**Python image manipulation to mimic the implementation of image filters**

This is a project I worked on approximately a month ago, I had taken a module in Image

processing for my undergraduate degree and was interested in putting some of that knowledge into practice. I then had a look online for interesting python projects that

implemented image processing techniques. I had found source code for the png.py

and image.py files which was of great help and used a combination of guided

tutorials and StackOverflow research to create the transform.py file. In this

code, I use Python to adjust the brightness and contrast, add blur, and detect

the edges of an image.

Here, you will find the following files:

- **png.py**: Python PNG `*Reader`* and *`Writer`* classes from Johann C. Rocholl

- **image.py:** This contains the `*Image*` class that reads and writes the images using the PNG `*Writer`* and `*Reader`*

- **transform.py**: Implemented image filter functions

- **Input folder**: Here I have saved two image files ‘lake.png’ and ‘city.png’, these will be inputs/original images

- **output folder**: Once the images have been manipulated they will be saved in the output folder

- **project context word document (this document**): this give you a brief overview, walks you through the code and highlights limitations and improvements

**png.py**

**This file contains code written by Johann C. Rocholl. Its purpose is a PNG reader and writer**

**This means the writer is a PNG encoder in python and the reader is a PNG decoder in python. Therefore, it takes a PNG image and decodes it into a python array for the reader and vice versa, it takes a python array and writes it to a PNG file for the writer.**

**Image.py**

This image class works to check if the user has input either a filename or values for the x pixels, y pixels, and the number of channels.

**\*When working with images we work with 3 RGB channels red, green, and blue, this will be represented as num\_channels = 3**

**Therefore, this class creates a 3D array with x pixels, y pixels and number of channels**

Read\_image function: This function takes in the image and encodes/decodes it using gamma. This ensures the operations are not exactly linear.

Write\_image function: This function call writes the image to a PNG file and then clips it between 0 and 1. It then reshapes the PNG file and writes it to the output file. The array is then resized to accurately represent the original input image without signs of the manipulations made to the image/array.

**Transform.py**

This file holds functions that will execute the image manipulations, including:

brighten function:

This increases each channel in every pixel by an input factor amount. Depending on how much you want to brighten the image. factor < 1 to darken and factor > 1 to brighten. Then using NumPy, the entire array is multiplied by this factor. The following vectorized line can be written in a more intuitive (non-vectorized) way to show the iteration process for every pixel.

#(vectorized)

new\_image = image.array\*factor

#(non-vectorized)

for x in range(x\_pixels):

for y in range(y\_pixels):

for c in range(num\_channels):

new\_im.array[x,y,c] =(image.array[x,y,c]\*factor)

adjust\_contrast function:

This adjusts the contrast by increasing the difference from the user-defined midpoint (mid) by a factor amount. Like the brightening function, I use a vectorized version to make the changes to the array.

#(vectorized)

new\_image = (image.array – mid)\*factor + mid

#(non-vectorized)

for x in range(x\_pixels):

for y in range(y\_pixels):

for c in range(num\_channels):

new\_im.array[x,y,c] =(image.array[x,y,c] – mid)\*factor+mid

*blur function:*

In image processing, a kernel (also known as convolution matrix or mask) represents a matrix that is moved over the image to perform convolution of this kernel matrix and the image data. It can also be thought of as a filter, It is used for blurring, edge detection, and more.

The link below is a short youtube video on how the kernel works.

<https://www.youtube.com/watch?v=C_zFhWdM4ic&ab_channel=Computerphile>

Image\*kernel = result image

Application, Teams

Description automatically generated

*Figure from Hands-on image processing with python by Sandipan Dey*

The blur function will take a kernel with a size specified by the user to create a blurred image.

apply\_kernel function:

This uses the Sobel method/filter to detect edges in an image. It is a method that looks for strong changes in the gradient of an image. Please watch the video linked below for a good explanation of how this works.

[*https://www.youtube.com/watch?v=uihBwtPIBxM&ab\_channel=Computerphile*](https://www.youtube.com/watch?v=uihBwtPIBxM&ab_channel=Computerphile)

*A pair of 3x3 convolutions masks are used to detect gradient changes in the x-direction and y-direction. The masks are shown below*

Diagram, table

Description automatically generated

*I have shown these in the transform.py file as below*

sobel\_x = apply\_kernel(city, np.array([  
 [-1, 0, 1],  
 [-2, 0, 2],  
 [-1, 0, 1]]))

sobel\_y = apply\_kernel(city, np.array([  
 [1, 2, 1],  
 [0, 0, 0],  
 [-1, -2, -1]]))

combine\_images function:

once the two edge detected images are made, this function combines them using the squared sum of squares.

From line 77 onwards of the transform.py file I have called the functions to play around with the different manipulation methods, you can do this yourself by commenting/uncommenting them. Once run the output images will be saved in the output folder. A pair of 3x3 convolutions masks are used to detect gradient changes in the x-direction and y-direction. The masks are shown below

**Limitations**

Though the code works fine on the two images I have provided, I am still trying to figure out why the code doesn’t process some other image formats. This will be something I need to do more research in, especially as I am using Johann C. Rocholl’s code, there could be an issue with image sizes/formats that do not work.

**Improvements**

I have used some repetition for iterating through the image array for each function/filter, and it looks quite long-winded. I would like to make these parts more efficient by using a more syntactic sugar method to write my code.