Pixel Art Project:

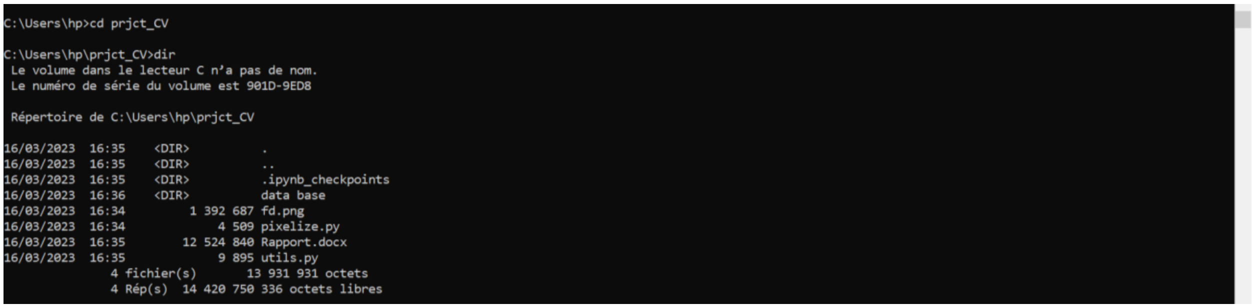
By: CHEHABI Mohammed

How to use the program:

To begin you need to put your data base images in the same folder, you can put all type of images the program can filter other extentions that PNG, JPG JPEG, and also can convert NON-RGB images to RGB.

Now we put our python file programs (**pixelize.py** and **utils.py**) in the same folder, then we open the CMD and navigate to our folder.

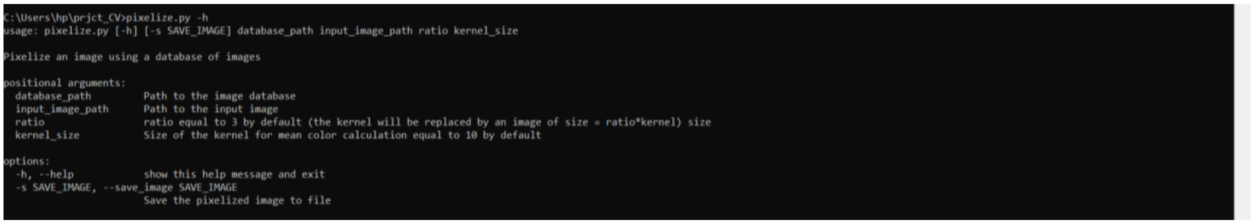
This image show the files in my folder, it is not mandatory to put the input image or the data_base with the other files.



```
C:\Users\hp>cd prjct_CV

C:\Users\hp\prjct_CV>dir
 Le volume dans le lecteur C n’a pas de nom.
 Le numéro de série du volume est 901D-9ED8

 Répertoire de C:\Users\hp\prjct_CV

16/03/2023  16:35    <DIR>          .
16/03/2023  16:35    <DIR>          ..
16/03/2023  16:35    <DIR>          .ipynb_checkpoints
16/03/2023  16:36    <DIR>          data base
16/03/2023  16:34         1 392 687 fd.png
16/03/2023  16:34             4 509 pixelize.py
16/03/2023  16:35        12 524 840 Rapport.docx
16/03/2023  16:35             9 895 utils.py
               4 fichier(s)       13 931 931 octets
               4 Rép(s)  14 420 750 336 octets libres
```

We can write the following command to get help.

>>pixelize.py -h

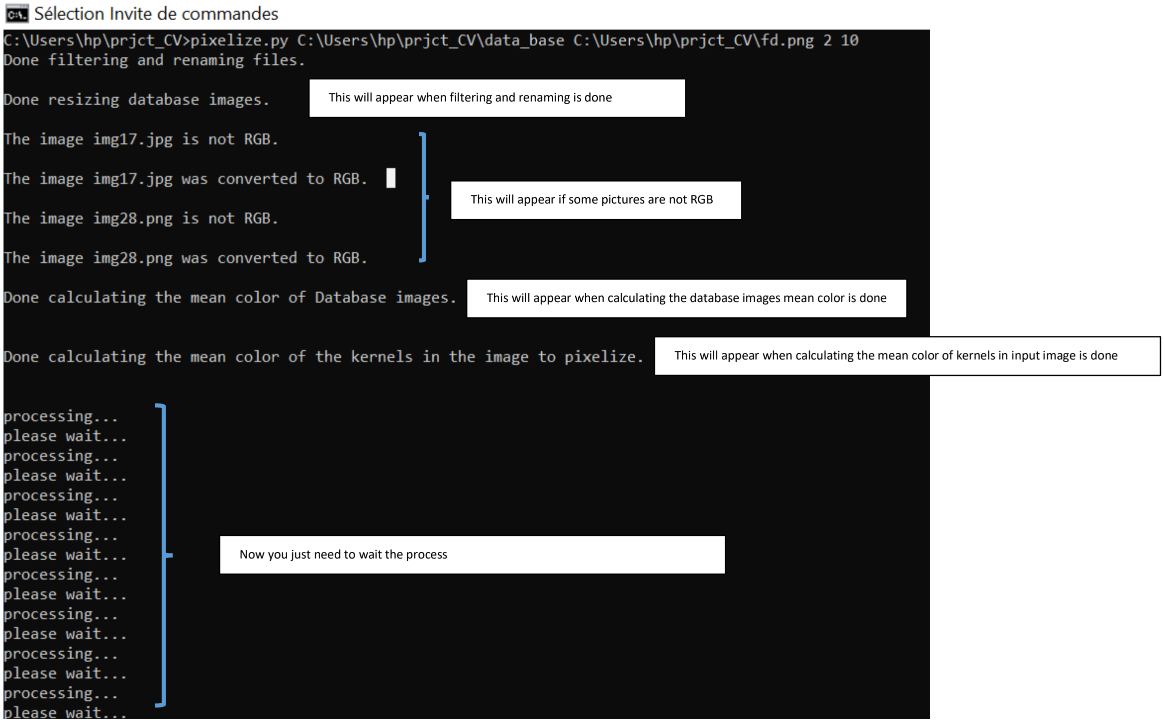


```
C:\Users\hp\prjct_CV>pixelize.py -h
usage: pixelize.py [-h] [-s SAVE_IMAGE] database_path input_image_path ratio kernel_size

Pixelize an image using a database of images

positional arguments:
  database_path        Path to the image database
  input_image_path     Path to the input image
  ratio                ratio equal to 3 by default (the kernel will be replaced by an image of size = ratio*kernel) size
  kernel_size          Size of the kernel for mean color calculation equal to 10 by default

options:
  -h, --help           show this help message and exit
  -s SAVE_IMAGE, --save_image SAVE_IMAGE
                       Save the pixelized image to file
```

It is mandatory to type the command as follow to execute the program:

>> pixelize.py data_base_path image to pixelize_path ratio kernel_size

After this command the image will appear at the end and won’t be saved



```
C:\Users\hp\prjct_CV>pixelize.py C:\Users\hp\prjct_CV\data_base C:\Users\hp\prjct_CV\fd.png 2 10
Done filtering and renaming files.

Done resizing database images.

The image img17.jpg is not RGB.

The image img17.jpg was converted to RGB.

The image img28.png is not RGB.

The image img28.png was converted to RGB.

Done calculating the mean color of Database images.


Done calculating the mean color of the kernels in the image to pixelize.


processing...
please wait...
processing...
please wait...
processing...
please wait...
processing...
please wait...
processing...
please wait...
processing...
please wait...
processing...
please wait...
processing...
please wait...
```

This will appear when filtering and renaming is done

This will appear if some pictures are not RGB

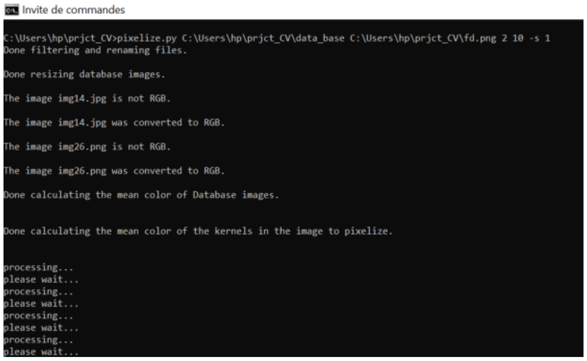
This will appear when calculating the database images mean color is done

This will appear when calculating the mean color of kernels in input image is done

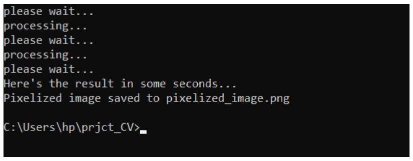
Now you just need to wait the process

If you want to save the image you need to add one optional argument (-s 1) that will save the image at the same folder of the program.

>> pixelize.py data_base_path image to pixelize_path ratio kernel_size –s 1



```
C:\Users\hp\prjct_CV>pixelize.py C:\Users\hp\prjct_CV\data_base C:\Users\hp\prjct_CV\fd.png 2 10 -s 1
Done filtering and renaming files.

Done resizing database images.

The image img14.jpg is not RGB.

The image img14.jpg was converted to RGB.

The image img26.png is not RGB.

The image img26.png was converted to RGB.

Done calculating the mean color of Database images.


Done calculating the mean color of the kernels in the image to pixelize.


processing...
please wait...
processing...
please wait...
processing...
please wait...
processing...
please wait...
```



```
please wait...
processing...
please wait...
processing...
please wait...
Here's the result in some seconds...
Pixelized image saved to pixelized_image.png

C:\Users\hp\prjct_CV>_
```

The image will be saved.

Functions used in the code:

verify_PNG_JPG(folder_path)

This function takes the path of a folder as input and checks that the images it contains have the extensions .png, .jpg or .jpeg. If an image does not have one of these extensions, it is deleted. If it has one of these extensions, it is renamed with the prefix "img" followed by a sequential number. The images are thus renamed so that there are no duplicates in the names. If the folder does not contain any images with one of the mentioned extensions, no operation is performed.

resize_images(folder_path, finale_size, output_dir)

This function takes the path of a folder, the desired final size for the images, and the path of the output folder as input. It resizes all the images in the input folder to the desired final size while maintaining the height/width ratio. If an output folder is specified, the resized images are saved in this folder. Otherwise, they are saved in the input folder by overwriting the original images.

convert_to_RGB(folder_path)

This function takes the path of a folder as input and checks that the images it contains are in the RGB format. If an image is not in the RGB format, it is converted to RGB and saved.

mean_color_DB_images(folder_path)

This function takes the path of a folder as input and returns a list of tuples containing the name of each image in the folder and the average value of its colors. The average value is calculated for each color channel (red, green, blue) of each image using the numpy library.

mean_color_pixels_image(image_path, kernel_size)

This function takes the path of an image and the size of the kernel as input. It divides the image into square regions of size kernel_size and calculates the average color value of each region. It returns an image where each region is replaced by a pixel with the average value color of that region.

color_distance(color1, color2, image_name, temporary_folder)

This function calculates the Euclidean distance between two colors. The two colors are represented as a triplet of RGB values. This function is used to calculate the distance between the average color of an image and a given reference color. The path of this temporary folder must be passed as a parameter, in order to delete it if there’s an error.

The main code explanation:

The main code pixelizes an input image by finding the closest match in a database of pre-existing images saved in a data base folder. The pixelization is performed by dividing the input image into small regions and replacing each region with the closest matching image from the database, resized depending on the ratio chosen, dataset image size = ratio*kernel_size.

It means that if size kernel=10 and ratio=4, we will replace every 10 pixel of the image by a 40 pixel image from our database!!

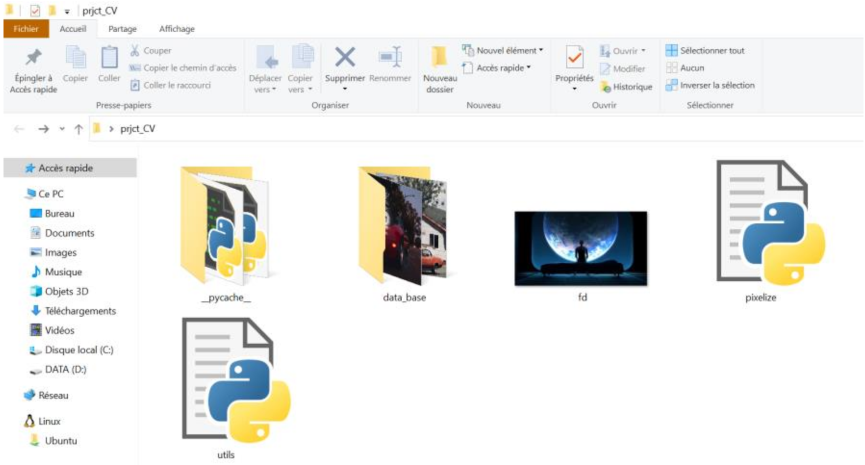


Figure 1: files used in my code.

The first part of the code sets up the input and output paths, as well as parameters such as the ratio of the output image size to the kernel size used to calculate the mean color of each region in the input image.

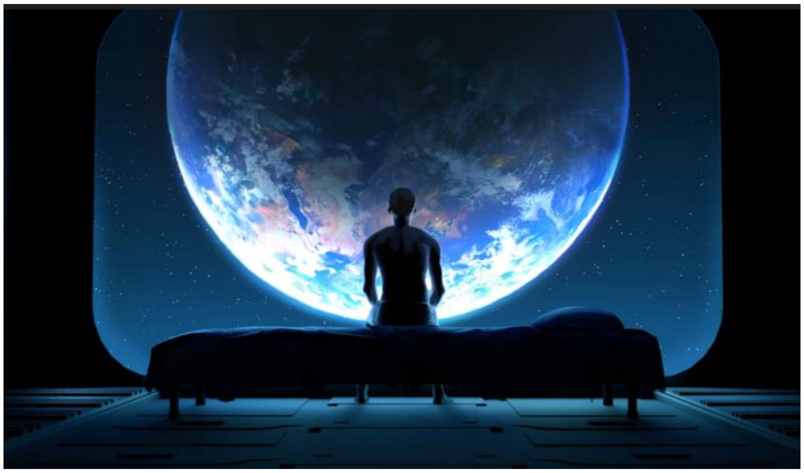
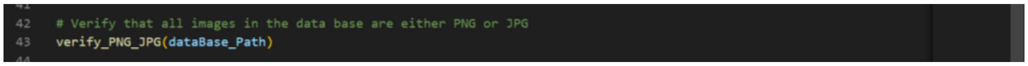


Figure 2: the input image

Next, the code calls a function `verify_PNG_JPG` to ensure that all images in the database are either PNG or JPG or JPEG and also to renamed images with the prefix "img" followed by a sequential number this.



```python
# Verify that all images in the data base are either PNG or JPG
verify_PNG_JPG(dataBase_Path)
```

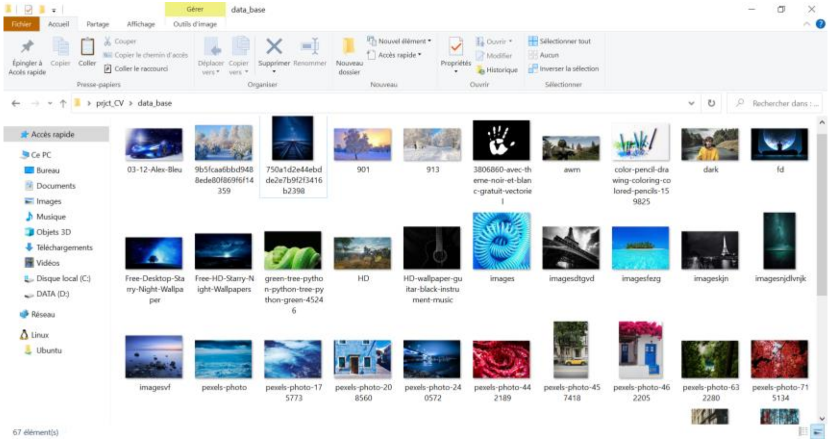

data_base
prjct_CV > data_base
03-12-Alex-Bleu
9b5fcaa6bbd9488ede80f869f6f14359
750a1d2e44ebdde2e7b9f2f3416b2398
901
913
3806860-avec-theme-noir-et-blanc-gratuit-vectoriel
awm
color-pencil-drawing-coloring-colored-pencils-159825
dark
fd
Free-Desktop-Starry-Night-Wallpaper
Free-HD-Starry-Night-Wallpapers
green-tree-python-python-tree-python-green-45246
HD
HD-wallpaper-guitar-black-instrument-music
images
imagesdtgvd
imagesfezg
imageskjn
imagesnjdlvnjk
imagesvf
pexels-photo
pexels-photo-175773
pexels-photo-208560
pexels-photo-240572
pexels-photo-442189
pexels-photo-457418
pexels-photo-462205
pexels-photo-632280
pexels-photo-715134
67 élément(s)

Figure 3: data base folder with strange names before the code

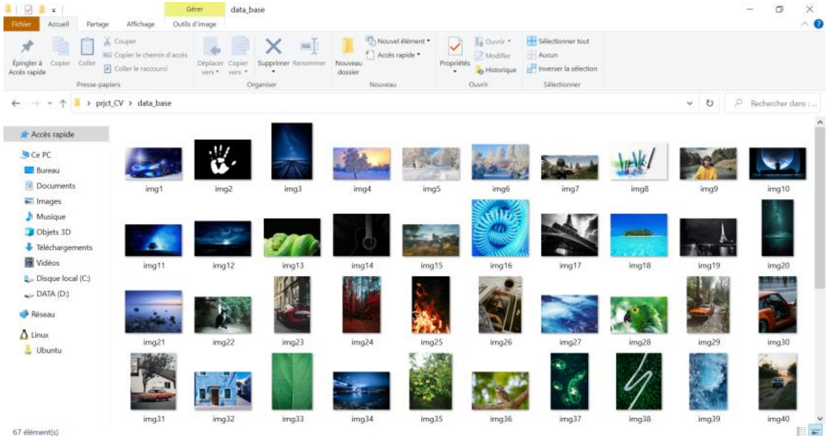
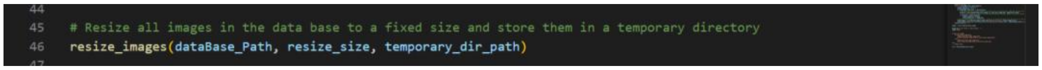


Figure 4: data base after filtering and renaming

And another function `resize_images` to resize all images in the database to a fixed size and store them in a temporary directory.



```python
# Resize all images in the data base to a fixed size and store them in a temporary directory
resize_images(dataBase_Path, resize_size, temporary_dir_path)
```

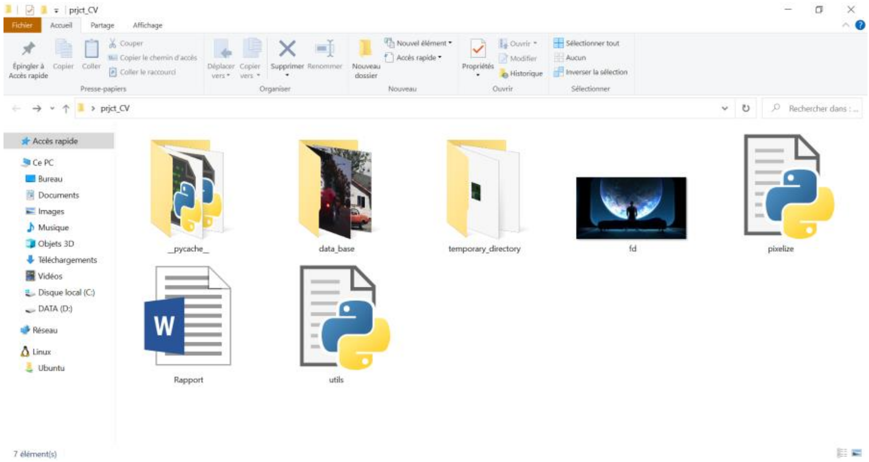


Figure 5: the temporary folder created by the program.

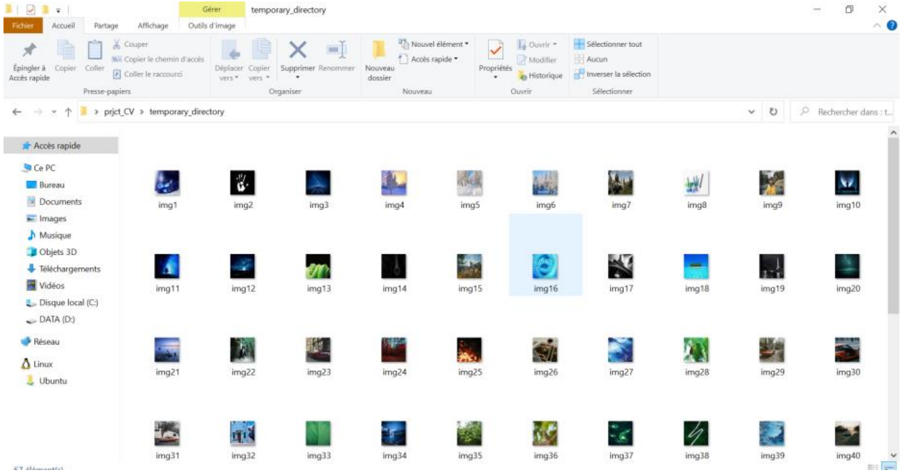
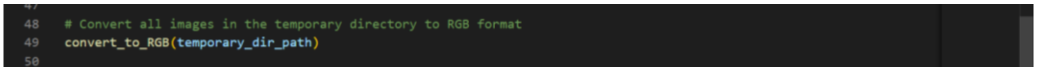


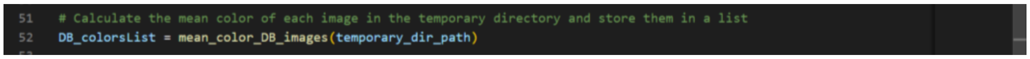
Figure 6: resized images stored in the temporary folder.

Then, the code calls a function `convert_to_RGB` to convert all images in the temporary directory to RGB format.



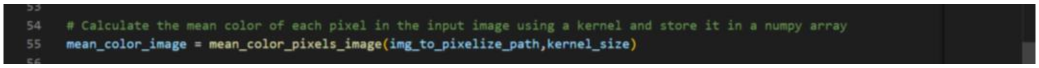
```python
# Convert all images in the temporary directory to RGB format
convert_to_RGB(temporary_dir_path)
```

And another function `mean_color_DB_images` to calculate the mean color of each image in the temporary directory and store them in a list.



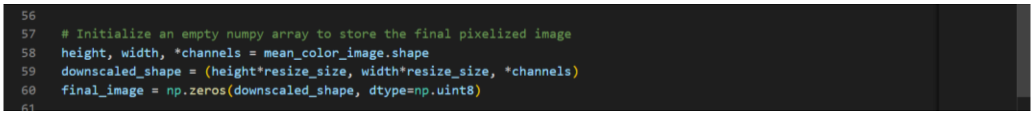
```python
# Calculate the mean color of each image in the temporary directory and store them in a list
DB_colorsList = mean_color_DB_images(temporary_dir_path)
```

Next, the code calculates the mean color of each pixel in the input image using a kernel chosen by the user by using the function `mean_color_pixels_image` and replace every kernel by a pixel with the RGB value of the mean color of every pixel of that kernel and return the resulting image that we will call **mean_color_image**, and now we just need to match every pixel with the corresponding image in the data base.



```python
# Calculate the mean color of each pixel in the input image using a kernel and store it in a numpy array
mean_color_image = mean_color_pixels_image(img_to_pixelize_path,kernel_size)
```

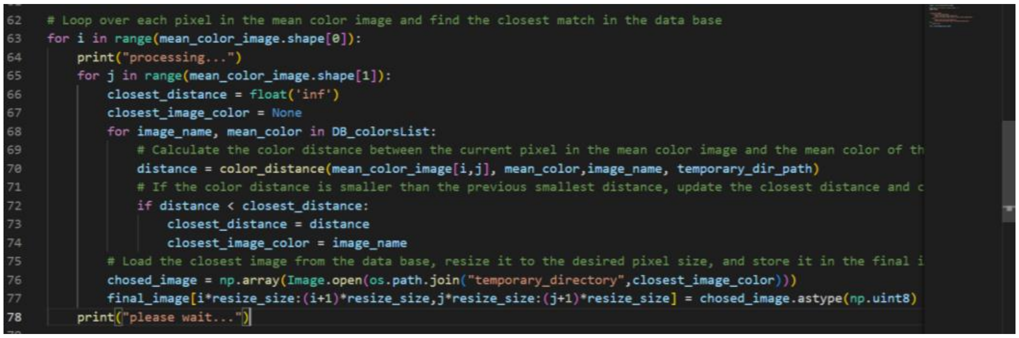
We initializes an empty numpy array with a size equal to:



```python

# Initialize an empty numpy array to store the final pixelized image
height, width, *channels = mean_color_image.shape
downscaled_shape = (height*resize_size, width*resize_size, *channels)
final_image = np.zeros(downscaled_shape, dtype=np.uint8)
```

Where we will store the final pixelized image after calculations.

Finally, the code loops over each pixel in the **mean_color_image**, finds the closest match in the database, loads the chosen image, and stores it in the final image array. The progress of the pixelization process is printed to the console.



```python
# Loop over each pixel in the mean color image and find the closest match in the data base
for i in range(mean_color_image.shape[0]):
    print("processing...")
    for j in range(mean_color_image.shape[1]):
        closest_distance = float('inf')
        closest_image_color = None
        for image_name, mean_color in DB_colorsList:
            # Calculate the color distance between the current pixel in the mean color image and the mean color of th
            distance = color_distance(mean_color_image[i,j], mean_color,image_name, temporary_dir_path)
            # If the color distance is smaller than the previous smallest distance, update the closest distance and c
            if distance < closest_distance:
                closest_distance = distance
                closest_image_color = image_name
        # Load the closest image from the data base, resize it to the desired pixel size, and store it in the final i
        chosed_image = np.array(Image.open(os.path.join("temporary_directory",closest_image_color)))
        final_image[i*resize_size:(i+1)*resize_size,j*resize_size:(j+1)*resize_size] = chosed_image.astype(np.uint8)
    print("please wait...")
```

The results:

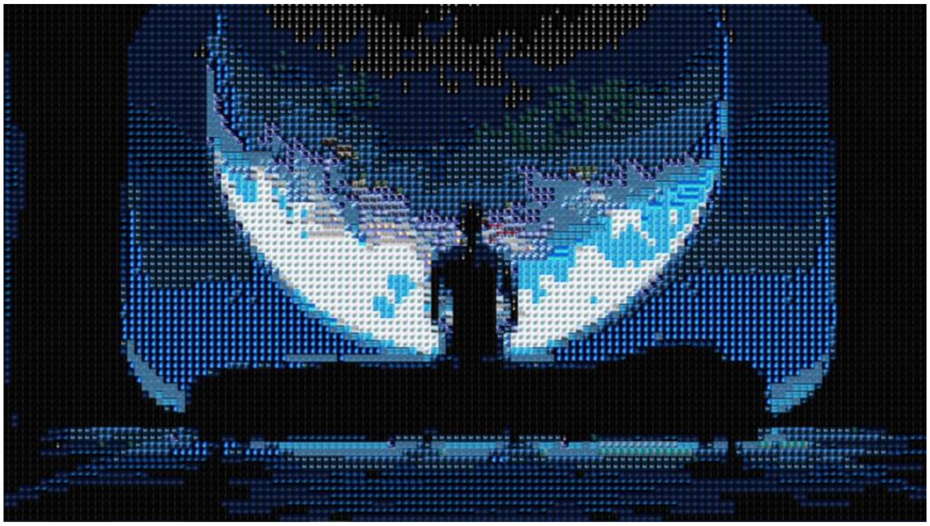


Figure 7: this is the resulting image.

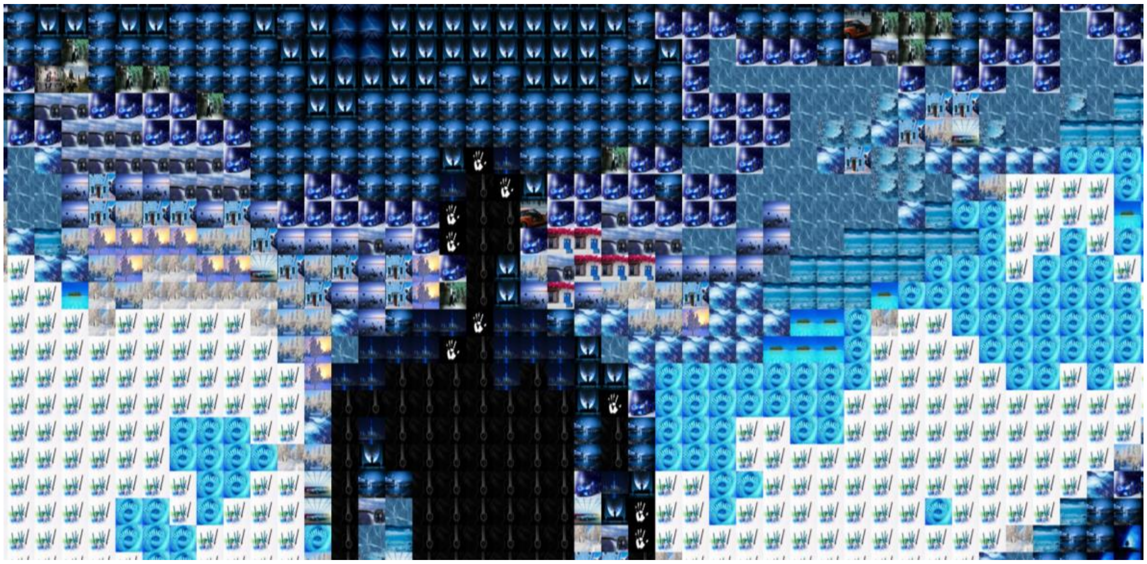


Figure 8: let's zoom in!

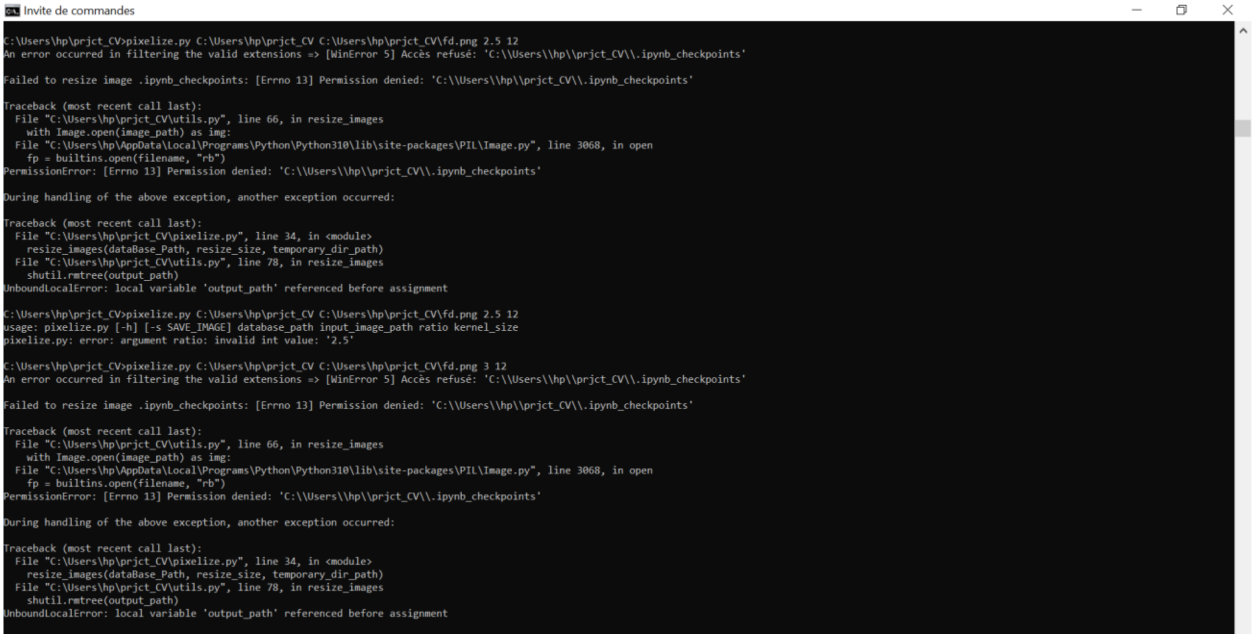
Some problems I got:

Deleting folders: it wasn't possible to delete a folder with files inside using os.remove() so I did solve this by using shutil.rmtree()

Problem with NON-RGB images: I got errors because I couldn’t calculate the mean color because some images wasn’t RGB, it took me some time to understand the error, at the end I created a function to check non-RGB images and convert them to RGB.

The temporary_directory: everytime I got an error I had to go delete the temporary_directory and sometimes I forget and get errors, that’s why in the code I delete that folder after every error and check in the beginning of the code if this directory was deleted, if not it will be deleted.

Add parser to the code: this wasn’t easy, I tried to execute this on the terminal of visual studio but that never worked, I tried it on the CMD of my computer and sometimes get some errors that I don’t understand, here’s an example:



```
C:\Users\hp\prjct_CV>pixelize.py C:\Users\hp\prjct_CV C:\Users\hp\prjct_CV\fd.png 2.5 12
An error occurred in filtering the valid extensions => [WinError 5] Accès refusé: 'C:\\Users\\hp\\prjct_CV\\.ipynb_checkpoints'

Failed to resize image .ipynb_checkpoints: [Errno 13] Permission denied: 'C:\\Users\\hp\\prjct_CV\\.ipynb_checkpoints'

Traceback (most recent call last):
  File "C:\Users\hp\prjct_CV\utils.py", line 66, in resize_images
    with Image.open(image_path) as img:
  File "C:\Users\hp\AppData\Local\Programs\Python\Python310\lib\site-packages\PIL\Image.py", line 3068, in open
    fp = builtins.open(filename, "rb")
PermissionError: [Errno 13] Permission denied: 'C:\\Users\\hp\\prjct_CV\\.ipynb_checkpoints'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\hp\prjct_CV\pixelize.py", line 34, in <module>
    resize_images(dataBase_Path, resize_size, temporary_dir_path)
  File "C:\Users\hp\prjct_CV\utils.py", line 78, in resize_images
    shutil.rmtree(output_path)
UnboundLocalError: local variable 'output_path' referenced before assignment

C:\Users\hp\prjct_CV>pixelize.py C:\Users\hp\prjct_CV C:\Users\hp\prjct_CV\fd.png 2.5 12
usage: pixelize.py [-h] [-s SAVE_IMAGE] database_path input_image_path ratio kernel_size
pixelize.py: error: argument ratio: invalid int value: '2.5'

C:\Users\hp\prjct_CV>pixelize.py C:\Users\hp\prjct_CV C:\Users\hp\prjct_CV\fd.png 3 12
An error occurred in filtering the valid extensions => [WinError 5] Accès refusé: 'C:\\Users\\hp\\prjct_CV\\.ipynb_checkpoints'

Failed to resize image .ipynb_checkpoints: [Errno 13] Permission denied: 'C:\\Users\\hp\\prjct_CV\\.ipynb_checkpoints'

Traceback (most recent call last):
  File "C:\Users\hp\prjct_CV\utils.py", line 66, in resize_images
    with Image.open(image_path) as img:
  File "C:\Users\hp\AppData\Local\Programs\Python\Python310\lib\site-packages\PIL\Image.py", line 3068, in open
    fp = builtins.open(filename, "rb")
PermissionError: [Errno 13] Permission denied: 'C:\\Users\\hp\\prjct_CV\\.ipynb_checkpoints'

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "C:\Users\hp\prjct_CV\pixelize.py", line 34, in <module>
    resize_images(dataBase_Path, resize_size, temporary_dir_path)
  File "C:\Users\hp\prjct_CV\utils.py", line 78, in resize_images
    shutil.rmtree(output_path)
UnboundLocalError: local variable 'output_path' referenced before assignment
```

The code was working before until I entered a float value and the code did crush and didn’t work again even with integer values, the code try to do changes on the .ipynb_checkpoint and consider it as an image!! I can’t see this file in my folder I only can see it from cmd if use the DIR command.

Finally I had to delete the folder and move to a new one and bring my codes to them, and now the code don’t support float values on Ratio☹.

And also a lot of problems with optional argument, when I tried to make the save output path as an optional argument, the code doesn’t work when I don’t write the path even that I put as optional!

Improvements:

All realized all the improvements I thought about, and don’t think there’s something to improve more! I thought about adding an argument for the path where the output will be saved but that will make too much arguments.

Also thought about a graphical interface, but the time will not be enough to do so.