edge detection
6 - embossed
7 - rectangle select
8 - magic wand select

Your choice:
```

If he selects "`e`", the program will quit. If he selects "`l`", the function will ask the user to enter a filename and load an image from that file. If he selects "`s`", the function will ask the user to enter a filename and save the current image in that file. The other options will simply apply the corresponding filters, explained later in this document. Of course, every time one action is performed, the menu is displayed again so that the user can select a new action.

Note that all inputs inserted by the user must be error checked: even if the user inserts weird inputs, your program must not crash. It is therefore very important that you spend some time trying to crash your program and correct the bug if this happens.

Moreover, the current image array `img` and current selection mask array `mask` are set to empty at the start (using the command `np.array([])`). Thus, the user can't apply any filter or save a file if an image has not be loaded before (you can check the size of an image with `img.size`). Therefore, at the start (and as long as an image hasn't been loaded), the menu should actually look like this:

```
What do you want to do ?
e - exit
l - load a picture

Your choice:
```

2.2 2nd step: implementing the various picture transformations

The second step, once the interface with the user is ready, is to fill the functions that will apply the various image filters/transformations. There are six of them: `change_brightness`, `change_contrast`, `grayscale`, `blur_effect`, `edge_detection`, `embossed`. Note that for each of these functions, you have to work with a copy of the input image (`img = image.copy()`).

2.2.1 Changing the brightness.

In order to change the brightness of an image, you can simply add to each of the three RGB components an integer `value` between -255 and +255 included (+255 will increase a lot the brightness, while -255 will reduce it a lot). Remember that in RGB encoding, a component value can't be smaller than 0 or greater than 255. The input `value` will be queried to the user in the menu function (with error check). The function will return the new image.

2.2.2 Changing the contrast.

One method to change the contrast of the image to a factor `value` (between -255 and +255 included) is to compute the contrast correction factor $F = \frac{259*(\texttt{value}+255)}{255*(259-\texttt{value})}$. Then, the three color components for each pixel can be replaced by:

$$R' = F \times (R - 128) + 128$$
$$G' = F \times (G - 128) + 128$$
$$B' = F \times (B - 128) + 128$$

Remember that in RGB encoding, a component value can't be smaller than 0 or greater than 255. The input `value` will be queried to the user in the menu function (with error check). The function will return the new image.

2.2.3 Grayscale effect.

Grayscale images are composed exclusively of shades of gray. In order to change an image to grayscale, one naive way would be to assign each of the three RGB components to their average: $(R+G+B)/3$. However, the contribution of red, green and blue component on the luminance of the image is not the same for the human vision. You can test it for your self, applying this formula will lead to an image that looks blacker than it should be. In order to keep the luminance stable, one should use a different weight for each color component. Instead, a better formula would then be to assign $(0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B)$ to each of the three RGB components for each pixel of the image. The function will return the new image.

2.2.4 Blur effect.

In order to blur a picture, a strategy is to compute for each pixel some weighted average between the pixel itself and its eight direct neighbours (left, right, top, bottom, and the four diagonals). This is called a convolution filter. Take the (3×3) matrix M defined by the pixel p and its eight neighbours in the image:

$$M = \begin{pmatrix} tl & t & tr \\ l & p & r \\ bl & b & br \end{pmatrix}$$

We will then use the following convolution matrix (or kernel) K, which corresponds to a Gaussian blur:

$$K = \begin{pmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{pmatrix}$$

For each pixel p of the image (actually each of its three color components independently), we will then obtain p', the new pixel in the corresponding image, by computing the sum of all the elements of the matrix convolution $M * K$:

$$\begin{aligned} p' &= \text{sum}(M * K) \\ &= \text{sum}(\begin{pmatrix} tl & t & tr \\ l & p & r \\ bl & b & br \end{pmatrix} * \begin{pmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{pmatrix}) \\ &= \text{sum}(\begin{pmatrix} 0.0625 \cdot tl & 0.125 \cdot t & 0.0625 \cdot tr \\ 0.125 \cdot l & 0.25 \cdot p & 0.125 \cdot r \\ 0.0625 \cdot bl & 0.125 \cdot b & 0.0625 \cdot br \end{pmatrix}) \\ &= 0.0625 \cdot tl + 0.125 \cdot t + 0.0625 \cdot tr + 0.125 \cdot l + 0.25 \cdot p + 0.125 \cdot r + 0.0625 \cdot bl + 0.125 \cdot b + 0.0625 \cdot br \end{aligned}$$

Note that the pixels located on a side of the picture will not be recomputed, as they have no neighbour on one of their side. The function will return the new image. For a stronger blur effect, you can apply this effect several times in a row.

2.2.5 Edge detection effect.

This filter will make the discontinuities in an image stand out. The technique is exactly the same as the blur effect, but instead the following kernel is used:

$$K = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

However, since the obtained picture will be very dark, we will add 128 to each component of each pixel after application of the kernel. Moreover, remember that in RGB encoding, a component value can't be smaller than 0 or greater than 255 (this is done after adding the 128 to each component). The function will return the new image.

2.2.6 Emboss effect.

This filter will create an emboss effect on your picture. Again, the technique is exactly the same as the blur effect, but instead the following kernel is used:

$$K = \begin{pmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

However, since the obtained picture will be very dark, we will add 128 to each component of each pixel after application of the kernel. Moreover, remember that in RGB encoding, a component value can't be smaller than 0 or greater than 255 (this is done after adding the 128 to each component). The function will return the new image.

2.3 3rd step: implementing the mask selection functions

The last step of the project is to implement the functions that will allow the user to select only a subpart of the picture on which to apply the various filters. For that, we will allow two methods.

2.3.1 Rectangle selection.

The first one, `rectangle_select`, is quite simple: given a top left pixel position `x` and a bottom right pixel position `y` as inputs, the selection mask will be updated to consist only of the pixels located in the rectangle defined by the points `x` and `y`. The new selection mask is then returned by the function.

Before calling the function `rectangle_select`, the `menu` function should ask the user (with error check) for two pixel positions `x` and `y`.

2.3.2 Magic wand selection.

The second one, `magic_wand_select`, is a bit more complex: given the image `image`, a pixel position `x` and a threshold value `thres` as inputs, the selection mask will consist of all the **connected** pixels (including `x`) that are at a color-distance at most `thres` from pixel `x`. There are many ways to define a color distance with RGB values between two pixels, but you can use the following formula (see `https://www.compuphase.com/cmetric.htm` for more details about this formula):

$$dist = \sqrt{(2 + r/256) \times \Delta_R^2 + 4 \times \Delta_G^2 + (2 + (255 - r)/256) \times \Delta_B^2}$$

where Δ_R, Δ_G, Δ_B represent the difference between the two pixels in the red/green/blue components respectively and where r is the average of the red component of the two pixels.

The algorithm to build the set of selected pixels is known as *flood fill*: starting from the original pixel `x`, check the four direct neighbours from `x` (left, right, up, down) and add them into the mask selection if their color-distance from `x` is lower or equal to the threshold `thres`. Then, for each newly added pixel, check again their neighbours (no need to check pixels that have already been checked) and so on ... until no more pixel remains to be checked (hint: you should use a list as a stack to implement this algorithm, see an example here `https://www.educative.io/edpresso/how-to-implement-stack-in-python`). With the proposed formula, a typical value of threshold would be around 200. The new selection mask is then returned by the function.

Before calling the function `magic_wand_select`, the `menu` function should ask the user (with error check) for a pixel position `x` and a threshold value `thres`.

2.3.3 Applying effects on masked pixels only.

Finally, instead of applying the various filters on the entire image, modify your code in `menu` function so that only the selected pixels (according to the selection mask) are indeed modified every time. For that, you can simply obtain the new image from the filter/effect output, and combine the new and old images according to the mask.

3 If you want to go further ...

This section proposes some possible extensions for your project. Note that these extensions are here if you would like to get extra challenges for your project, but they will not participate to the final grade.

- more filters (sepia, color filters, etc.) !
- a nicer graphical interface, maybe with a mouse control ?
- possibility to undo an action
- try to optimize your code, so that the filtering runs faster (try to leverage vectorization capability of Numpy arrays)
- ...