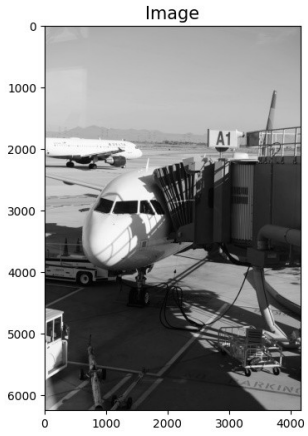
Report on Project_1

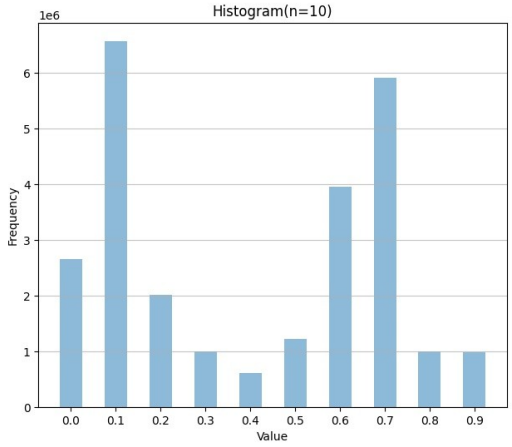
1. Build a histogram

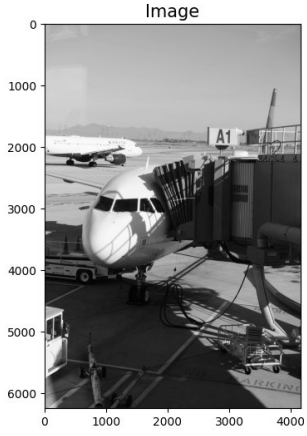
In this question, I wrote a function from scratch to build histogram from an image. The following steps were followed to write this function:

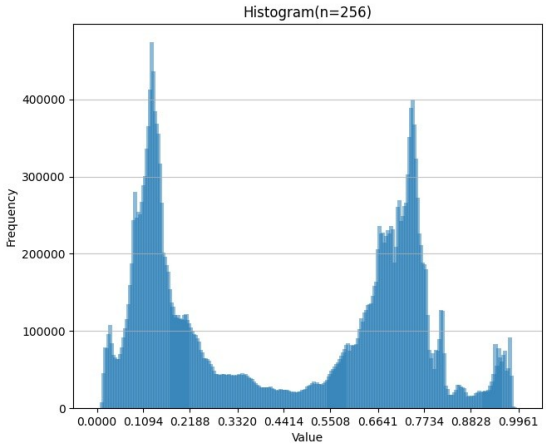
- The function takes two parameters: image and the number of bins.
- First, read an image and convert it to grayscale using **skimage.io.imread** and **skimage.color.rgb2gray** function.
- Then, calculate the bin edges using **numpy.linspace** function to get evenly spaced bin edges.
- After that, iterate through each bin value and find the indexes that has pixel value between the edges of a bin using **numpy.where** function and calculate the total number of pixels.
- Finally, rearrange the bin edges and bin count arrays in two columns by using the **np.vstack** function.

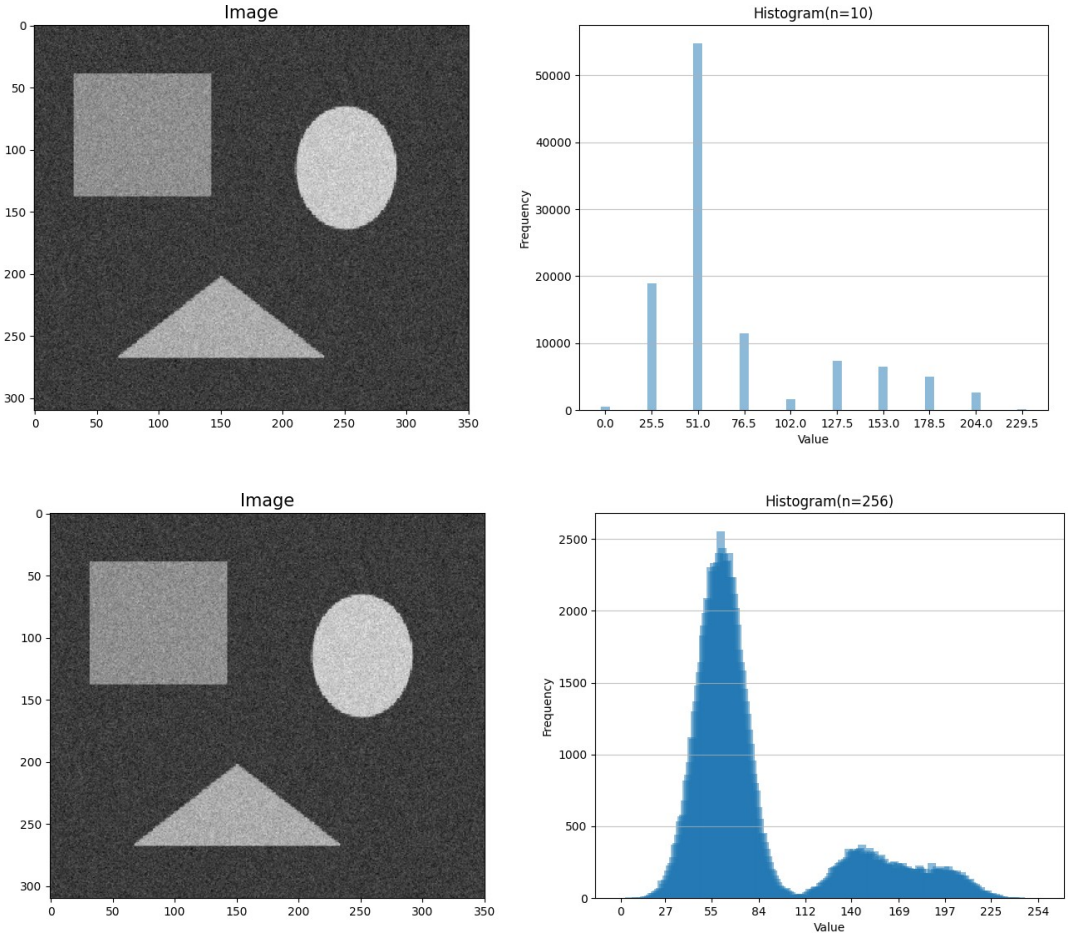
Example outputs:











From the outputs, we can see that large number of bins tend to provide more detailed distribution of the pixel values and hence, it will be more useful to choose thresholds for different applications.

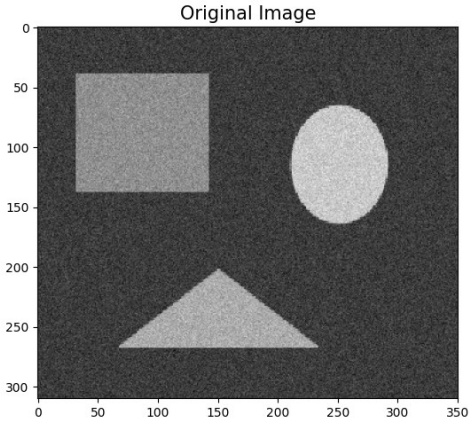
2. Regions and components

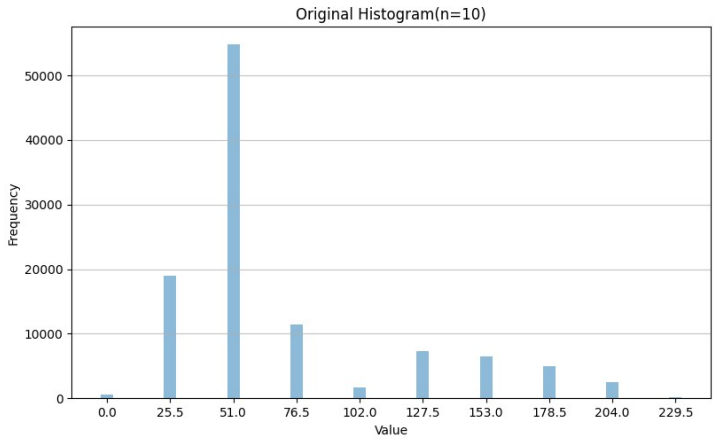
Here's the steps I followed for writing the double sided thresholding function:

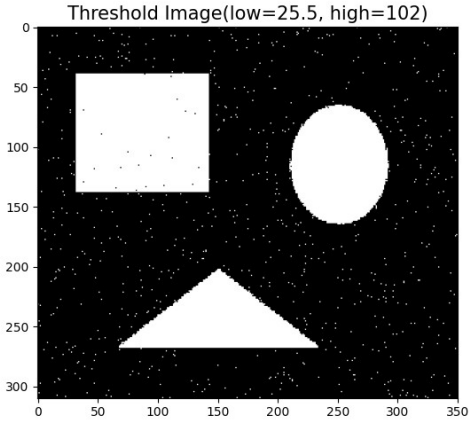
- The function takes three parameters: low threshold, high threshold and image.
- First, read an image and convert it to grayscale using **skimage.io.imread** and **skimage.color.rgb2gray** function.
- Then create an array with same shape as the image but fill all the pixels with the highest pixels value of the image. This array will be used to create the threshold image using **numpy.full** function.

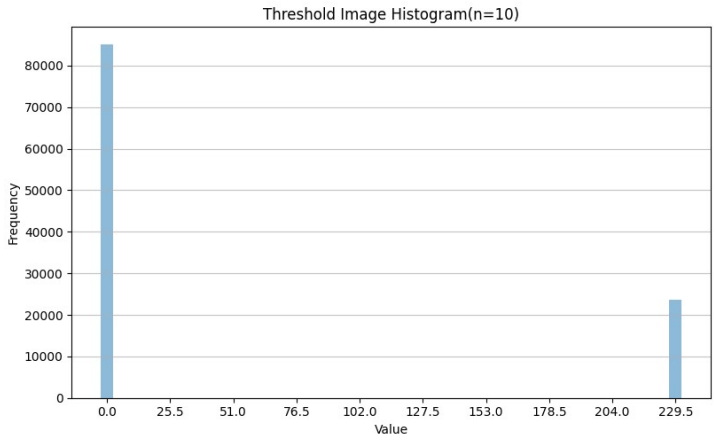
- Then find the indexes that has pixel value greater than the low threshold and lower than the high threshold using **numpy.where**.
- Set the pixel values for those indexes with the minimum pixel value of the image.

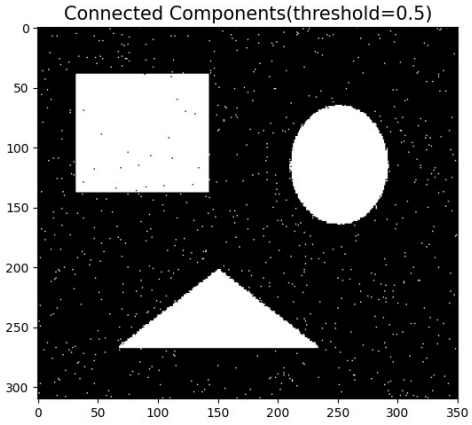
Example outputs:

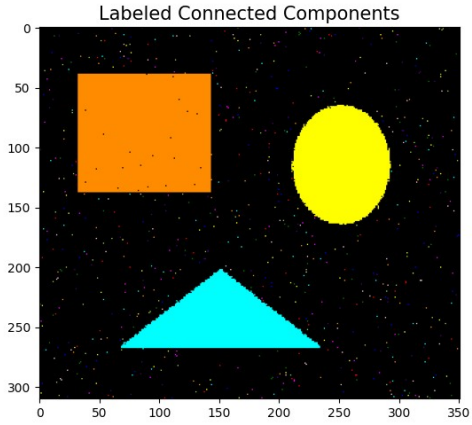






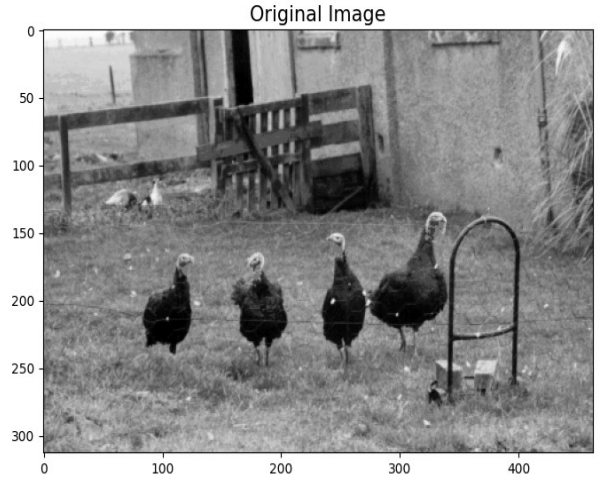


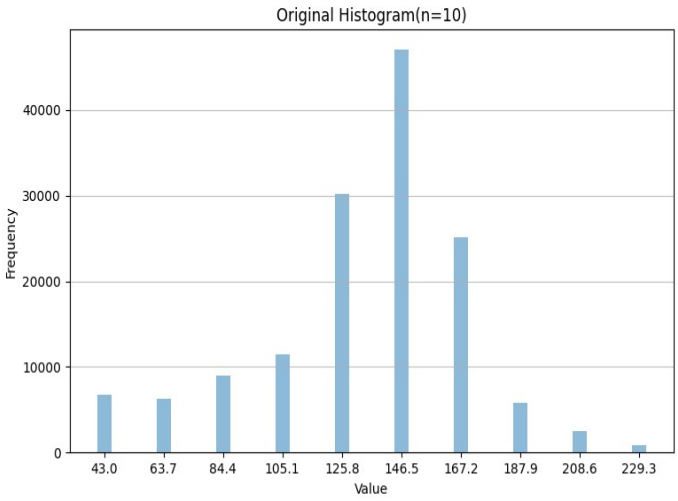


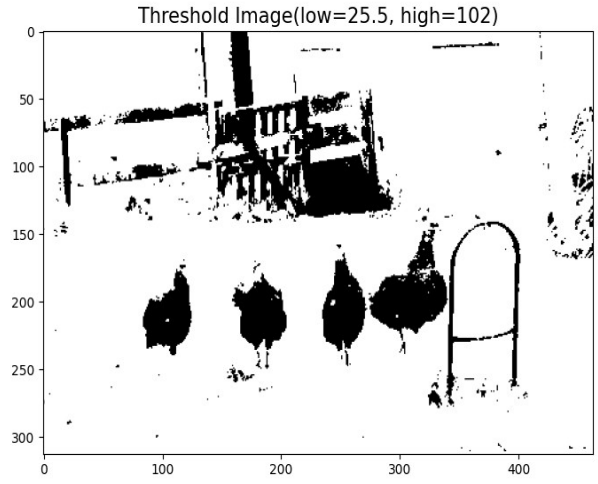


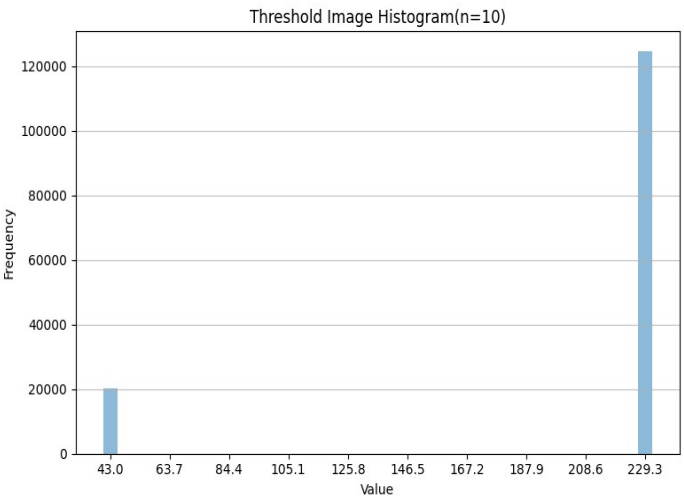
Here, we can see 3 connected components were labeled with different colors after choosing taking pixels that has the highest bin counts.

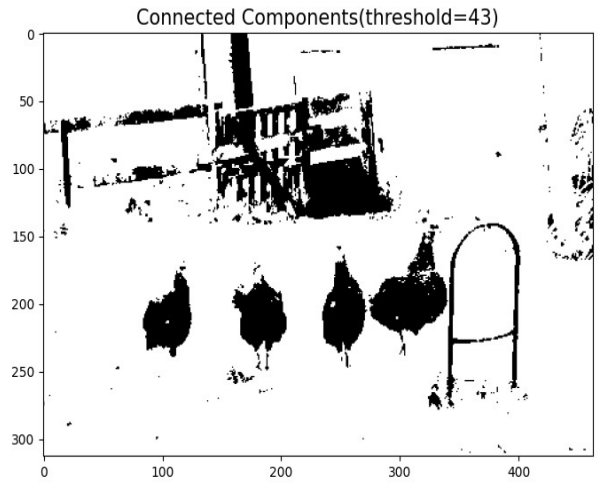
Here, taking threshold values that have lower bin counts can’t provide distinction between each turkey.

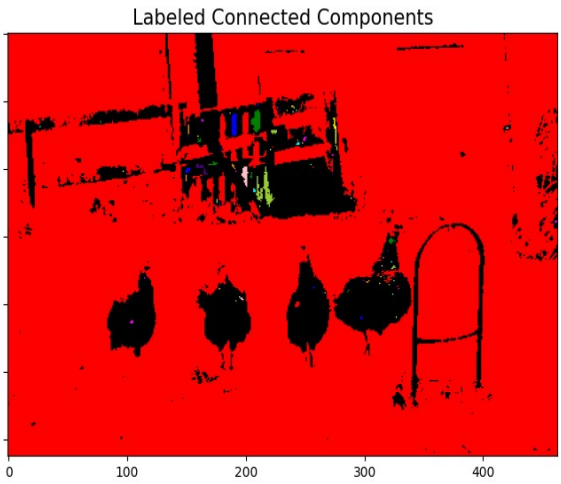




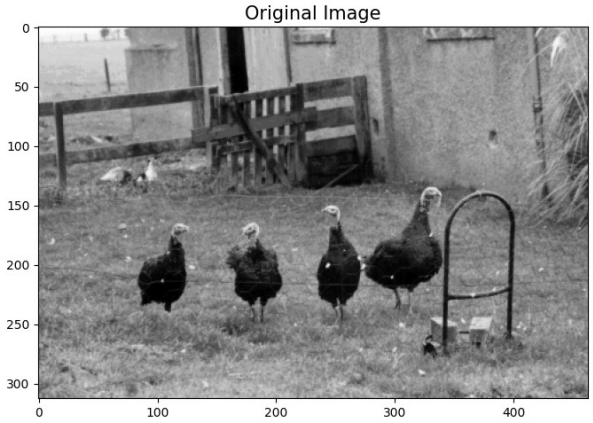


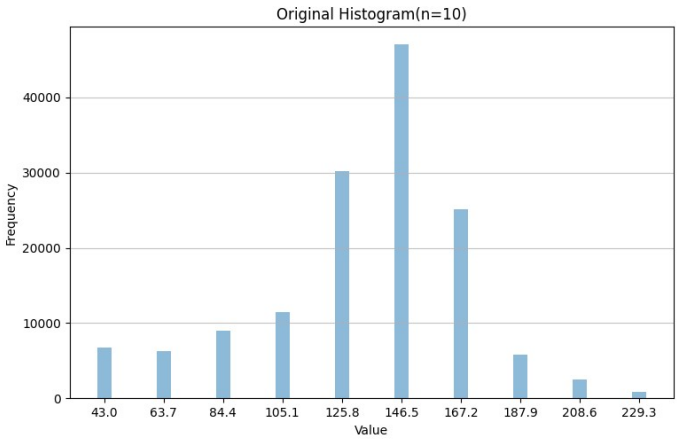


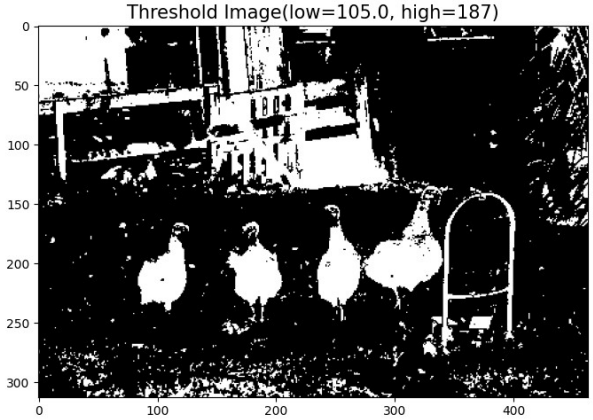


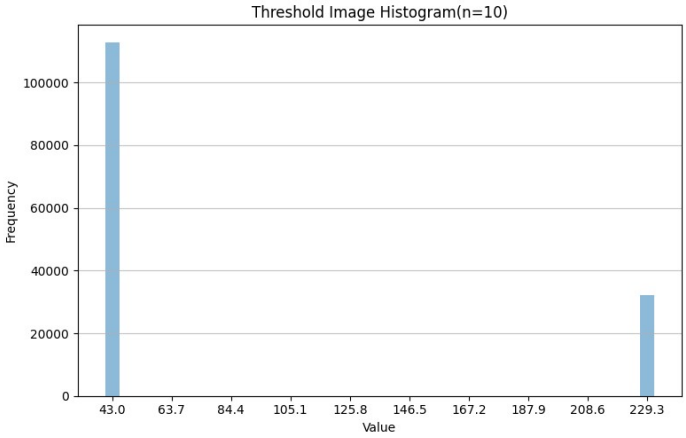


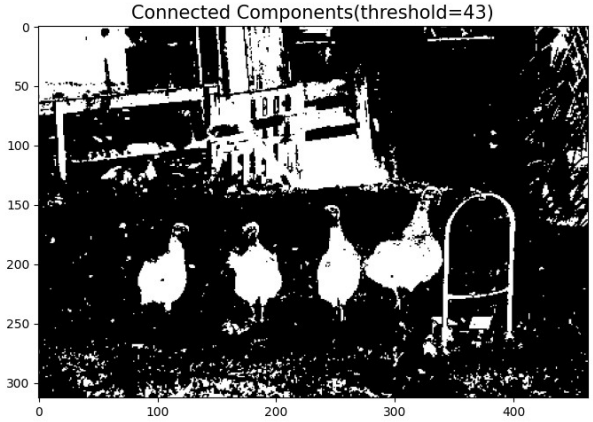
However, when we choose threshold values that has pixel values with higher bin count the connected components are better distinguishing the turjeys. This is how histogram bin edges can be used to choose threshold values in order to get better connected components.

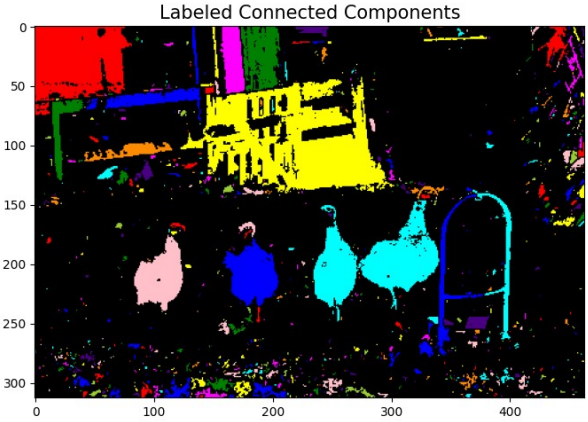






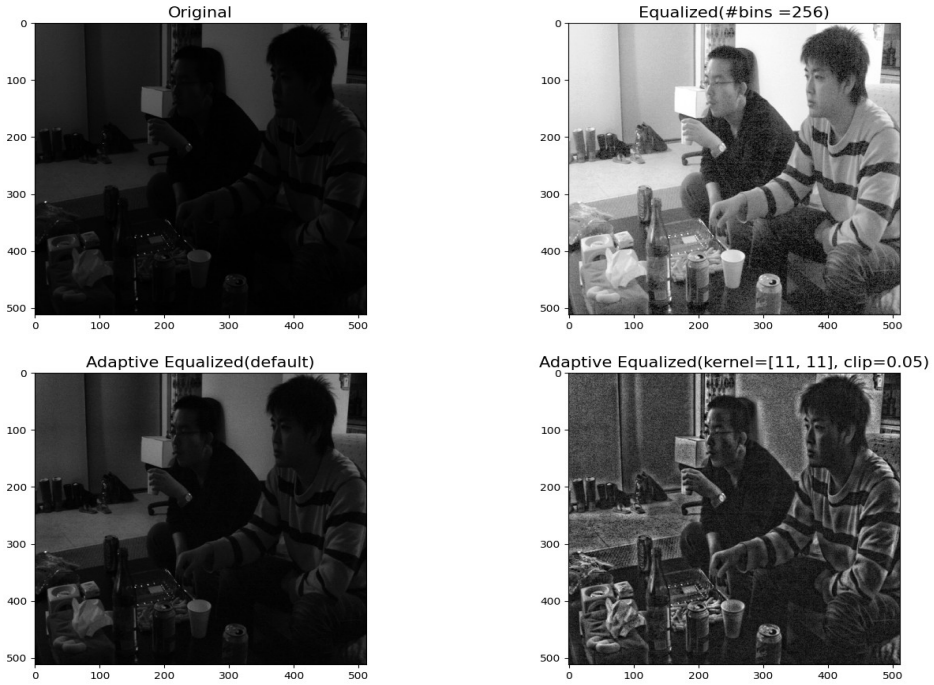


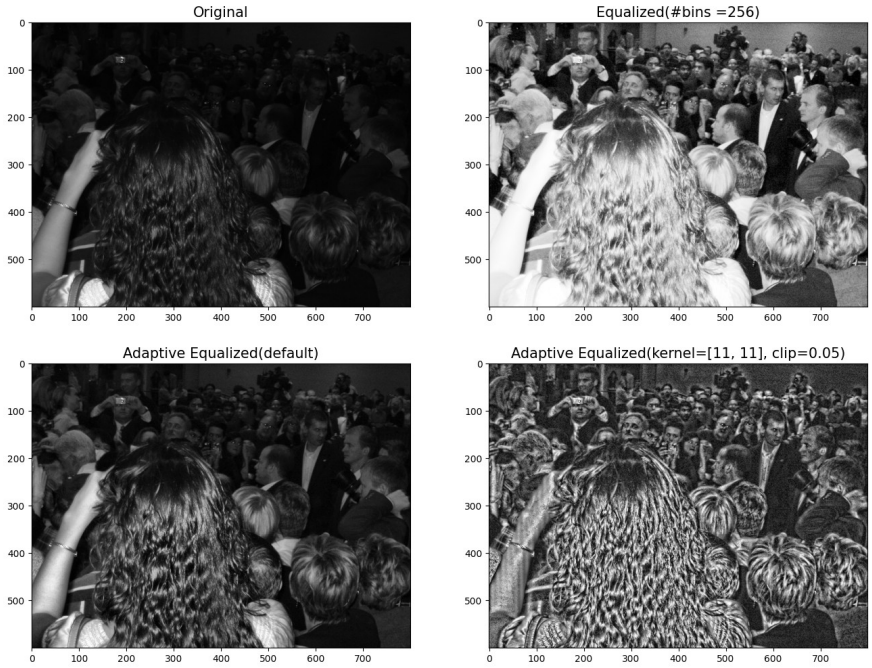


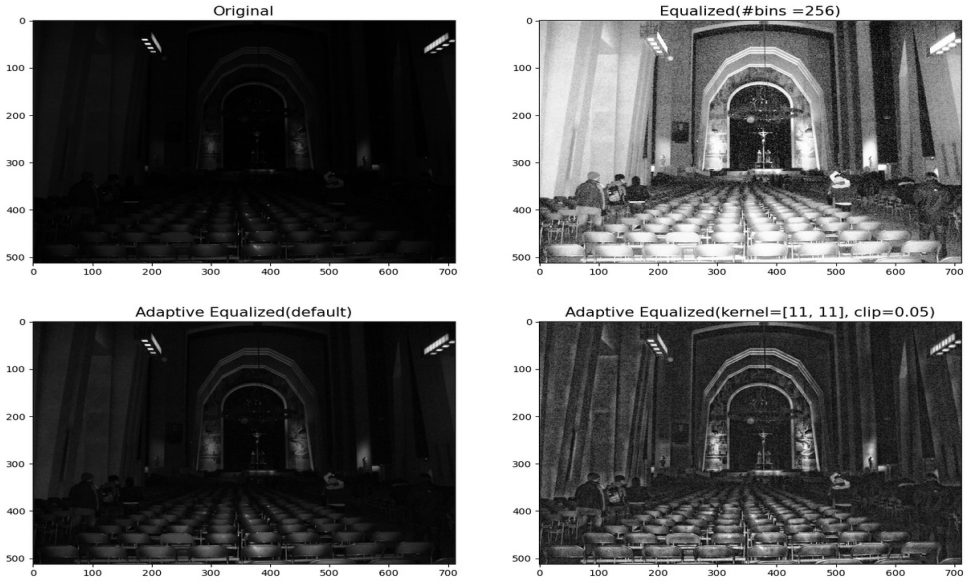


3. Histogram equalization

For histogram equalization we used two function: **skimage.exposure.equalize_hist** (one parameter: number of bins)**,** **skimage.exposure.equalize_adapthist** (two parameters: kernel size, clipping size). Here's some outputs with different parameter settings.






Original
Equalized(#bins =256)
Adaptive Equalized(default)
Adaptive Equalized(kernel=[11, 11], clip=0.05)
0
100
200
300
400
500
600
700