

Image Processing – HW2

Submitters :

Maya Atwan , ID:314813494

Obaida Khateeb , ID: 201278066

Problem 1 – Point Operations

For the first image:

Our expectation: the grayscale values in this image are very close to each other which reduces the visibility of details. By applying **histogram equalization** the entire grayscale range from 0 to 255 is utilized, that means we want to make each 2 neighbor colors in the image/histogram image as far as possible by utilizing the entire grayscale range, resulting in a more evenly distributed intensity histogram. this improves the image's contrast making the differences between gray levels noticeable and providing a clearer and higher quality result.

For the gamma method in this case it may brighten or darken the image but won't enhance the contrast and the details visibility as effectively as the histogram equalization.

For the brightness and contrast method, this method its not optimize the pixel intensity distribution like the histogram equalization because it makes the same changes on all the pixels and that will not lead to enhancing the image and the visibility details like the histogram equalization do.

For the second image:

The image is mostly dark with some regions being brighter. Our expectation is applying **gamma correction with $\gamma < 1$** will be most effective, since it stretches the intensity values in the darker range brightening shadows while compressing the brighter range preserving highlights. However using $\gamma > 1$ will darken the brighter regions which is not suitable for this case as it reduces the visibility of the details.

For the brightness and contrast: Since it scales intensities uniformly, it will not handle the uneven lighting in the image well and may overexpose bright areas.

For the histogram equalization method, it may over enhance bright regions and amplify noise in the dark regions making the overall image appear unbalanced.

For the third image:

This image is challenging because the sky is too bright but the land and the girl look fine. **Brightness and contrast method** wont help here because it changes the whole image uniformly which either make the sky too bright or darken the land and the girl unnecessarily. So we need to choose between gamma correction and histogram equalization.

Histogram equalization might enhance the overall contrast but it can introduce noise and make the image look unnatural.

For the **gamma correction** using $\gamma < 1$ won't help in this case as it would brighten the image further. so **the best choice would be a gamma correction with $\gamma > 1$** . in this case we can darken the sky while keeping the land and the girl natural without overexposing.

Problem 1 – part B:

For the first image we chose **histogram equalization**. The fixed image:



for the second image we chose **gamma correction and $\gamma = 0.5$** . the fixed image:



for the third image we choose **gamma correction and $\gamma = 1.15$** . the fixed image:



The implementation of the 3 methods:

Histogram equalization: we used the built in function `equalizeHist` in the `cv2` library.

```
return cv2.equalizeHist(image)
```

Brightness and contrast: using the formula $a * (pixel_{value} - m) + m + b$ where m is for mean and a controls the contrast and b controls the brightness. After that we clamped values to 0-255.

```
def brightness_and_contrast(image, a, b):  
    m = np.mean(image)  
    brightness_contrast = a * (image - m) + m + b  
    #making sure the values are in the range [0,255]  
    brightness_contrast[brightness_contrast < 0] = 0  
    brightness_contrast[brightness_contrast > 255] = 255  
    #rounding the values to the closest integer  
    brightness_contrast = np.round(brightness_contrast).astype(np.uint8)  
    return brightness_contrast
```

Gamma correction: using the formula $255 * \left(\frac{pixel\ val}{255}\right)^\gamma$.

```
def gamma_correction(image, gamma):
    return np.array(255 * (image / 255) ** gamma, dtype='uint8')
```

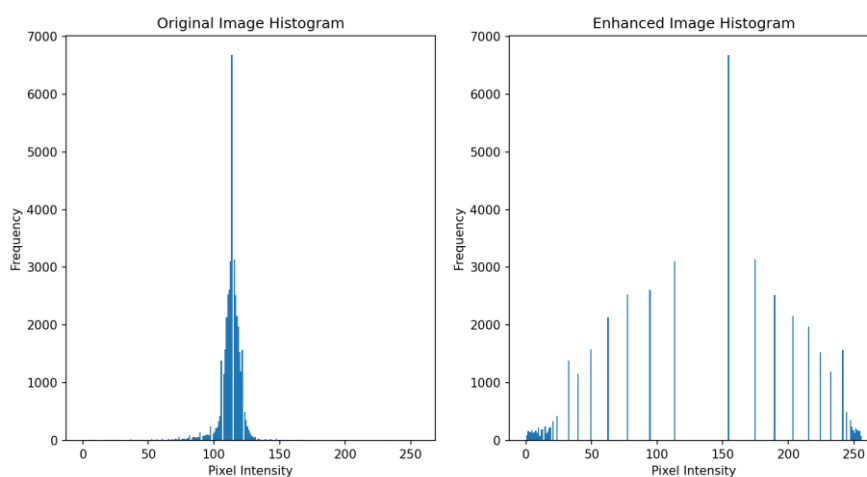
Review of the impact of each method:

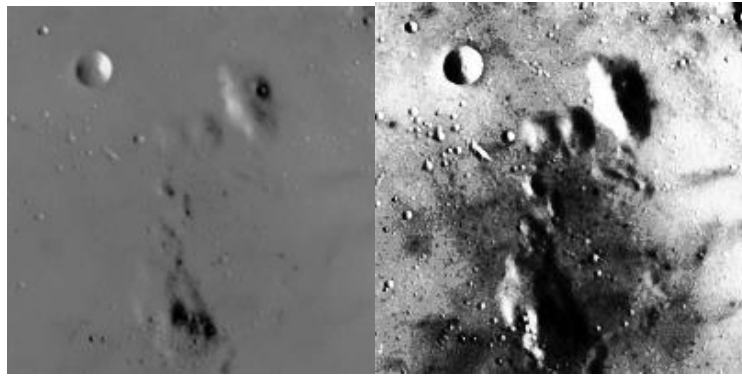
In this section, we applied the three methods on each of the images. The results confirms our previously mentioned expectations.

Image 1:

After running the below code to compare the histogram before and after applying the histogram equalization:

```
78
79 def plot_histograms(image, fixed_image):
80     plt.figure(figsize=(12, 6))
81     plt.subplot(1, 2, 1)
82     plt.hist(image.ravel(), 256, [0, 256])
83     plt.title('Original Image Histogram')
84     plt.xlabel('Pixel Intensity')
85     plt.ylabel('Frequency')
86     plt.subplot(1, 2, 2)
87     plt.hist(fixed_image.ravel(), 256, [0, 256])
88     plt.title('Enhanced Image Histogram')
89     plt.xlabel('Pixel Intensity')
90     plt.ylabel('Frequency')
```





Before, the original histogram is clustered and the image lacks contrast and details. After, post histogram equalization the grayscale values are spread across the entire range, and the image enhanced overall quality and details visibility.

So histogram equalization successfully improves the image as we expected, by spreading pixel values across the intensity range and the image now has better contrast and detail visibility.

Here are the results of applying gamma correction using the various gamma values:

gamma		gamma	
1		1.2	
0.5		1.5	
0.67		2	
0.8		2.2	

As we can see gamma values for both $\gamma < 1$ and $\gamma > 1$ the image is not enhanced.

The below are the results of applying Brightness and contrast correction using different alpha and beta values:

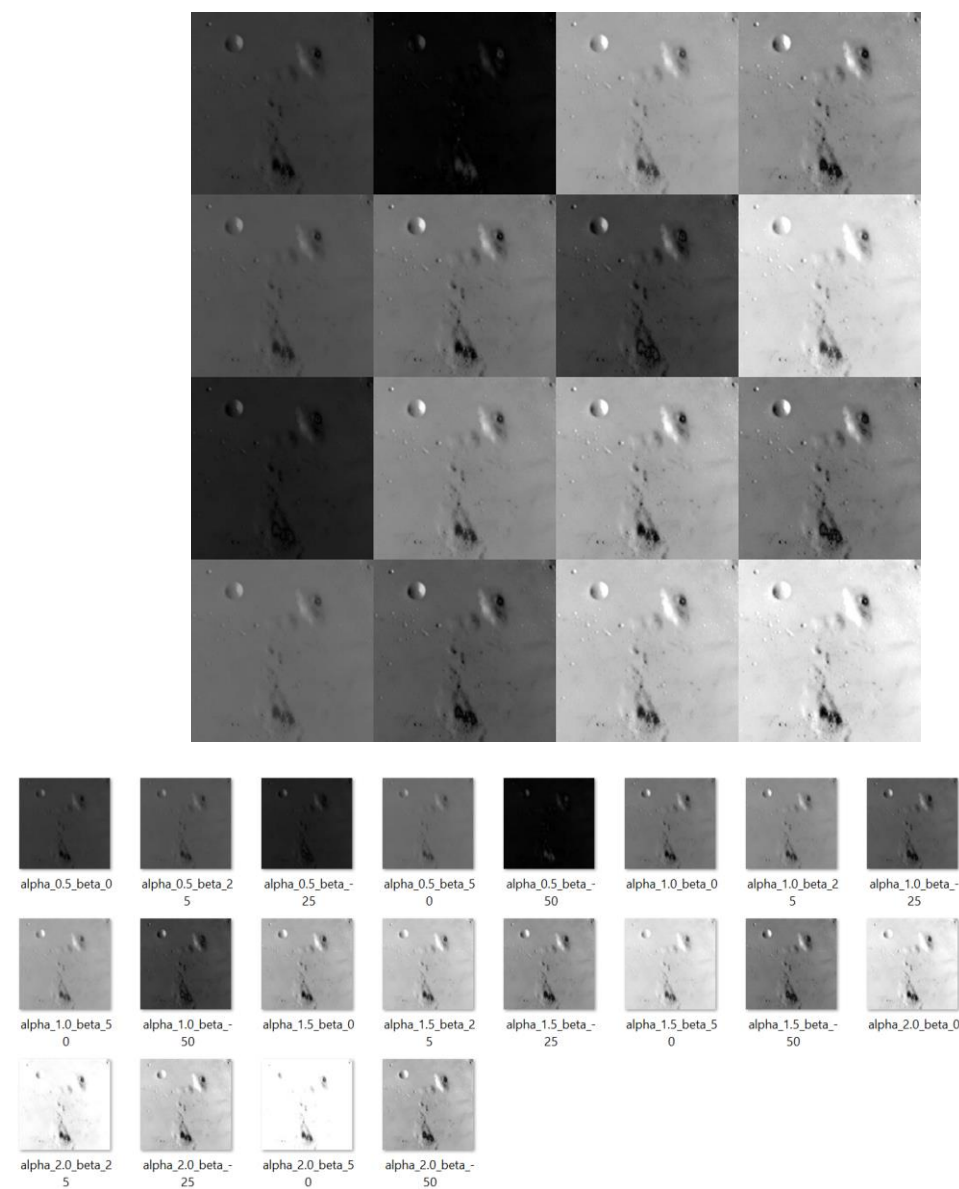










Image 2:

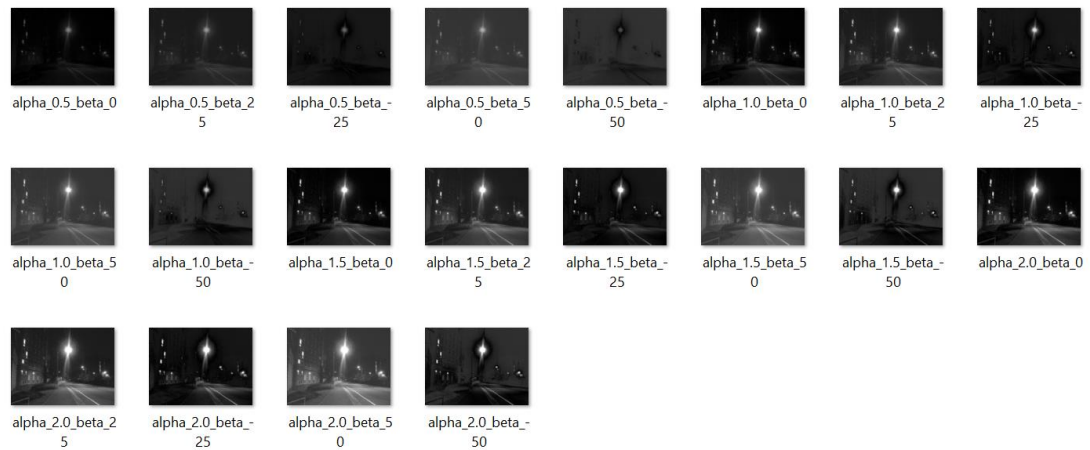
Here are the results using gamma correction with various gamma values:

gamma		gamma	
1		1.2	

0.5			1.5		
0.67			2		
0.8			2.2		

Here are the result after applying brightness and contrast method with different alpha and beta values:













Here is the result after applying histogram equalization:



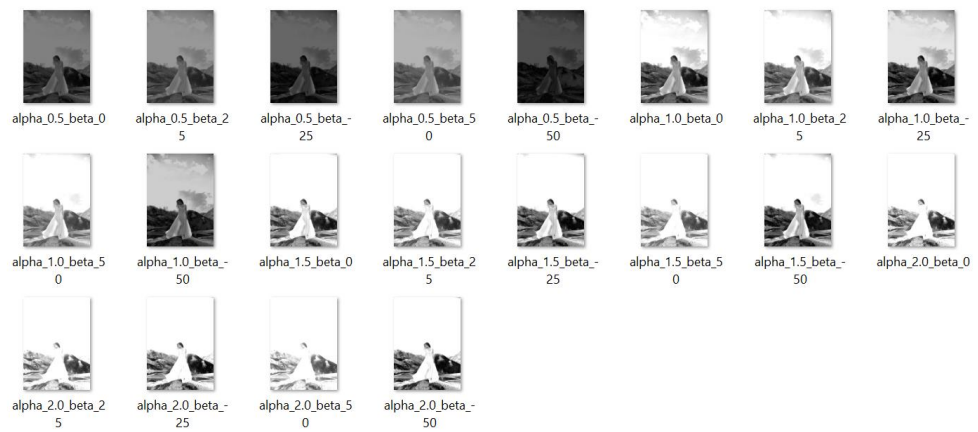
Image 3:

Here are the results of applying gamma correction with different gamma values:

gamma		gamma	
1		1.2	

0.5		1.5		
0.67		2		
0.8		2.2		

The below are the result of using brightness and contrast method with different alpha and beta values:



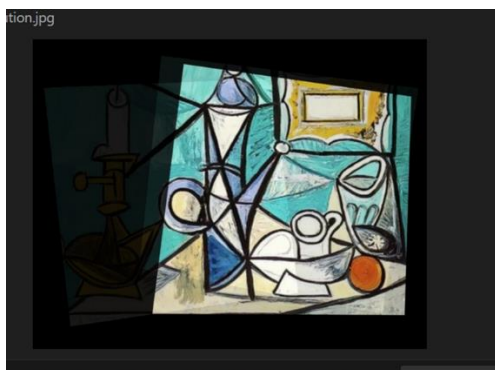
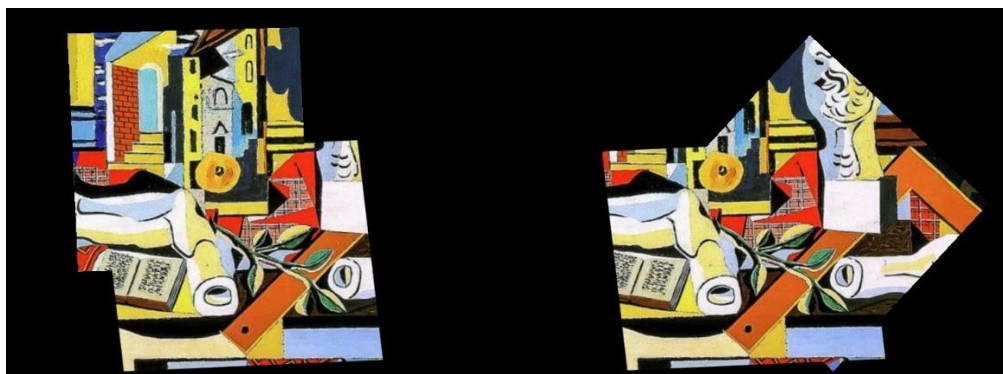
The below is the result got using histogram equlaization:



Problem 2 – Puzzle solving with geometric operations

A) this part was challenging, we used the website <https://pixspy.com/> which help us in detecting pixels coordinates.

For affine transformations we chose 3 pairs of coordinates and for projective we used 4 pairs. Some points didn't work well initially so we refined the matches. Each image was aligned individually with the first image which simplified detecting which image caused issues in the solution. The matching pixels were chosen based on them being special and relatively easy to locate and differentiate from the surrounding environment, e.g. pixels lie in a border where colors get changes, pixels that is part of a dark small dot surrounded by a bright area.



B) the call already implemented in main using the line: `matches, is_affine, n_images = prepare_puzzle(puzzle)` we ensured that by printing the returned values for each:

```
Starting puzzle_affine_1
is_affine: True
n_images: 2
matches 0:
[[[210  58]
  [281  80]]

 [[492 414]
  [ 17 275]]

 [[182 372]
  [ 36  80]]]
```

```
Starting puzzle_affine_2
is_affine: True
n_images: 5
matches 0:
[[[381 282]
  [174  94]]

 [[421 227]
  [180 147]]

 [[514 331]
  [ 78 136]]]
matches 1:
[[[449 210]
  [106 201]]

 [[382 252]
  [140 252]]

 [[494 400]
  [245 144]]]
matches 2:
[[[310 210]
  [203 105]]

 [[259 361]
  [168 230]]

 [[230 223]
  [136 128]]]
matches 3:
[[[230 306]
  [ 57 101]]

 [[283 262]
  [ 89 144]]

 [[146 227]
  [119  39]]]
```

```

Starting puzzle_homography_1
is_affine: False
n_images: 3
matches 0:
[[[592 415]
  [246  24]]

 [[477 144]
  [ 60 122]]

 [[314 339]
  [225 213]]

 [[374 116]
  [ 50 191]]]
matches 1:
[[[299 294]
  [178 164]]

 [[424 197]
  [283  99]]

 [[378 105]
  [255  26]]

 [[437 380]
  [276 240]]]

```

```

matches, is_affine, n_images = prepare_puzzle(puzzle)
#prints to ensure the correctness of the data returned by prepare_puzzle
print(f'is_affine: {is_affine}')
print(f'n_images: {n_images}')
for i in range(n_images-1):
    print(f'matches {i}:')
    print(matches[i])

```

c) The function 'get_transform' was implemented. It get as parameters a set of 3-4 pairs of coordinates, such that the first of each pair relate to one piece of puzzle, and the second to another piece. In addition, it receives 'is_affine' Boolean parameter, which indicates if the transformation is affine or projective. It return the appropriate transformation matrix using 'estimateAffine2D' and 'findHomography' cv2 methods.

```

def get_transform(matches, is_affine):
    src_points, dst_points = matches[:,0], matches[:,1]
    if is_affine:
        Transform, _ = cv2.estimateAffine2D(src_points, dst_points)
    else:
        Transform, _ = cv2.findHomography(src_points, dst_points)
    return Transform

```

D) The function 'inverse_transform_target_image' implemented such that it receives the target image, the transform matrix and the output size parameters. The function firstly checks if the transformation matrix is 2x3 size, this happens when the transformation is affine, in this case it extends it to 3x3 by adding a third [0, 0, 1] row. Then, it computes the inverse transformation matrix and apply it on the target image using 'cv2.warpPerspective' method, and return the result which is the inverse transformed image.

```
def inverse_transform_target_image(target_img, original_transform, output_size):  
  
    if original_transform.shape == (2, 3):  
        original_transform = np.vstack([original_transform, [0, 0, 1]])  
    inverse_transform = np.linalg.inv(original_transform)  
    warped_img = cv2.warpPerspective(target_img, inverse_transform, output_size,  
                                     borderMode=cv2.BORDER_TRANSPARENT)  
    return warped_img
```

E) the function 'stitch' merges 2 images by keeping the brightest pixel at each point using 'np.maximum' this creates a single image.

```
def stitch(img1, img2):  
    return np.maximum(img1, img2)
```

F)

image 1:



Image 2:



Image 3:

