CS-GY 6643 Computer Vision
Project Report 1

Runze Li
rl50xx@nyu.edu

February 19, 2024

1 Histogram Equalization

Implement histogram equalization and apply to image people.png, which you will find uploaded on NYU Brightspace. The programming environment supported by our TAs is Python. You are encouraged to use libraries/built in functions to read/and write images and display figures a nd g raphs, b ut t he r est o f t he i mplementation m ust be your own. That specifically i ncludes c reating a n i mage h istogram (probability d ensity f unction, P DF), computing the corresponding cumulative distribution function (CDF), and the creation of a new contrast adjusted image.

1.1 Include a brief introduction and description of how histogram equalization works.

Histogram equalization is a technique which is used to find t ransformation for c ontrast e nhancement. T his method is to create an image with equally distributed brightness levels over the whole brightness scale.

The steps of histogram equalization are as follows:

1. Compute the histogram of the input image. The histogram represents the frequency distribution A.1 of pixel intensities in the image and get normalized intensity histogram A.2 from an input image.
2. Calculate the cumulative distribution function (CDF) of the histogram A.3, which gives the cumulative sum of histogram values up to each intensity level.
3. Next, the intensity values of the input image are mapped to new intensity values using the computed CDF.
4. Finally, the equalized image A.4 is generated by applying the mapped intensity values to the corresponding pixels of the input image.

1.2 Show people.png before and after histogram equalization, and the corresponding histograms (PDFs).

Through histogram equalization we can get the following image and corresponding histogram 1.

1.3 Discuss how the image and histogram have changed

After applying histogram equalization to an image, the image and corresponding histogram change significantly.

For images, histogram equalization will improve contrast. Regions which are too dark or bright will exhibit a wider range of intensities, which will make details more visible. What's more, histogram equalization can spread out the intensity values evenly and gain a more balanced distribution of brightness levels in the image.

For histogram, some peaks will be flatten out after histogram equalization, and the histogram becomes more evenly distributed across the intensity range. The histogram will span the entire intensity range, indicating that the wider range of brightness levels will be utilized.

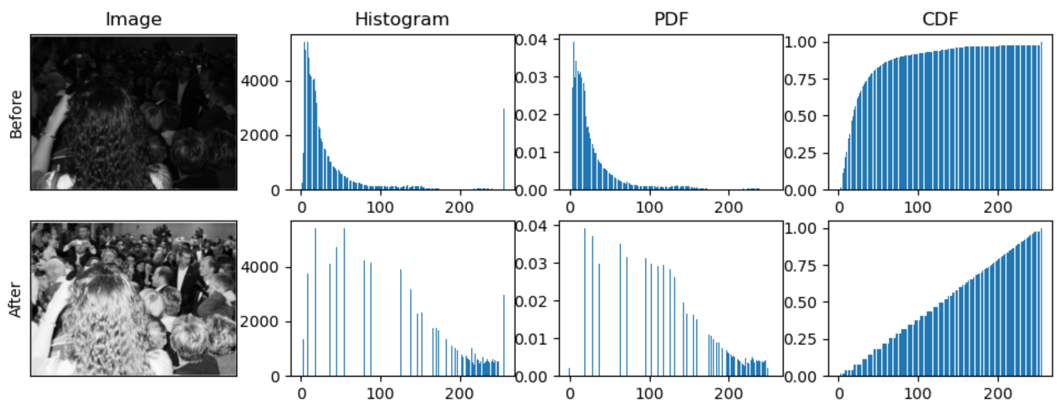

Figure 1: Image and corresponding histogram(PDF and CDF)

1.4 Show the cumulative distribution function before and after histogram equalization

Through histogram equalization we can get the cumulative distribution function 1 before and after histogram equalization.

After equalization, CDF will ideally become a straight line, which indicates that the intensity values have been evenly distributed across the range. Equalized CDF suggests that the full range of intensity values is being utilized, resulting in enhanced contrast.

1.5 Reapply the histogram equalization procedure on the corrected image

After reapplying histogram equalization on the corrected image, we can get the new image and corresponding histogram 2. After using histogram equalization again, we found that the picture and histogram did not change much.

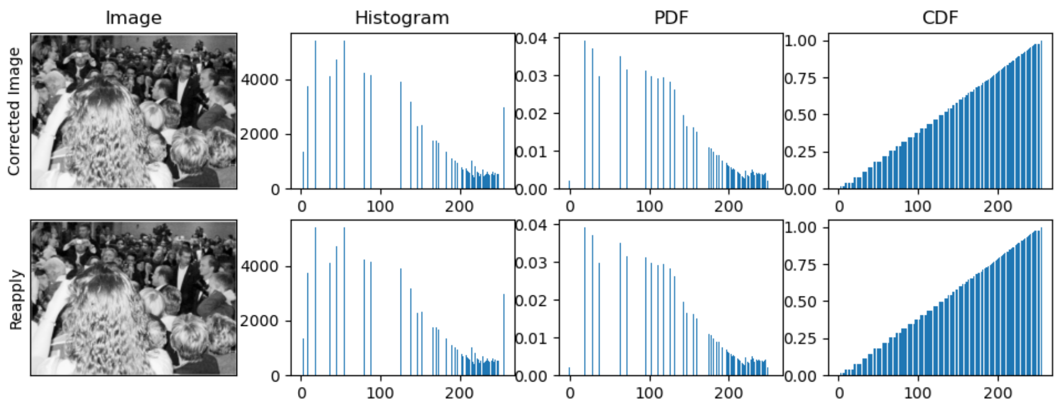

Figure 2: Reapply the histogram equalization

1.6 Apply histogram equalization to another low contrast image (greyscale)

We selected a lower-resolution image in the game Goose Goose Duck and performed histogram equalization. Before inputting the image, we need to process the image:

```
original_image = Image.open('gooseGooseDuck.png')
```

```
resized_image = original_image.resize((256, 256))
im = np.array(resized_image.convert('L'))
im_uint8 = ((im-np.min(im))*(1/(np.max(im)-np.min(im))*255)).astype('uint8')
```

The resulting image and histogram are as follows 3. From the figure, we can see that the equalized image is brighter than the original image, and more details can be seen, the PDF distribution is more dispersed, and the CDF forms a straight line.

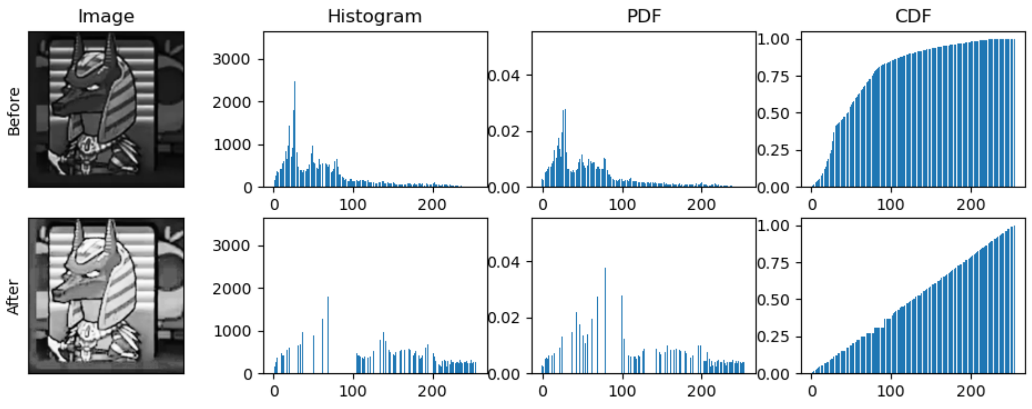

Figure 3: Apply histogram equalization to another low contrast image

2 Otsu Image Thresholding

For this section, you will implement Otsu thresholding algorithms to generate binarized images (where pixels are set to either 0 or 255 to produce a fully black and white image). See images b2_a.png, b2_b.png, and b2_c.png on NYU Brightspace alongside the assignment. Please note that b2_b is an original microscopy image, and b2_c is the image corrected by multiplication with an illumination bias field as discussed in our first lecture (see slides). Show results for all of these images in your assignment.

2.1 Manual Thresholding

For manual threshold A.5, we choose the threshold value to be 128 and generate binary images, as shown in the figure??.

```
# We choose the threshold to be 128.
manual_threshold_value = 128
```

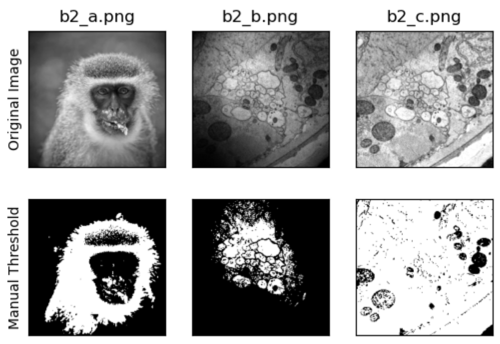

Figure 4: Manual Threshold

2.2 Otsu Image Thresholding

Otsu Image thresholding A.6 is a technique used for automatic thresholding in image processing. The aim is to separate pixels in an image into two value (0 and 255) based on the intensity values. The steps of Otsu thresholding are as follows:

1. Compute the histogram of the input image. The histogram represents the frequency distribution of pixel intensities in the image and get PDF and CDF from an input image.
2. Iterate through all possible threshold values (from 0 to 255 for an 8-bit grayscale image). For each threshold, divide the pixel intensities into two classes: those below the threshold and those equal to or above the threshold.
3. Compute the probabilities and mean intensities of the two classes and Calculate the intra-class variance for each threshold.
4. Select the threshold that maximizes the inter-class variance, which is computed based on the probabilities and mean intensities of the two classes. Finally we can binarize the image using the optimal threshold.

Using Otsu Image thresholding, we can get the resulting histograms for each image and the plot of the inter-class variance, as shown in figure 5.

In the figure, the second column indicates the binary image produced by the algorithm. The blue histogram in the third column represents the PDF of the image in the same column, and the red curve represents the CDF of the image. In addition, the green curve in the fourth column represents the inter-class variance, while the orange straight line represents the optimal threshold value. According to the Otsu's algorithm, we can get the optimal thresholds of the three images:

```
Inter-class Variance of b2_a.png : 132
Inter-class Variance of b2_b.png : 86
Inter-class Variance of b2_c.png : 171
```

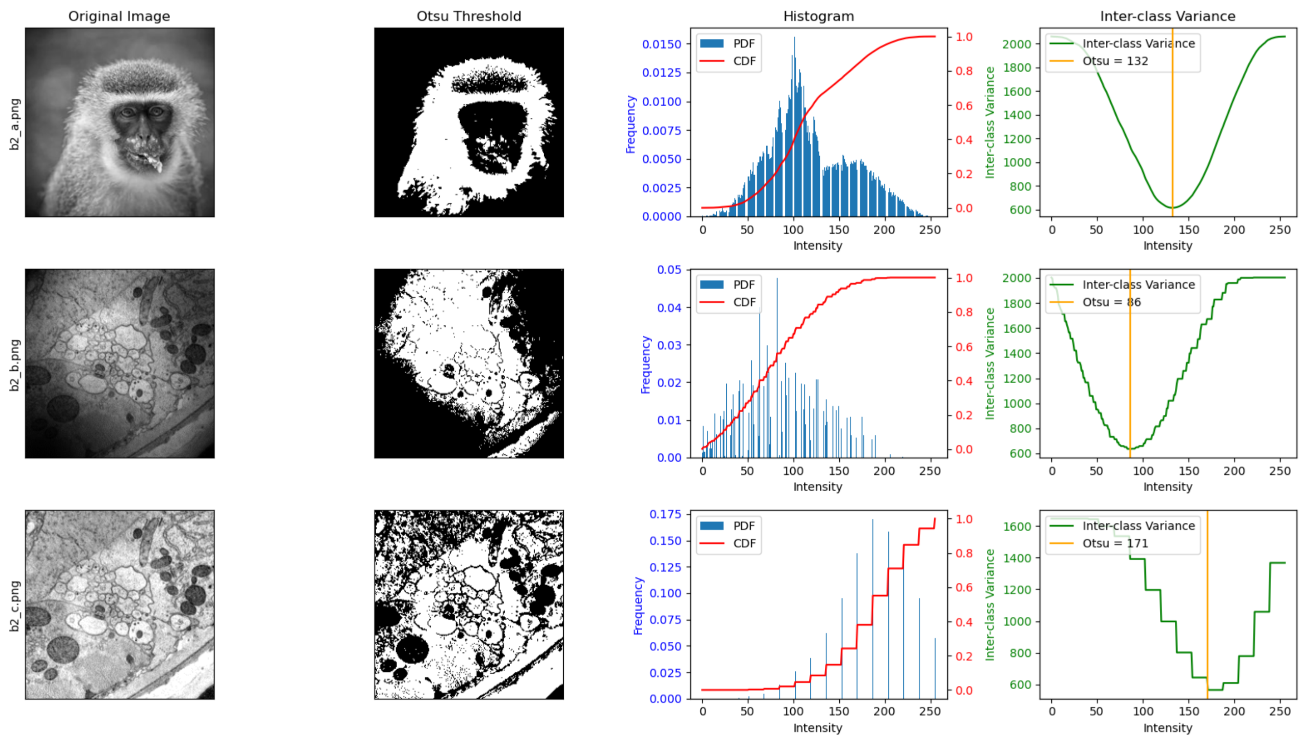

Figure 5: Otsu Image Thresholding

According to the results, we can find that automatic threshold can produce a decent result compared to manual thresholding.

For b2_a.png, the outline of the object is relatively smooth compared to manual thresholding. For b2_b.png, More cells in the microscope are classified as foreground (0) than background (255) compared to manual thresholding because the original image is relatively darker. For b2_c.png, the foreground and background of the picture are more balanced compared to manual thresholding.

While Otsu's method generally provides effective automatic thresholding, there are some potential areas for improvement. For example, we can preprocess the image such as noise reduction and histogram equalization, which can make thresholding more robust.

3 Creative Part (Histogram Matching)

Now that you have a code for histogram equalization, you can apply, extend and test the method to applications on your own. Choose the specific area of histogram matching and test the method on your own images.

Histogram matching A.7 is a technique which is used to transfer the contrast characteristics from one image to another. The steps to implement histogram matching are as follows:

1. Compute the histogram of the source image and target image. The histogram represents the frequency distribution of pixel intensities in the image and get PDF and CDF from an input image.
2. Map the pixel values of the source image through a mapping function to match the CDF of the target image.
3. Finally, we can adjust the pixel values of the source image, resulting in a matched image.

We can get the matched image using histogram matching, as shown in figure 6.

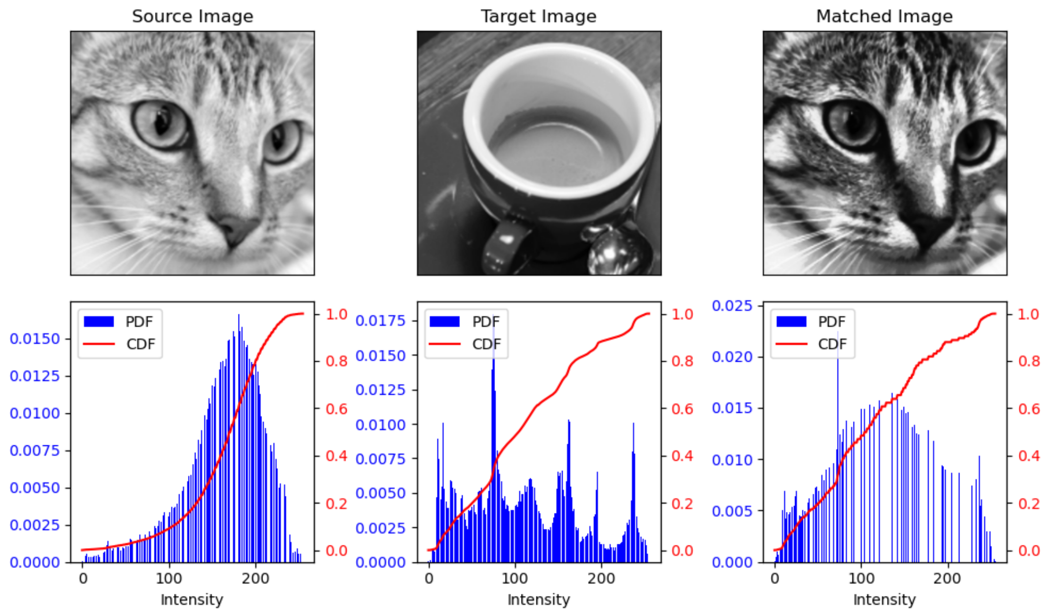

Figure 6: Histogram Matching

According to the result, we can find that the CDF of matched image has the similar shape as the original image and the pixels with higher intensity levels become more prominent in the image.

A Appendix Code

I have placed the complete code (including functions and drawings) in the **code.ipynb** file. In the appendix, I added the implementation of core functions such as histogram Equalization, Otsu Image Thresholding, etc.

A.1 Create histogram

```python
def create_hist(image):
    pixel_count = np.zeros(256)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            intensity = image[i][j]
            pixel_count[intensity] += 1
    return pixel_count
```

A.2 Create PDF

```python
def create_pdf(im_in):
    # Create normalized intensity histogram from an input image
    pdf = np.zeros(256)
    for i in range(im_in.shape[0]):
        for j in range(im_in.shape[1]):
            intensity = int(im_in[i][j])
#             intensity = im_in[i][j]
            pdf[intensity] += 1
    pdf = pdf/im_in.size
    return pdf
```

A.3 Create CDF

```python
def create_cdf(pdf):
    # Create the cumulative distribution function from an input pdf
    cdf = np.zeros(256)
    cdf[0] = pdf[0]
    for i in range(1, 256):
        cdf[i] = cdf[i-1] + pdf[i]
    return cdf
```

A.4 Histogram Equalization

```python
def histogram_equlization(im_in):
    pdf = create_pdf(im_in) # Your previously implemented function
    cdf = create_cdf(pdf)  # Your previously implemented function
    # Create a histogram equalized image using your computed cdf
    equalized_im = np.zeros(len(im_in.ravel()))
    for i, pixel_value in enumerate(im_in.ravel()):
        equalized_im[i] = int(cdf[pixel_value] * 255)
    equalized_im = np.reshape(equalized_im, im_in.shape)
    equalized_im = equalized_im.astype(int)
    return equalized_im
```

A.5 Manual Threshold

```
def manual_threshold(im_in, threshold):
    # Threshold image with the threshold of your choice
    manual_thresh_img = np.zeros_like(im_in)
    manual_thresh_img[im_in > threshold] = 255
    return manual_thresh_img
```

A.6 OTSU Threshold

```
def otsu_intraclass_variance(image, threshold):
    """
    Otsu's intra-class variance.
    If all pixels are above or below the threshold, this will throw a warning
       that can safely be ignored.
    """
    return np.nansum([
        np.mean(cls) * np.var(image,where=cls)
        #   weight   *  intra-class variance
        for cls in [image>=threshold,image<threshold]
    ])
    # NaNs only arise if the class is empty, in which case the contribution
       should be zero, which `nansum` accomplishes.

def otsu_threshold(im_in):
    # Create Otsu thresholded image
    otsu_threshold_value = min(
            range( np.min(im_in)+1, np.max(im_in) ),
            key = lambda th: otsu_intraclass_variance(im_in,th)
        )
    otsu_thresh_img = manual_threshold(im_in, otsu_threshold_value)
    # return otsu_thresh_img
    return otsu_thresh_img, otsu_threshold_value
```

A.7 Histogram Matching

```
def histogram_matching(source_im, target_im):
    source_pdf = create_pdf(source_im)
    source_cdf = create_cdf(source_pdf)
    target_pdf = create_pdf(target_im)
    target_cdf = create_cdf(target_pdf)
    lut = np.interp(source_cdf, target_cdf, range(256))
    matched_im = lut[source_im]
    return matched_im
```