CPS 843 Problem Set 1

Ritesh Ahlawat
Ryerson University
Toronto, Ontario
rahlawat@ryerson.ca

***Abstract*— Solutions for problem set 1.**

I. Problem 1

The formulas for log, inverse log, and power-law transformations are as follows:

$$s = c\log(1 + r) \tag{1}$$

$$s = clog^{-1}(r) \tag{2}$$

$$s = cr^{y} \tag{3}$$

Where, s and r are the pixel values of the output and input image and c is a constant. The log transform stretches low intensity levels and compresses high intensity levels, and the inverse log transform does the opposite. The power-law transform is more versatile and can either "act" as a log or inverse log transform, or even somewhere in between depending on the value of γ.

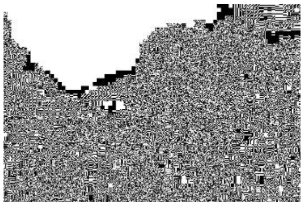
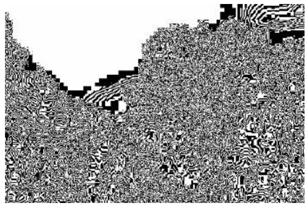
For demonstration, I apply the power-law transform to a generic image shown below:

a. *b.* *c.*

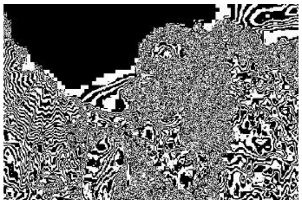
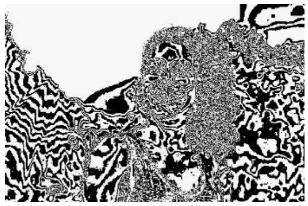
Figure 1: a) Before applying power-law transform b) power-law transform with $\gamma = 0.3$ c) power-law transform with $\gamma = 3$

As shown in Figure 1, a $\gamma < 1$ yields an effect similar to the log transform (a visually brighter image) and a $\gamma > 1$ yields an effect similar to the inverse log transform (a visually darker image).

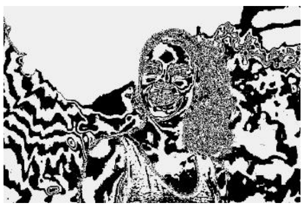
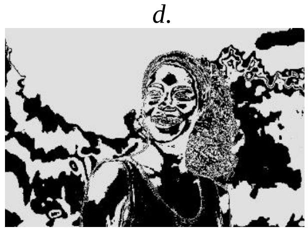
II. PROBLEM 2

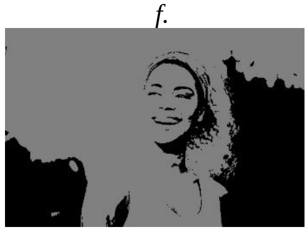
a. *b.*

c. *d.*

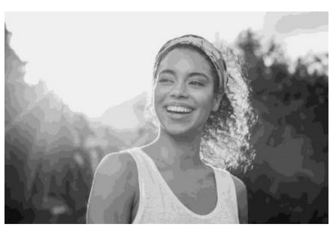
e. *f.*

g. *h.*

Figure 2: Images in a-h represent the 8-bit plane slicing results.

As shown in Figure 2, the lower bits are mostly filled with noise and have little to no relevant data.

a. *b.*

Figure 3: a) Reconstructed image from highest 2 bits b) Reconstructed image from highest 4 bits

As shown in Figure 3, The reconstructed image from the highest bits is contains the most relevant data to where the image from the 4^{th} highest bits is barely distinguishable from the original.

III. PROBLEM 3

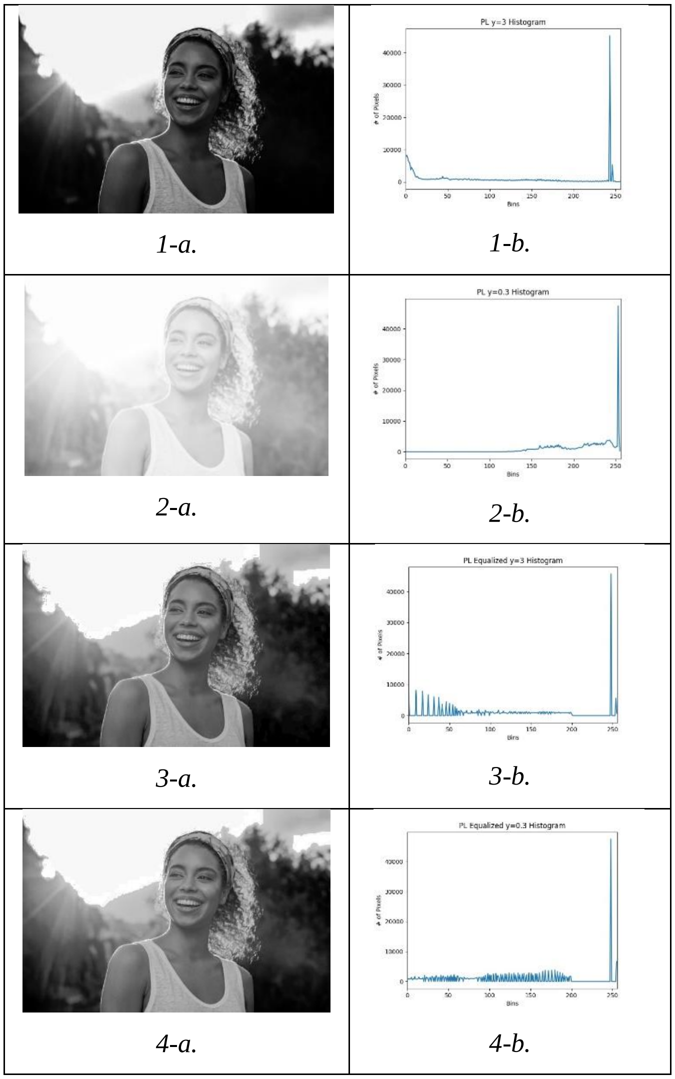


Figure 4: a) and b) represent an image and its respective histogram. 1 & 2) represent the original image with $\gamma = 3$ *and* $\gamma = 0.3$ *respectively. 3) represents 1) after histogram equalization. 4) represents 2) after histogram equalization*

As shown in Figure 4. The histogram of 1-a is more concentrated on the lower intensities whereas the histogram of 2-a is more concentrated on the higher intensities. This corresponds to how 1-a’s power-law transform is applied with a $\gamma > 1$ producing a darker image, and how 2-a’s power-law transform is applied with a $\gamma < 1$ producing a brighter image. After histogram normalization both images with high and low power-law transformations applied have a similar image after being equalized despite having different histograms (3-b & 4-b).

IV. PROBLEM 4

$$s = T(r) \tag{4}$$

$$s = G(z) \tag{5}$$

$$z = G^{-1}(s) \tag{6}$$

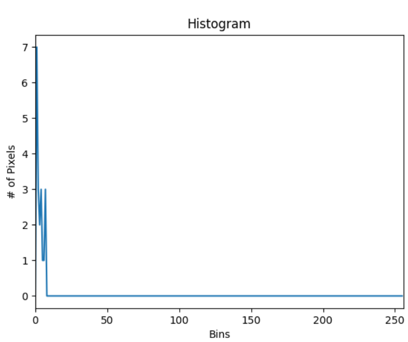
In histogram matching you first calculate the PDF that represents the original image, r. You use the PDF of the image to perform a histogram equalization on r (4). You then do the same for the target image z to get the target histogram (5). Lastly you perform the inverse mapping (6) to get the target transformation function f, where $z = f(r)$.

However, you will not obtain the same histogram as the desired one after histogram matching. This is because we are modifying r based on the distribution of z, it is not a 1-1 matching.

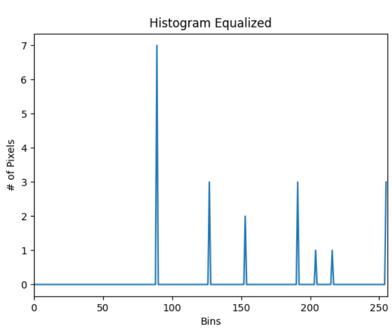
V. PROBLEM 5

The process for equalizing an image is to first get the normalized cumulative sum of the histogram. We then use this sum to generate a lookup table from which we then generate the equalized image. Since it's flattened to 1 dimension, we have to finally reshape it to its original dimensions.

```
def equalize_histogram(image: np.ndarray,
bits=8):
    num_of_vals = 2 ** bits
    # Get normalized cumulative histogram
    hist = np.bincount(image.flatten(),
minlength=num_of_vals)
    num_pix = np.sum(hist)
    hist = hist / num_pix
    cum_hist = np.cumsum(hist)

    # Pixel mapping lookup table
    transform = np.floor((num_of_vals-1)
* cum_hist)
    transform = transform.astype('uint8')

    # Transformation
    img_flattened = image.flatten()
    eq_img = np.array([transform[i] for i
in img_flattened])

    # Reshape to original dims
    return np.reshape(eq_img,
image.shape)
```



a.



b.

Figure 5: a) Histogram of the original image b) Histogram of equalized image

As shown in Figure 5, after being equalized, the histogram is more spread out through the 8 bins of the 3-bit image.

VI. PART 2

A. Step 1

a. *b.*

Figure 6: a) Original image b) Original image with shear transformation applied

As shown in Figure 6, a shear transformation is applied to an image and the background being filled in with an orange color.

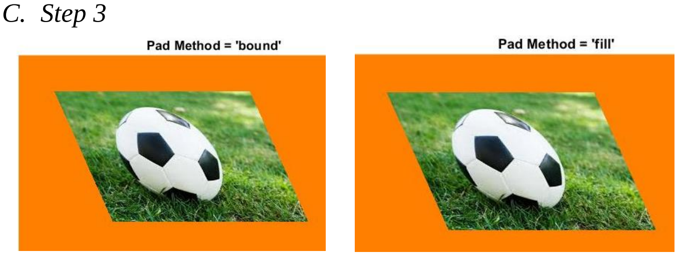
B. Step 2

a. *b.*

Figure 7: a) Original with grid and circles b) Sheared with grid and circles

In step 2, as shown in Figure 7, we visually can see how the shear transformation is applied by adding grids and circles to the image.

C. Step 3



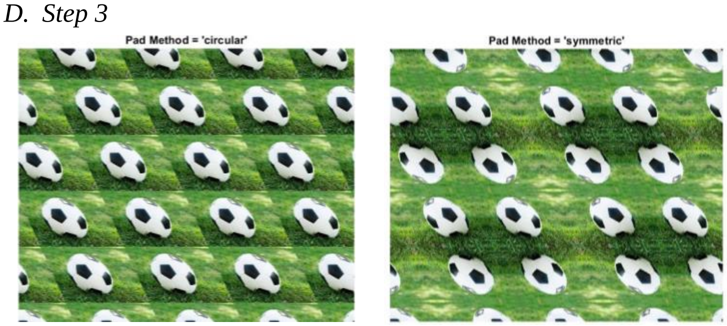
a. *b.*

c.

Figure 8: a) Bound padding method b) Fill padding method c) Replicate padding method

In step 3, as shown by Figure 8, we apply different padding methods to show their difference. It is to note that *'bound'* and *'fill'* both look similar but zooming close would show that edges are smoother with *'fill'* since cubic interpolation is applied across the edges in the *'fill'* method.

D. Step 3



a. *b.*

Figure 9: a) Circular padding method b) Symmetric padding method

In step 4, as shown in Figure 9, we apply the *'circular'* and *'symmetric'* padding methods. Where *'circular'* simply copies the sheared image in a circular fashion whereas the *'symmetric'* method mirrors the image before applying the *'circular'* padding.

VII. APPENDIX

All source code for this assignment can be found on GitHhub at **riteshahlawat/cps843-assignment-1**