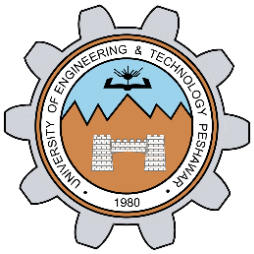
Assignment 02



Spring 2024

Digital Image Processing

Submitted by:
Safi Ullah Khan (20pwcse1943)
Class Section: **B**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Submitted to:
Dr. Abeer Irfan
(May 24, 2024)

Department: Computer System Engineering

Figure 1 code 4
Figure 2 code2 5
Figure 3 code3 5
Figure 4 output1 6
Figure 5 output2 6
Figure 6 code4 7
Figure 7 code5 7
Figure 8 outpu4 8
Figure 9 output5 8
Figure 10 output6 9
Figure 11 output7 9
Figure 12 histocode 10
Figure 13 histogram output 10

What are Histograms?

A histogram is an approximate representation of the distribution of some numerical data. It is the most commonly used graph to show frequency distributions. It looks very much like a bar chart.

In terms of image:

Histogram is a graphical representation of the intensity distribution of an image. In simple terms, it represents the number of pixels for each intensity value considered.

Histograms applications

1. Histograms are preferred in applications, when their is a need for any of the following:
2. The data are numerical and the distribution of data is needed to be observed, especially when determining whether the output of a process is distributed approximately normally or not
3. Seeing whether a process change has occurred from one time period to another
4. Determining whether the outputs of two or more processes are different
5. Statistical properties need to be modeled
6. Some specific use cases of histograms are
7. In hydrology the histogram and estimated density function of rainfall are used to gain insight in their behaviour and frequency of occurrence.
8. In many Digital Image processing programs there is a histogram tool, which show you the distribution of the contrast / brightness of the pixels.

Histogram Equalizing

Histogram equalization is a method for contrast adjustment using the image's histogram.Basically, it is a computer image processing technique used to improve contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values. When the useable data is represented by near contrast values, this approach generally boosts the global contrast of pictures. This enables locations with poor local contrast to gain a higher contrast.

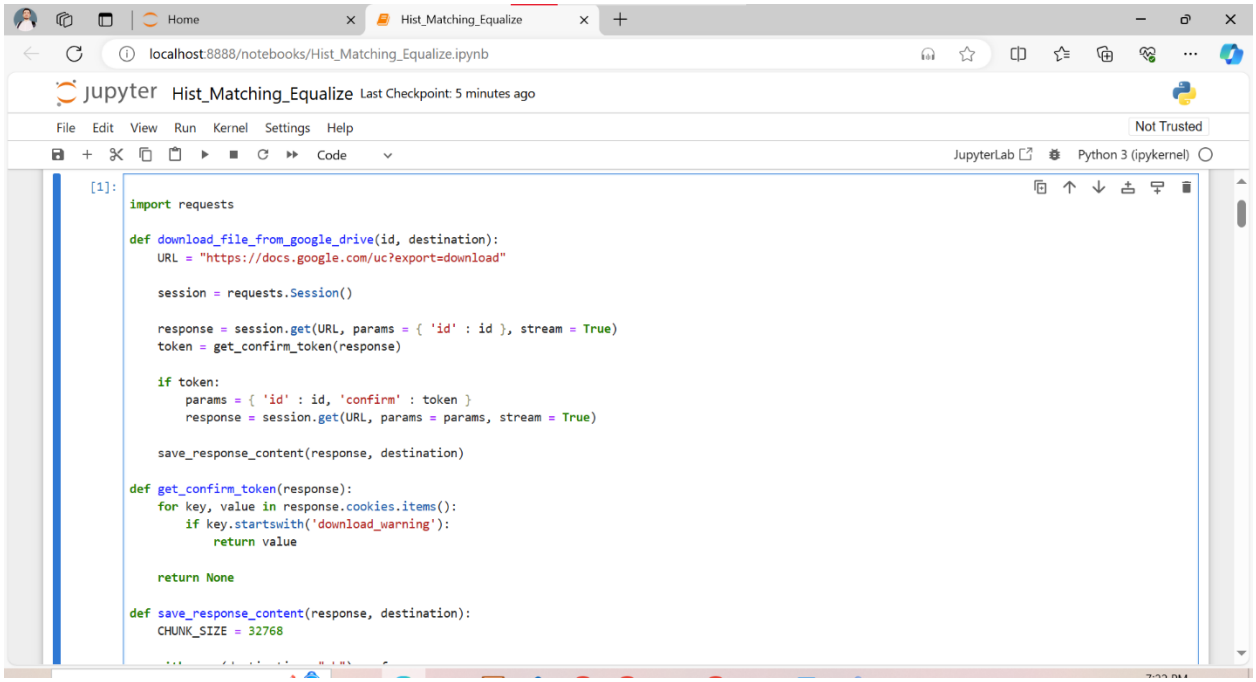


Figure 1 code

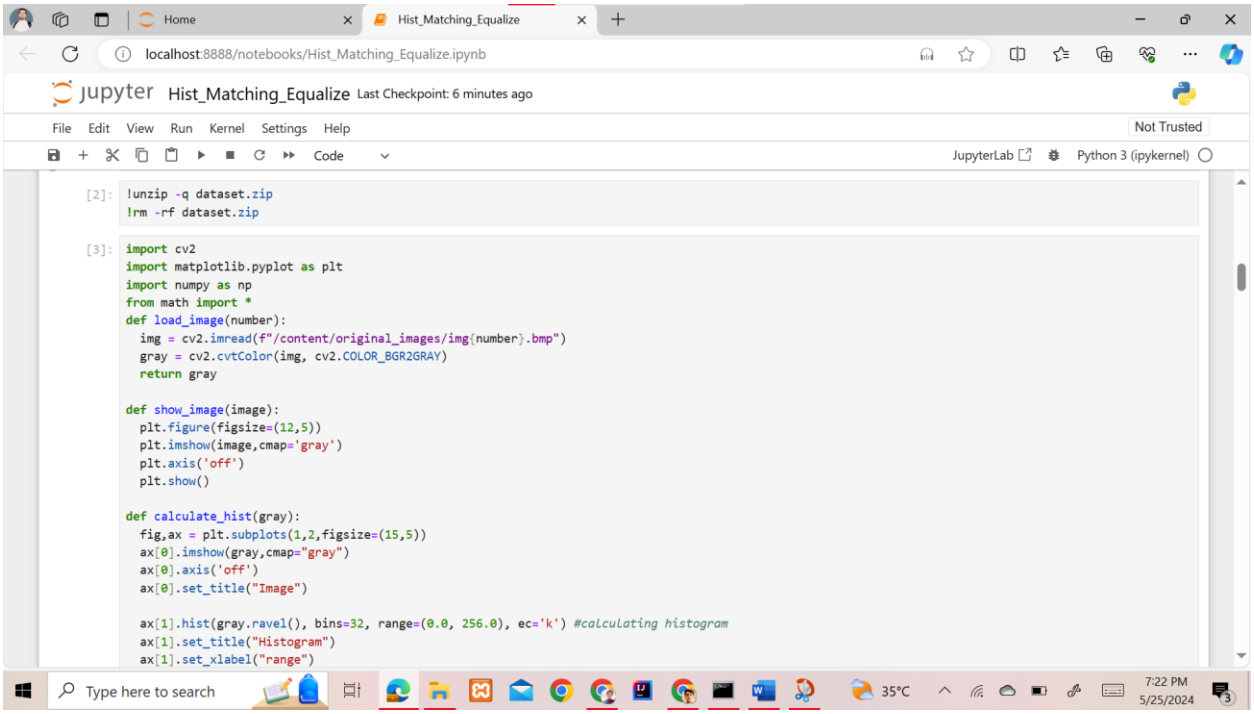


Figure 2 code2

```
def compare_matched_hist(src,dst,matched_src):
  images = [src,dst,matched_src]
  headings = ["Source","Destination","Matched Source"]
  n,m = len(images),2
  fig,ax = plt.subplots(n,m,figsize=(15,10))

  for i, (heading,img) in enumerate(zip(headings,images)):
    ax[i,0].imshow(img,cmap="gray")
    ax[i,0].axis('off')
    ax[i,0].set_title(heading)

    ax[i,1].hist(img.ravel(), bins=32, range=(0.0, 256.0), ec='k') #calculating histogram
  plt.show()
```

Figure 3 code3

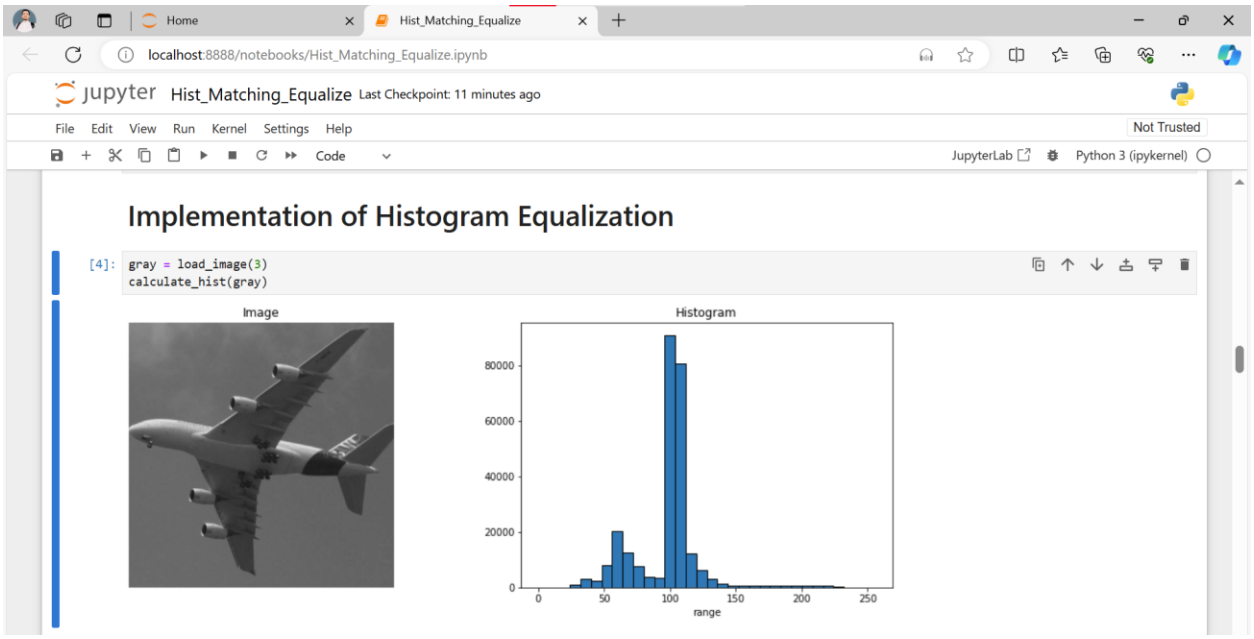


Figure 4 output1

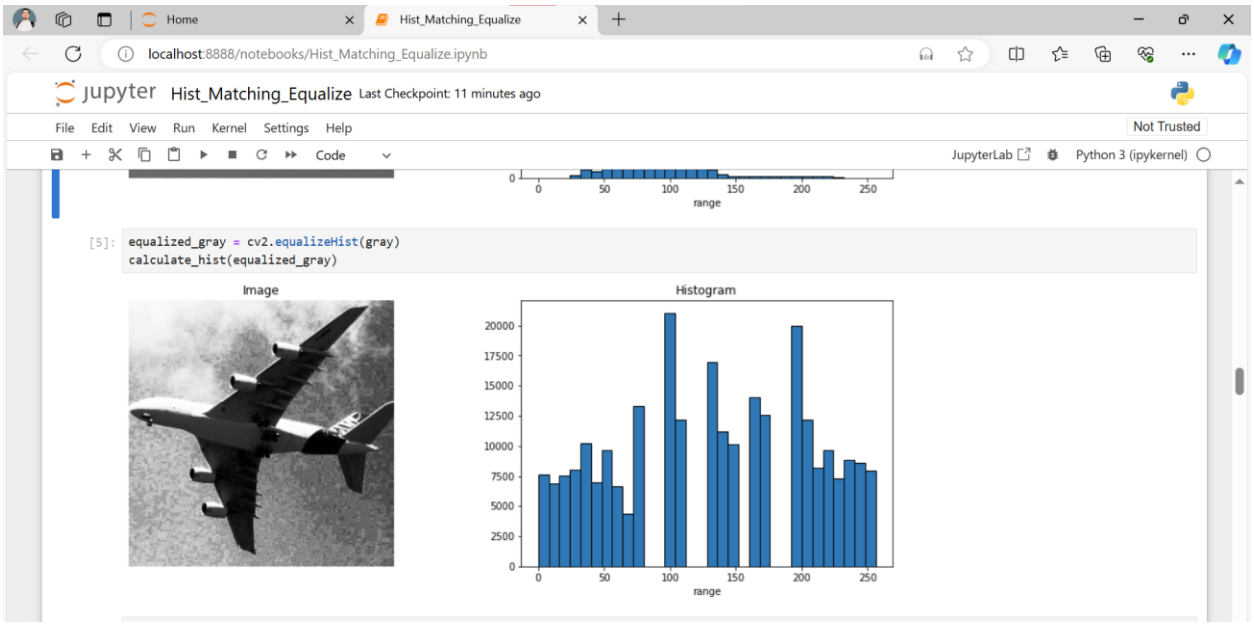


Figure 5 output2

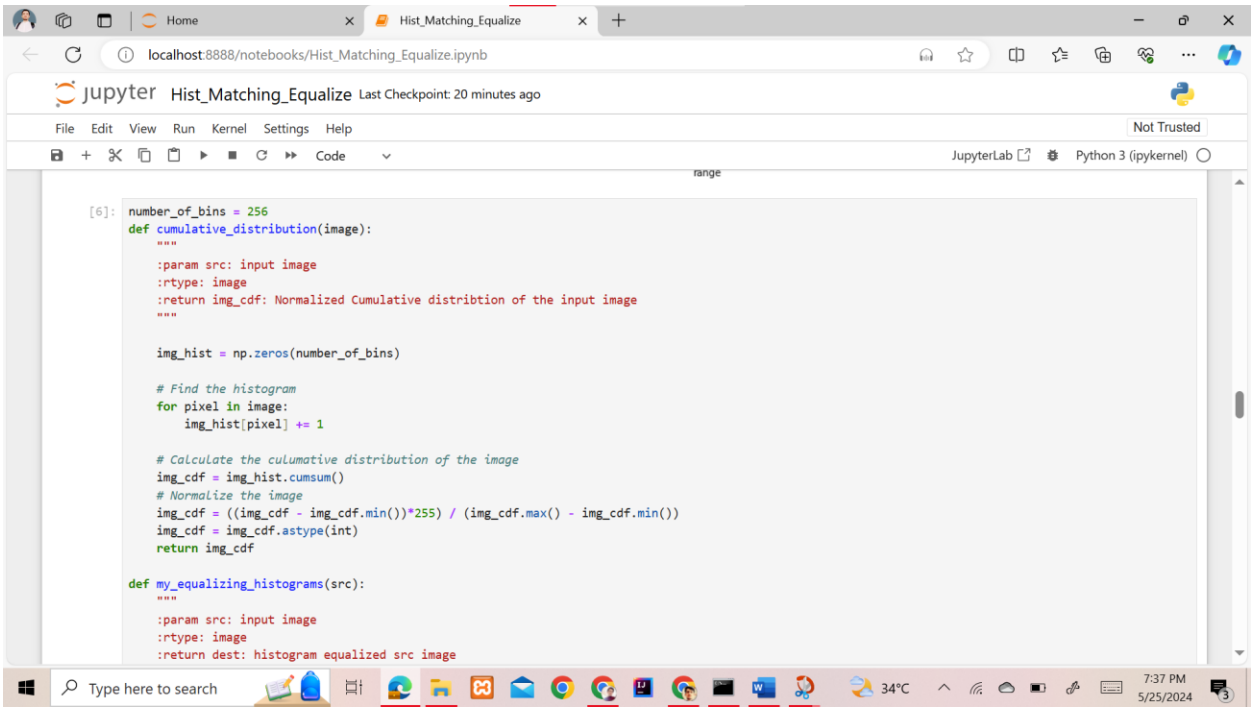


Figure 6 code4

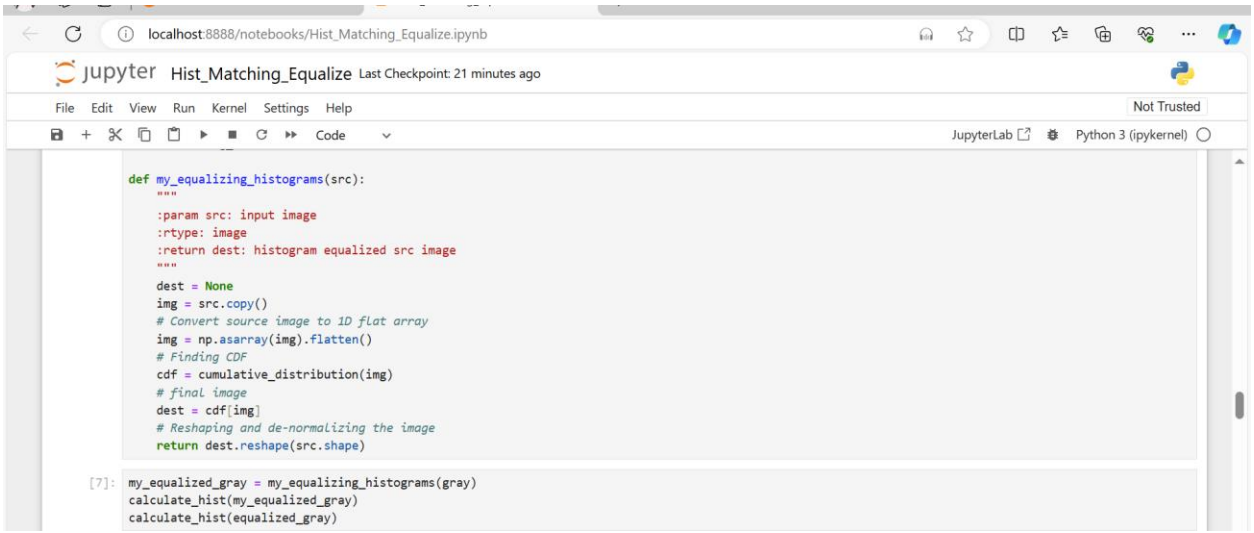


Figure 7 code5

```
    return dest.reshape(src.shape)
```

```
[7]: my_equalized_gray = my_equalizing_histograms(gray)
     calculate_hist(my_equalized_gray)
     calculate_hist(equalized_gray)
```

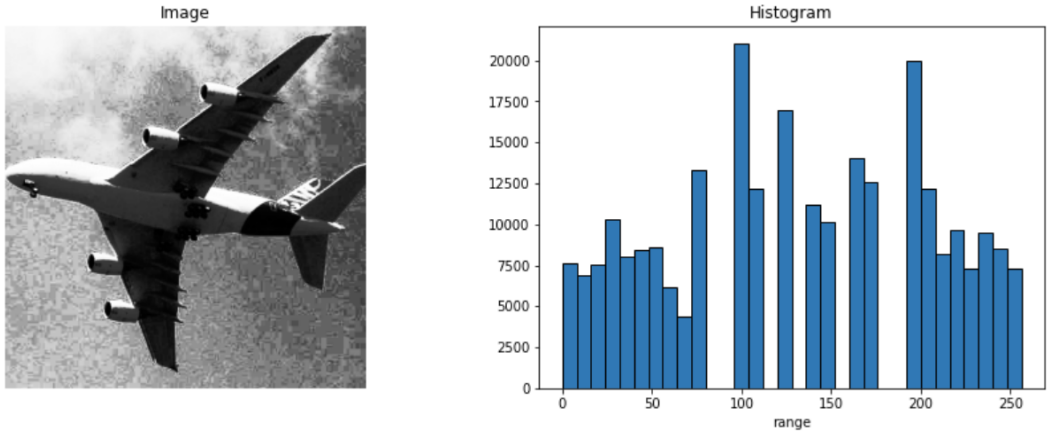


Figure 8 outpu4

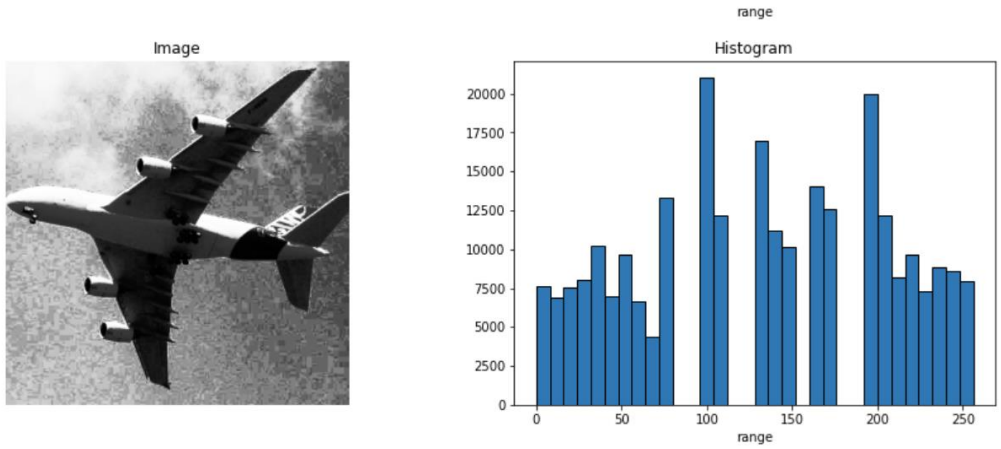


Figure 9 output5

Histogram matching

Comparing with the inbulit results, my custom implementation seems to be reproducing almost similar results. The image histogram of the inbuilt function and the custom implementation are almost similar. Apart from the histogram, the contrast of the image itself has been improved, and the implementation seems to be performing histogram equalization correctly.

Histogram matching is the transformation of an image so that its histogram matches a specified histogram. In order to match the histogram of images A and B, we need to first equalize the histogram of both images. Then, we need to map each pixel of A to B using the equalized histograms. Then we modify each pixel of A based on that of B.

Histogram matching may be used to balance detector responses. It can be used to equalise two pictures that were taken in the same place with the same local lighting (such as shadows), but with different sensors, atmospheric conditions, or global illumination.

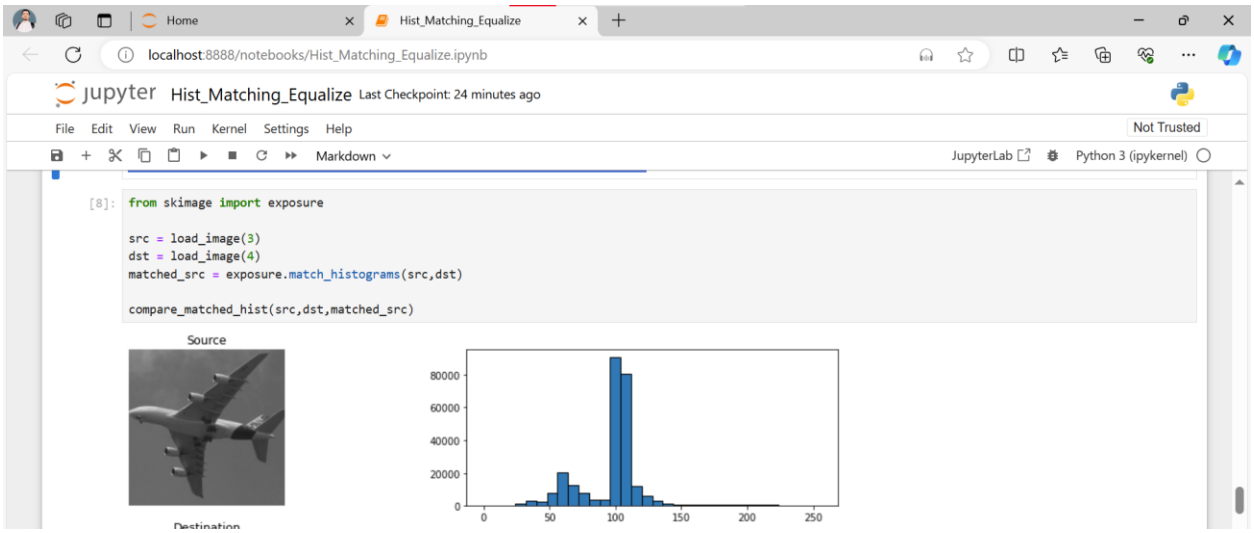


Figure 10 output6

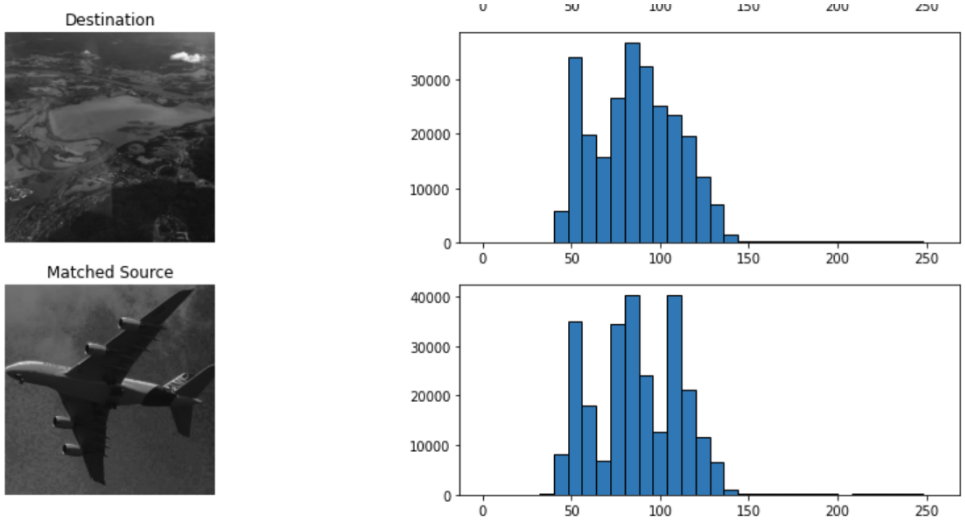


Figure 11 output7

[9]:
```python
def my_matching_histograms(src,dst):
    """
    :param src: input image
    :param dst: reference image
    :rtype: image
    :return mathced_src: histogram matched src image
    """
    src_values, src_unique_indices, src_counts = np.unique(src.ravel(), return_counts=True, return_inverse=True)
    dst_values, dst_counts = np.unique(dst.ravel(), return_counts=True)

    # Normalizing cdf for source and destination images
    src_cdf = np.cumsum(src_counts) / len(src)
    dst_cdf = np.cumsum(dst_counts) / len(dst)

    matched_src = np.interp(src_cdf, dst_cdf, dst_values)
    return matched_src[src_unique_indices].reshape(src.shape)
```

Figure 12 histocode

[10]:
```python
my_matched_src = my_matching_histograms(src,dst)
compare_matched_hist(src,dst,my_matched_src)
compare_matched_hist(src,dst,matched_src)
```

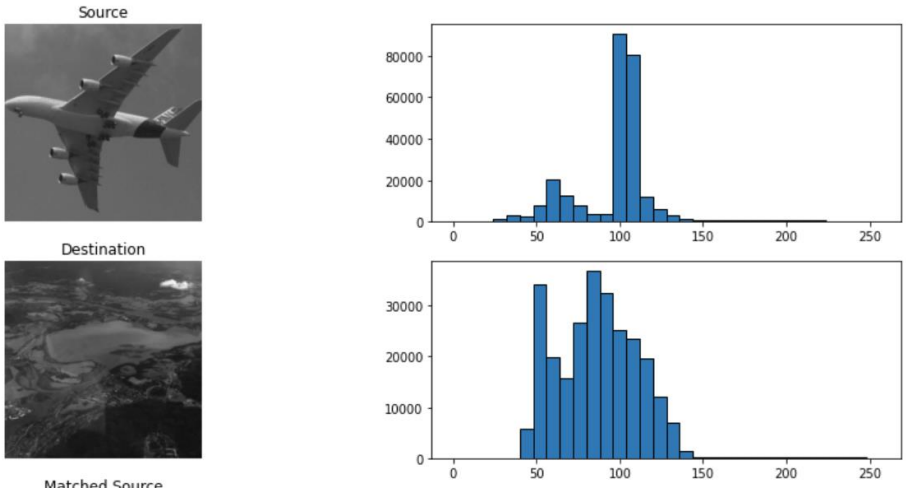
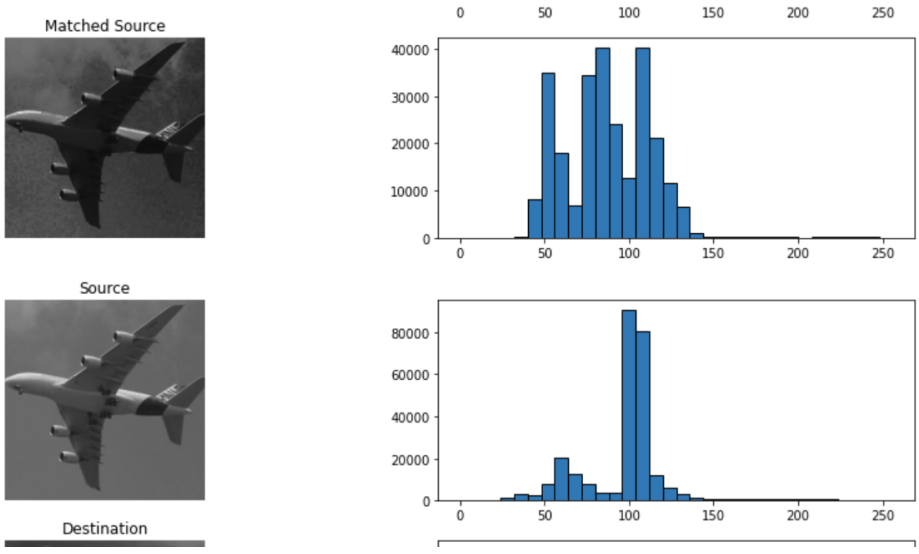
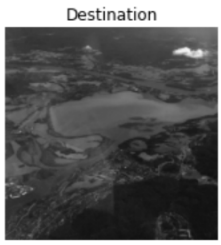
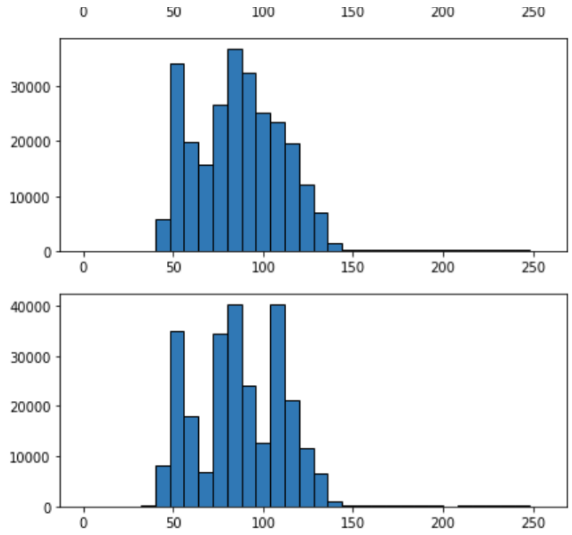


Figure 13 histogram output







Comparing with the inbuilt results, both the image and the histogram appear to be almost similar. The purpose of histogram matching also seems to have been achieved as far as it can be. Since the source and the destination images are completely different one, therefore tha matching is not perfect. However the overall lighting and contrast seems to have been matched to the destination image.

```
[11]: src = load_image(1)
      dst = load_image(2)

[12]: %%timeit

      my_equalizing_histograms(src)
      my_matching_histograms(src,dst)

      1 loop, best of 5: 203 ms per loop
```

Reference

- https://en.wikipedia.org/wiki/Histogram_equalization
- https://en.wikipedia.org/wiki/Cumulative_distribution_function
- https://en.wikipedia.org/wiki/Histogram_matching
- http://paulbourke.net/miscellaneous/equalisation/