National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

EE 433: Digital Image Processing

Amal Saqib (282496)

Muhammad Hadi (305774)

Class: BSCS 9C

Lab 7: Local Histogram Equalization

Date: 1st November 2021

Time: 2.00Pm to 5.00Pm

Instructor: Dr. Imran Malik

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Lab 7

Local Histogram Equalization

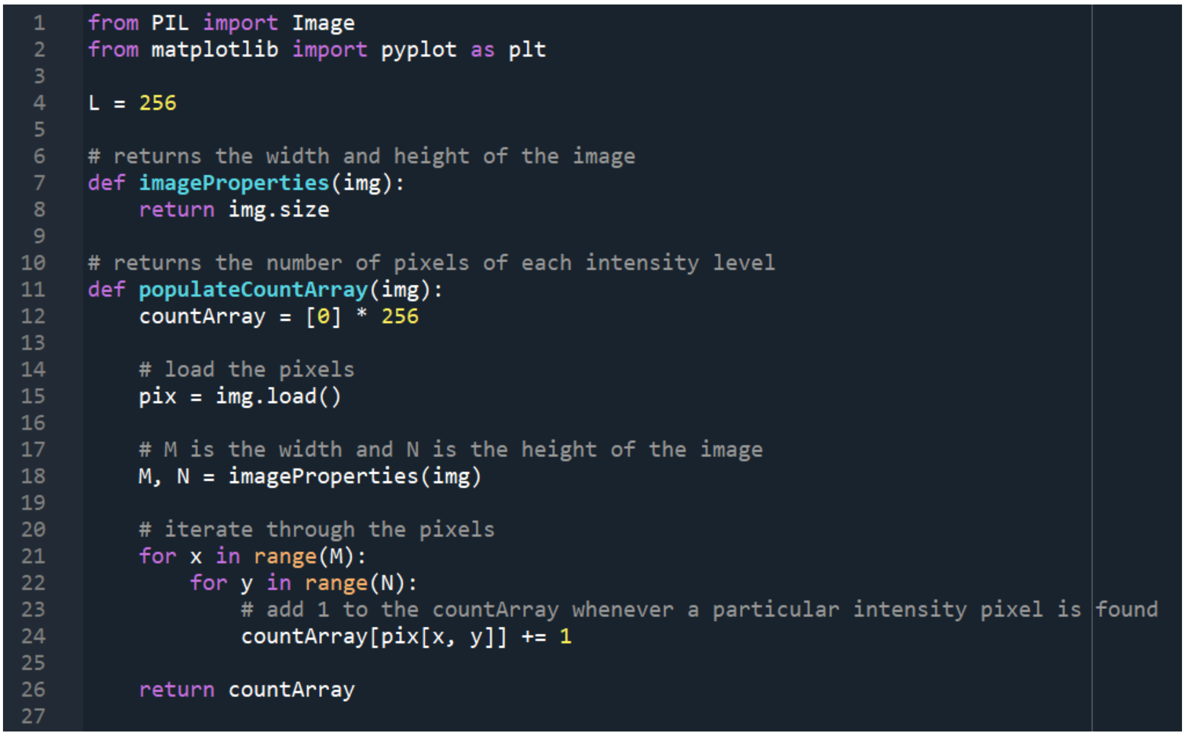
Task 1

Global Histogram Equalization

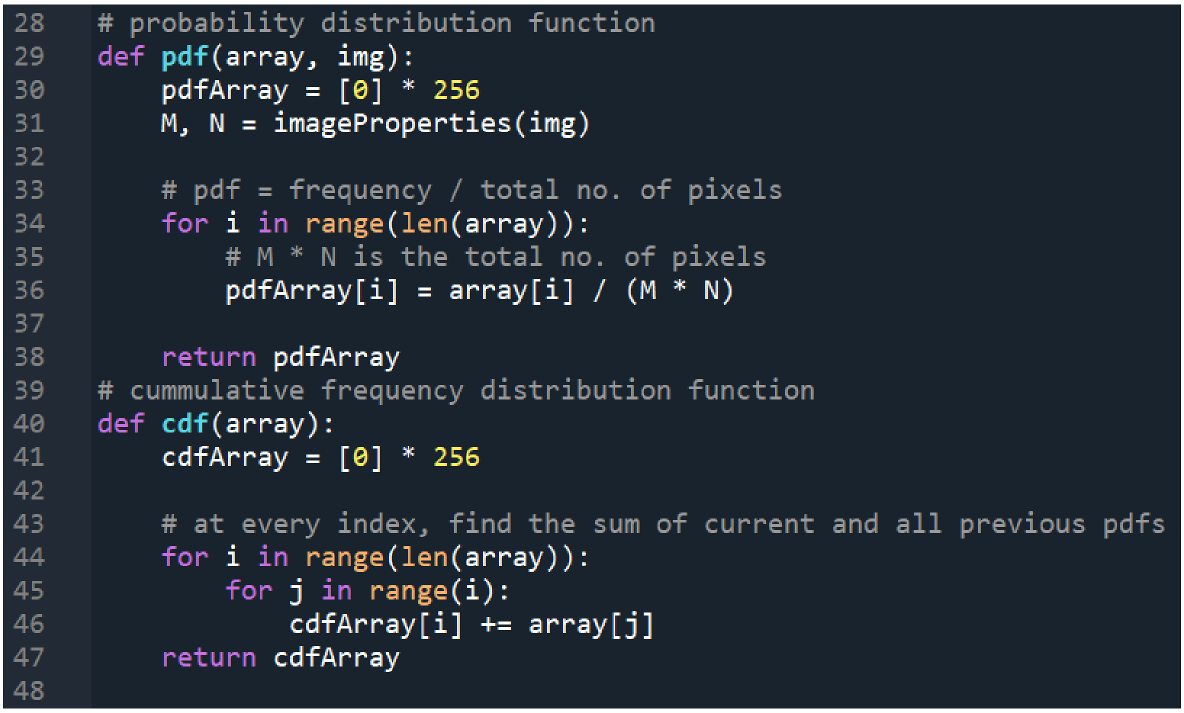
CODE

```
from PIL import Image

from matplotlib import pyplot as plt


L = 256


# returns the width and height of the image

def imageProperties(img):

    return img.size


# returns the number of pixels of each intensity level

def populateCountArray(img):

    countArray = [0] * 256


    # load the pixels

    pix = img.load()


    # M is the width and N is the height of the image

    M, N = imageProperties(img)


    # iterate through the pixels

    for x in range(M):
        for y in range(N):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1

    return countArray

# probability distribution function
def pdf(array, img):
    pdfArray = [0] * 256
    M, N = imageProperties(img)

    # pdf = frequency / total no. of pixels
    for i in range(len(array)):
        # M * N is the total no. of pixels
        pdfArray[i] = array[i] / (M * N)

    return pdfArray
# cummulative frequency distribution function
def cdf(array):
    cdfArray = [0] * 256

    # at every index, find the sum of current and all previous pdfs
    for i in range(len(array)):
        for j in range(i):
            cdfArray[i] += array[j]
    return cdfArray

# map cdf to intensity values
def transformation(array):
    transformed = [0] * 256

    for i in range(len(array)):

        transformed[i] = round(array[i] * (L - 1))

    return transformed


# plot the histogram

def showAndSaveHistogram(array, string):

    plt.title("Intensity Histogram")

    plt.xlabel("Intensity")

    plt.ylabel("Frequency")

    plt.bar([i for i in range(256)], array)

    plt.savefig(string + " histogram")

    plt.show()


# apply histogram equalization to the input image

def outputImage(array, img):

    pix = img.load()

    M, N = imageProperties(img)


    # iterate the pixels

    for x in range(M):

        for y in range(N):

            # change the intensity of the pixel to the transformed one

            pix[x, y] = array[pix[x, y]]


    return img




# open image
```

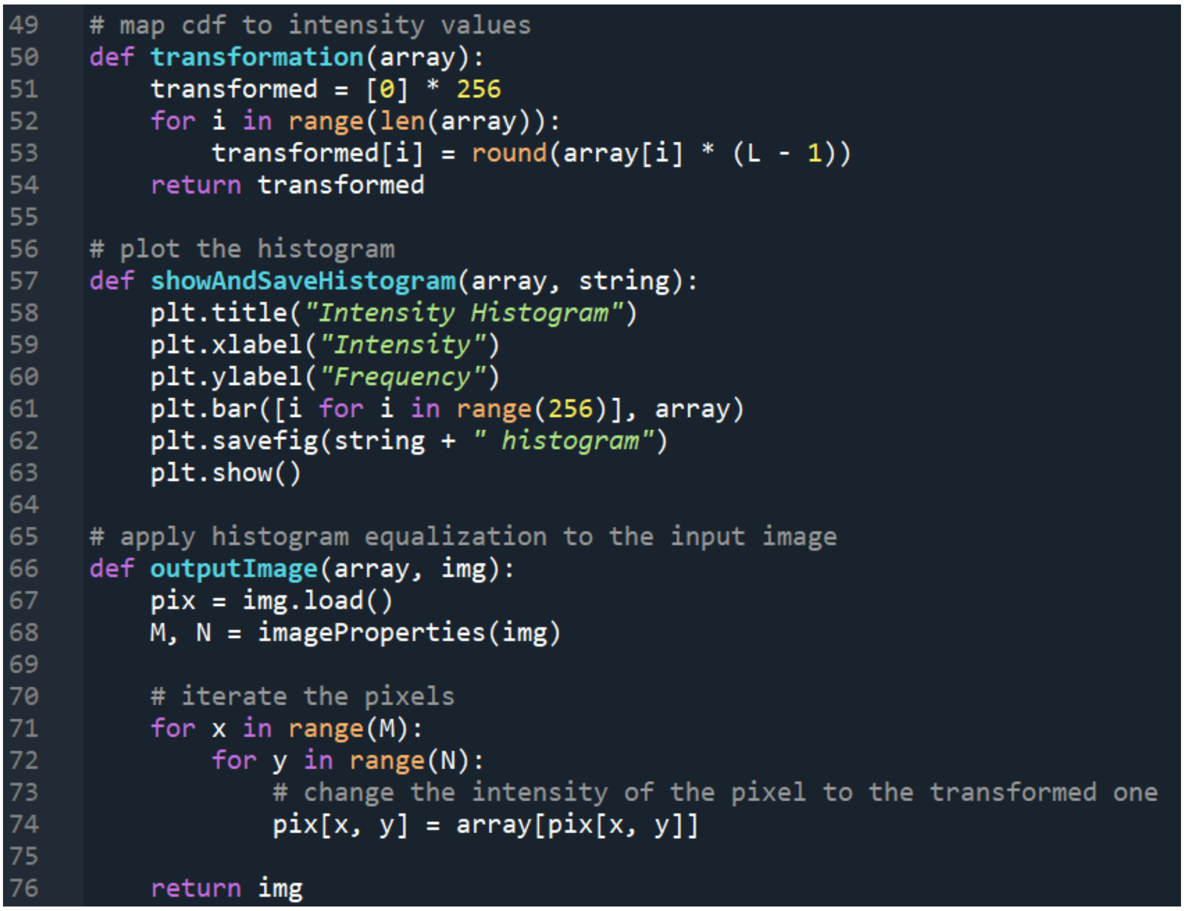
```
image = Image.open("lab07_img.png").convert('L')
histArray = populateCountArray(image)
pdfArray = pdf(histArray, image)
cdfArray = cdf(pdfArray)
transformed = transformation(cdfArray)
# save and show the histogram for the input image
showAndSaveHistogram(histArray, "input")
# get output image
output = outputImage(transformed, image)
histOutput = populateCountArray(image)
# save and show the histogram for the output image
showAndSaveHistogram(histOutput, "output")
# save output image
output.save("Transformed Image.jpg")
```



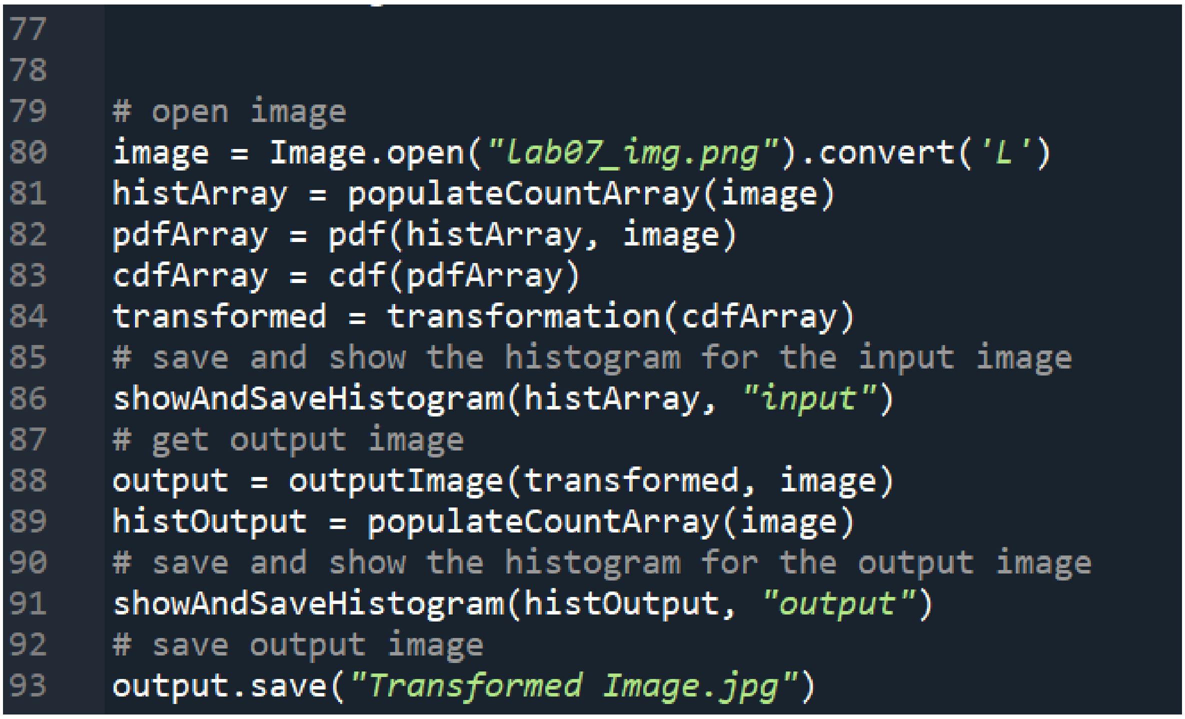
```python
from PIL import Image
from matplotlib import pyplot as plt

L = 256

# returns the width and height of the image
def imageProperties(img):
    return img.size

# returns the number of pixels of each intensity level
def populateCountArray(img):
    countArray = [0] * 256

    # load the pixels
    pix = img.load()

    # M is the width and N is the height of the image
    M, N = imageProperties(img)

    # iterate through the pixels
    for x in range(M):
        for y in range(N):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1

    return countArray

```



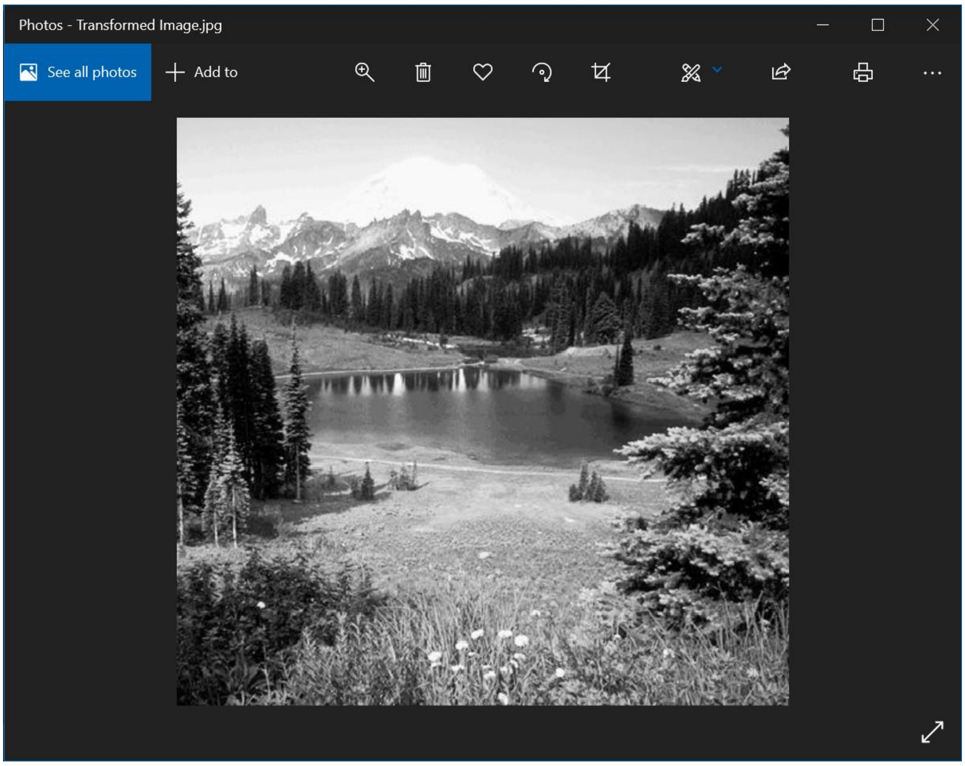
```python
# probability distribution function
def pdf(array, img):
    pdfArray = [0] * 256
    M, N = imageProperties(img)

    # pdf = frequency / total no. of pixels
    for i in range(len(array)):
        # M * N is the total no. of pixels
        pdfArray[i] = array[i] / (M * N)

    return pdfArray
# cummulative frequency distribution function
def cdf(array):
    cdfArray = [0] * 256

    # at every index, find the sum of current and all previous pdfs
    for i in range(len(array)):
        for j in range(i):
            cdfArray[i] += array[j]
    return cdfArray

```

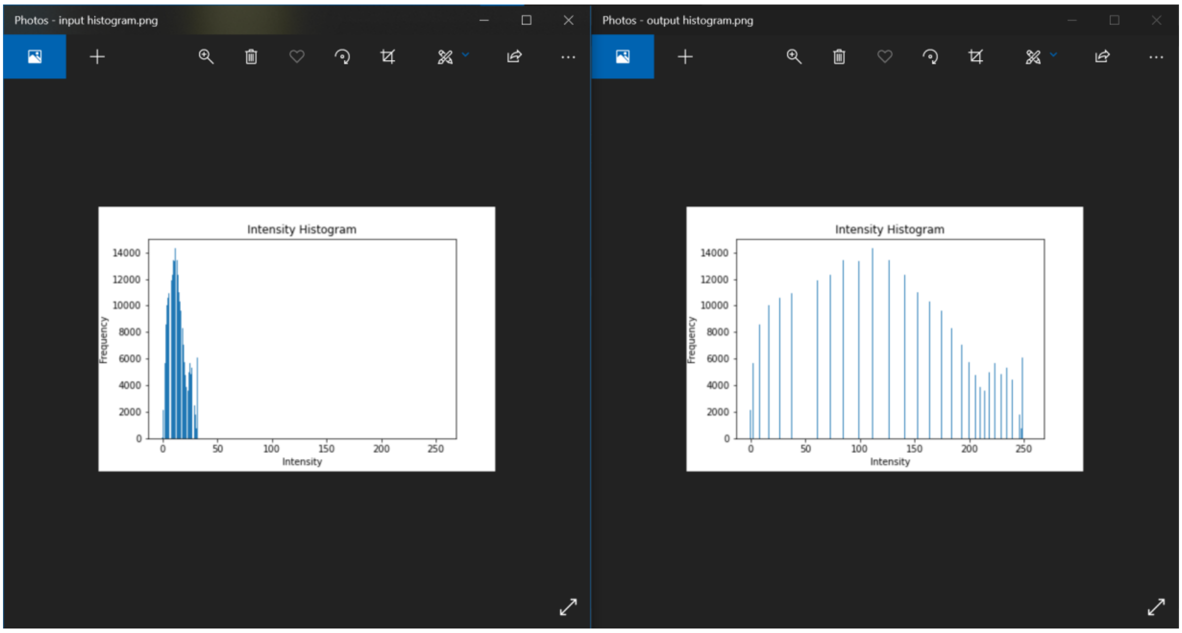


```python
# map cdf to intensity values
def transformation(array):
    transformed = [0] * 256
    for i in range(len(array)):
        transformed[i] = round(array[i] * (L - 1))
    return transformed

# plot the histogram
def showAndSaveHistogram(array, string):
    plt.title("Intensity Histogram")
    plt.xlabel("Intensity")
    plt.ylabel("Frequency")
    plt.bar([i for i in range(256)], array)
    plt.savefig(string + " histogram")
    plt.show()

# apply histogram equalization to the input image
def outputImage(array, img):
    pix = img.load()
    M, N = imageProperties(img)

    # iterate the pixels
    for x in range(M):
        for y in range(N):
            # change the intensity of the pixel to the transformed one
            pix[x, y] = array[pix[x, y]]

    return img
```

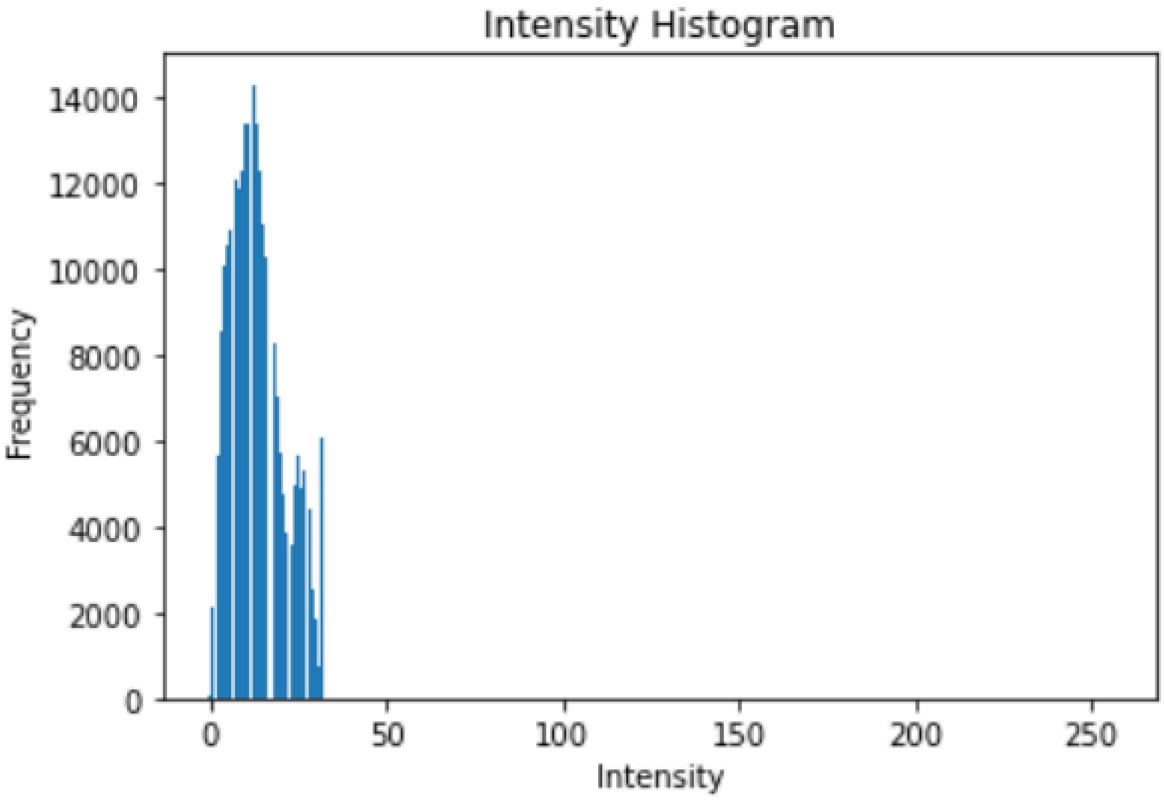


```python


# open image
image = Image.open("Lab07_img.png").convert('L')
histArray = populateCountArray(image)
pdfArray = pdf(histArray, image)
cdfArray = cdf(pdfArray)
transformed = transformation(cdfArray)
# save and show the histogram for the input image
showAndSaveHistogram(histArray, "input")
# get output image
output = outputImage(transformed, image)
histOutput = populateCountArray(image)
# save and show the histogram for the output image
showAndSaveHistogram(histOutput, "output")
# save output image
output.save("Transformed Image.jpg")
```

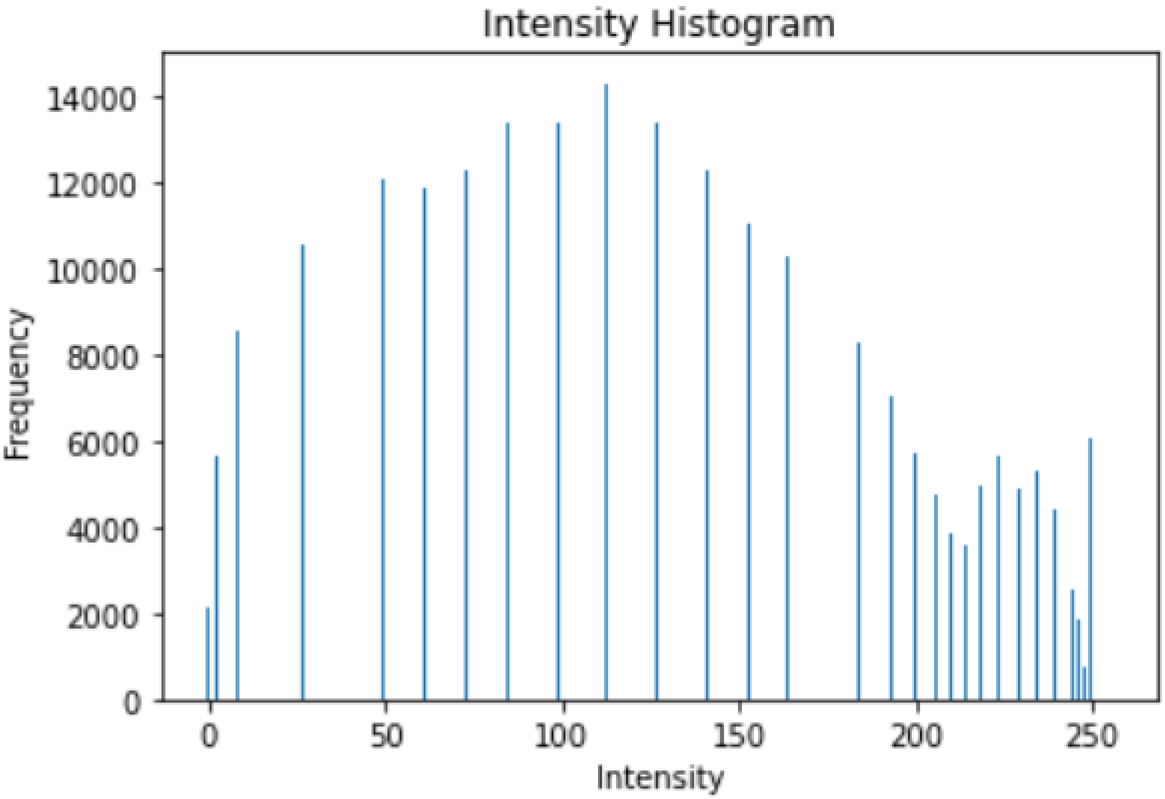




Before:



After:



Task 2

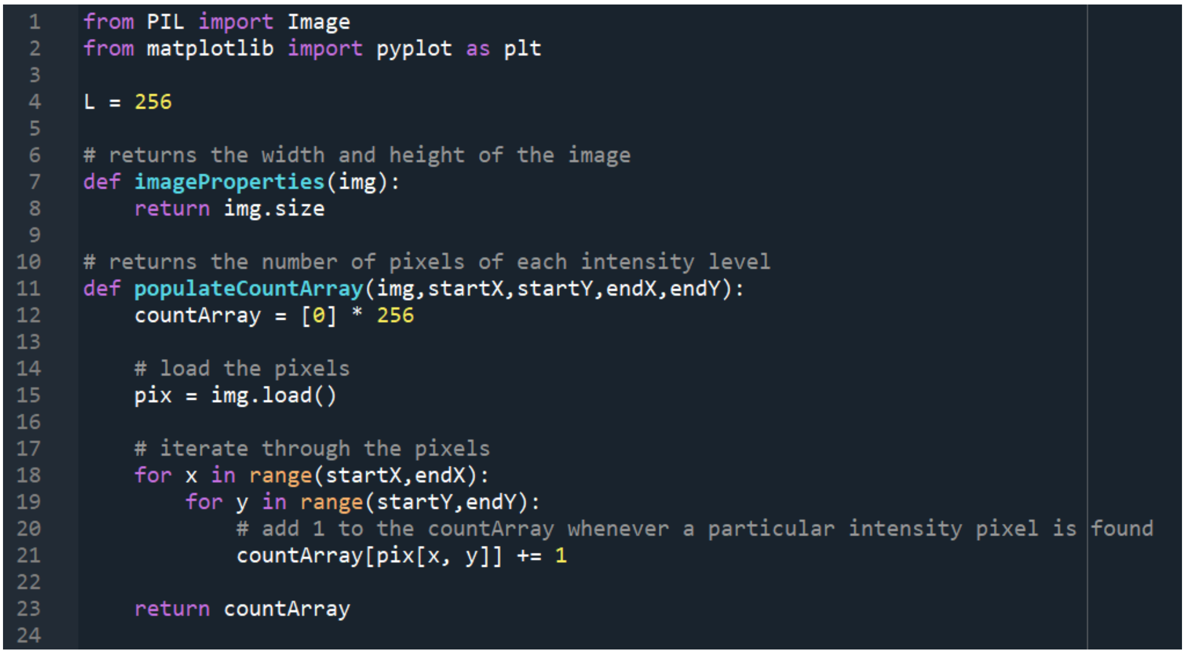
Local Histogram Equalization

Tiling

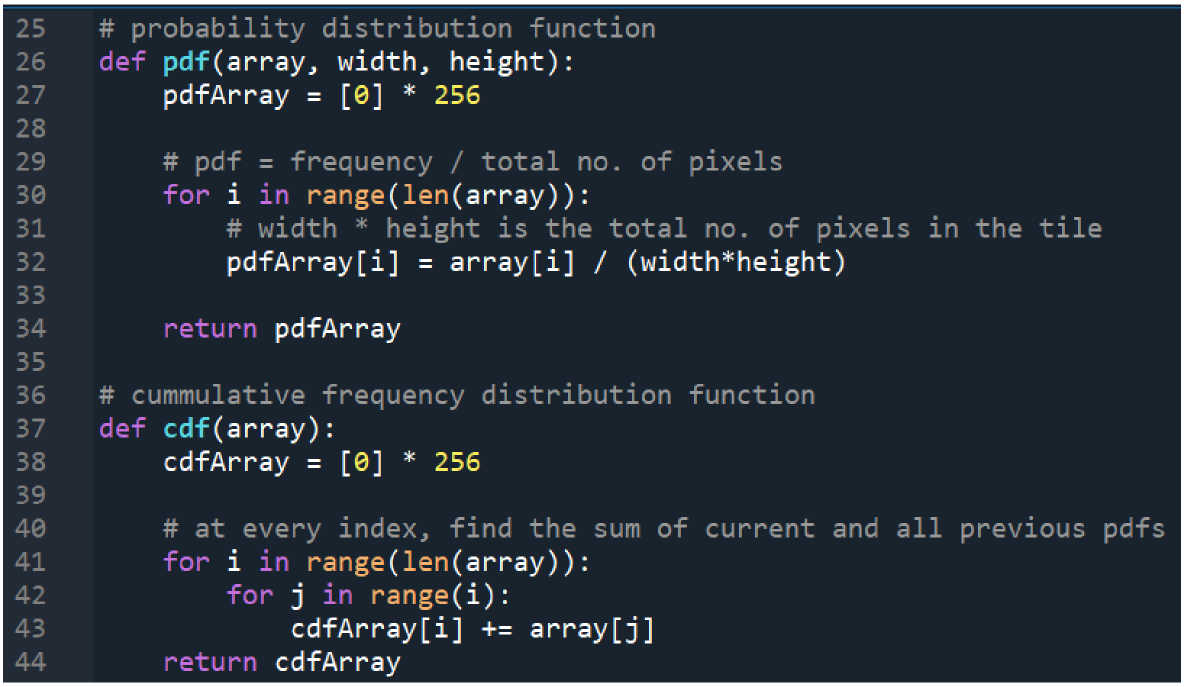
CODE

```
from PIL import Image

from matplotlib import pyplot as plt


L = 256


# returns the width and height of the image

def imageProperties(img):

    return img.size
```

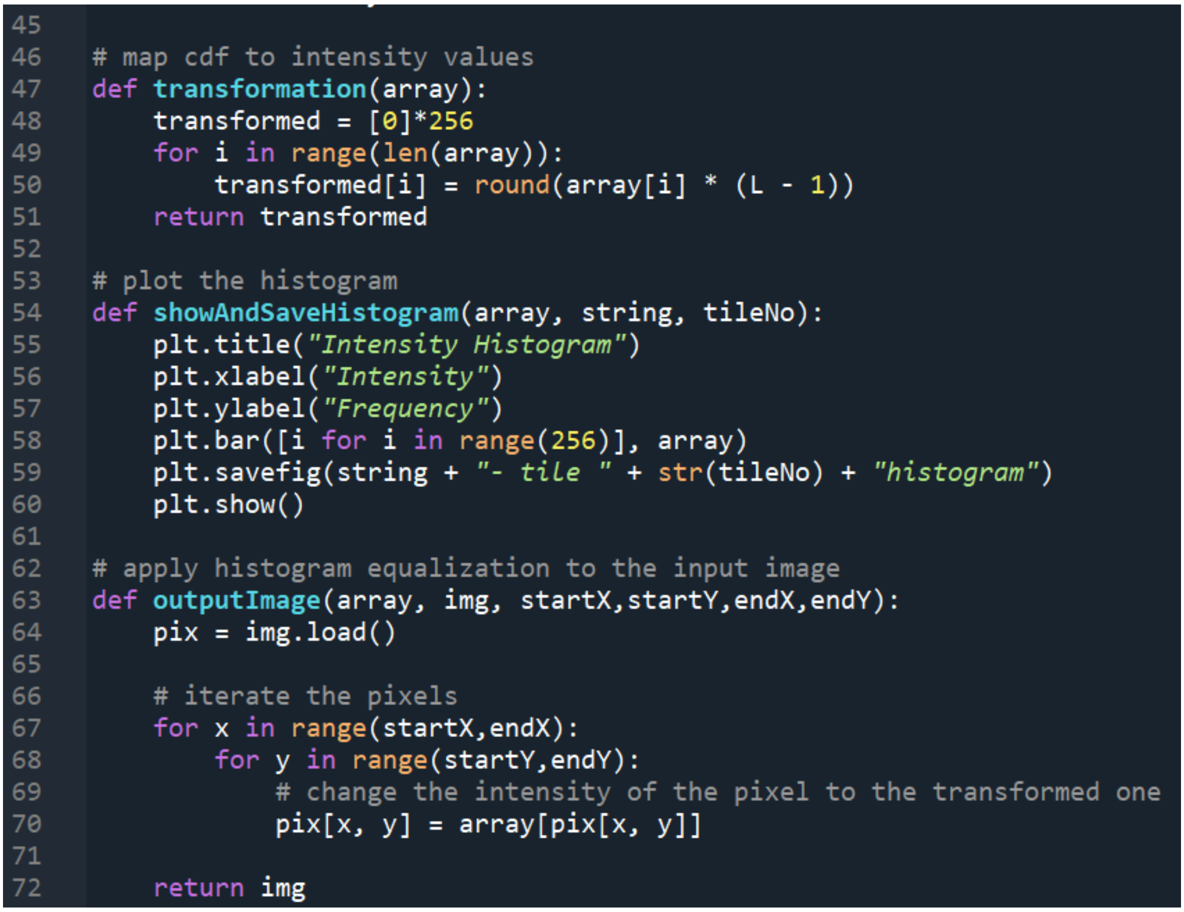
```
# returns the number of pixels of each intensity level
def populateCountArray(img,startX,startY,endX,endY):
    countArray = [0] * 256

    # load the pixels
    pix = img.load()

    # iterate through the pixels
    for x in range(startX,endX):
        for y in range(startY,endY):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1

    return countArray

# probability distribution function
def pdf(array, width, height):
    pdfArray = [0] * 256

    # pdf = frequency / total no. of pixels
    for i in range(len(array)):
        # width * height is the total no. of pixels in the tile
        pdfArray[i] = array[i] / (width*height)

    return pdfArray

# cummulative frequency distribution function
def cdf(array):
    cdfArray = [0] * 256
    # at every index, find the sum of current and all previous pdfs
    for i in range(len(array)):
        for j in range(i):
            cdfArray[i] += array[j]
    return cdfArray

# map cdf to intensity values
def transformation(array):
    transformed = [0]*256
    for i in range(len(array)):
        transformed[i] = round(array[i] * (L - 1))
    return transformed

# plot the histogram
def showAndSaveHistogram(array, string, tileNo):
    plt.title("Intensity Histogram")
    plt.xlabel("Intensity")
    plt.ylabel("Frequency")
    plt.bar([i for i in range(256)], array)
    plt.savefig(string + "- tile " + str(tileNo) + "histogram")
    plt.show()

# apply histogram equalization to the input image
def outputImage(array, img, startX,startY,endX,endY):
    pix = img.load()

    # iterate the pixels
    for x in range(startX,endX):
        for y in range(startY,endY):
            # change the intensity of the pixel to the transformed one
            pix[x, y] = array[pix[x, y]]

    return img



# open image
image = Image.open("lab07_img.png").convert('L')
M, N = image.size

currentTile = 0

for i in range(2):
    for j in range(2):
        # histogram of input tile
        tile = populateCountArray(image, j*int(M/2), i*int(N/2), (j+1)*int(M/2), (i+1)*int(N/2))
        pdfArray = pdf(tile,M/2,N/2)
        cdfArray = cdf(pdfArray)
        transformed = transformation(cdfArray)
        # for naming reasons
        currentTile += 1
        # showing and saving the input tile's histogram
        showAndSaveHistogram(tile, "input", currentTile)
        # map the transformed array to an output image of that tile
        output = outputImage(transformed, image, j*int(M/2), i*int(N/2), (j+1)*int(M/2), (i+1)*int(N/2))
         # histogram of output tile
        outputHist = populateCountArray(output, j*int(M/2), i*int(N/2), (j+1)*int(M/2), (i+1)*int(N/2))
        # showing and saving the output tile's histogram

        showAndSaveHistogram(outputHist, "output", currentTile)



output.save("Task2.png")
```



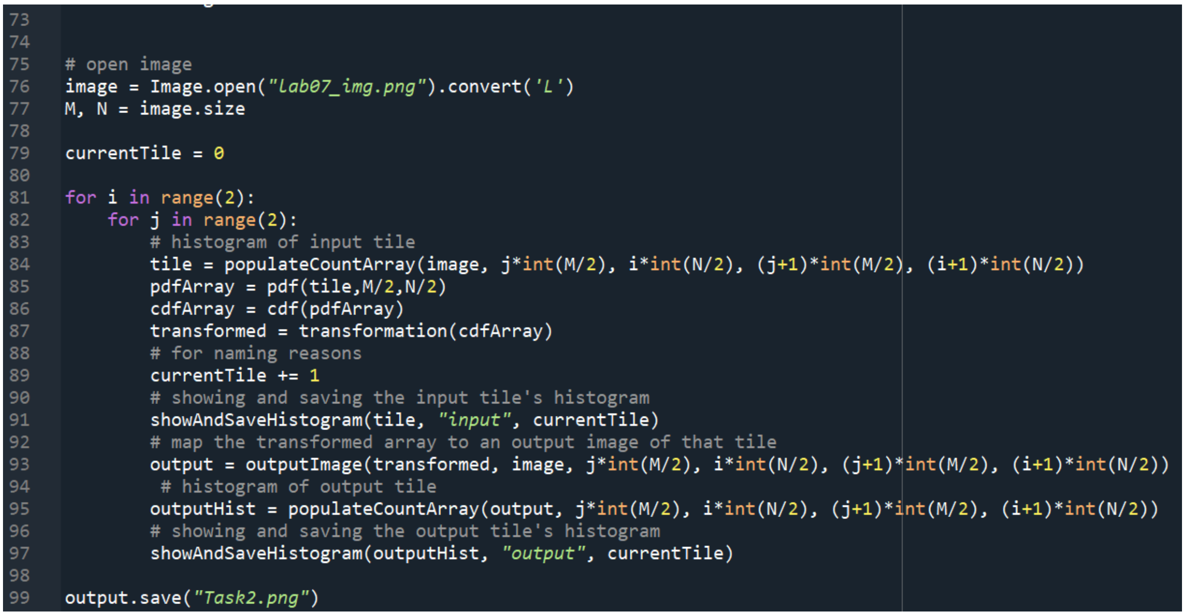
```python
from PIL import Image
from matplotlib import pyplot as plt

L = 256

# returns the width and height of the image
def imageProperties(img):
    return img.size

# returns the number of pixels of each intensity level
def populateCountArray(img,startX,startY,endX,endY):
    countArray = [0] * 256

    # load the pixels
    pix = img.load()

    # iterate through the pixels
    for x in range(startX,endX):
        for y in range(startY,endY):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1

    return countArray

```



```python
# probability distribution function
def pdf(array, width, height):
    pdfArray = [0] * 256

    # pdf = frequency / total no. of pixels
    for i in range(len(array)):
        # width * height is the total no. of pixels in the tile
        pdfArray[i] = array[i] / (width*height)

    return pdfArray

# cummulative frequency distribution function
def cdf(array):
    cdfArray = [0] * 256

    # at every index, find the sum of current and all previous pdfs
    for i in range(len(array)):
        for j in range(i):
            cdfArray[i] += array[j]
    return cdfArray
```



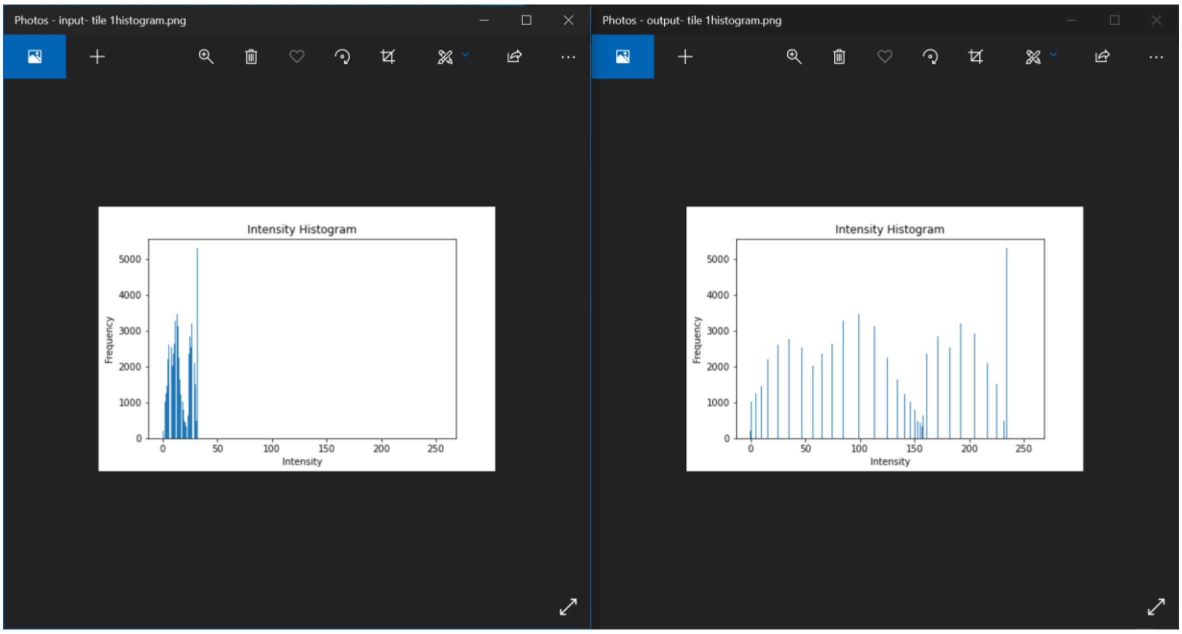
```python

# map cdf to intensity values
def transformation(array):
    transformed = [0]*256
    for i in range(len(array)):
        transformed[i] = round(array[i] * (L - 1))
    return transformed

# plot the histogram
def showAndSaveHistogram(array, string, tileNo):
    plt.title("Intensity Histogram")
    plt.xlabel("Intensity")
    plt.ylabel("Frequency")
    plt.bar([i for i in range(256)], array)
    plt.savefig(string + "- tile " + str(tileNo) + "histogram")
    plt.show()

# apply histogram equalization to the input image
def outputImage(array, img, startX,startY,endX,endY):
    pix = img.load()

    # iterate the pixels
    for x in range(startX,endX):
        for y in range(startY,endY):
            # change the intensity of the pixel to the transformed one
            pix[x, y] = array[pix[x, y]]

    return img
```



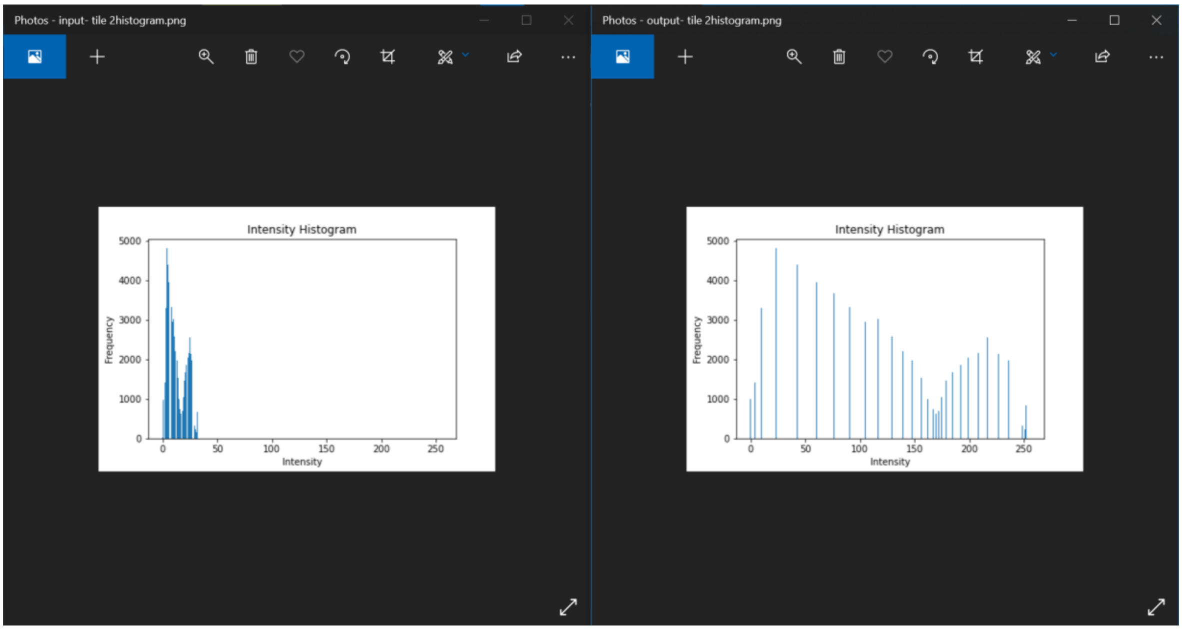
```python


# open image
image = Image.open("Lab07_img.png").convert('L')
M, N = image.size

currentTile = 0

for i in range(2):
    for j in range(2):
        # histogram of input tile
        tile = populateCountArray(image, j*int(M/2), i*int(N/2), (j+1)*int(M/2), (i+1)*int(N/2))
        pdfArray = pdf(tile,M/2,N/2)
        cdfArray = cdf(pdfArray)
        transformed = transformation(cdfArray)
        # for naming reasons
        currentTile += 1
        # showing and saving the input tile's histogram
        showAndSaveHistogram(tile, "input", currentTile)
        # map the transformed array to an output image of that tile
        output = outputImage(transformed, image, j*int(M/2), i*int(N/2), (j+1)*int(M/2), (i+1)*int(N/2))
         # histogram of output tile
        outputHist = populateCountArray(output, j*int(M/2), i*int(N/2), (j+1)*int(M/2), (i+1)*int(N/2))
        # showing and saving the output tile's histogram
        showAndSaveHistogram(outputHist, "output", currentTile)

output.save("Task2.png")
```

HISTOGRAMS

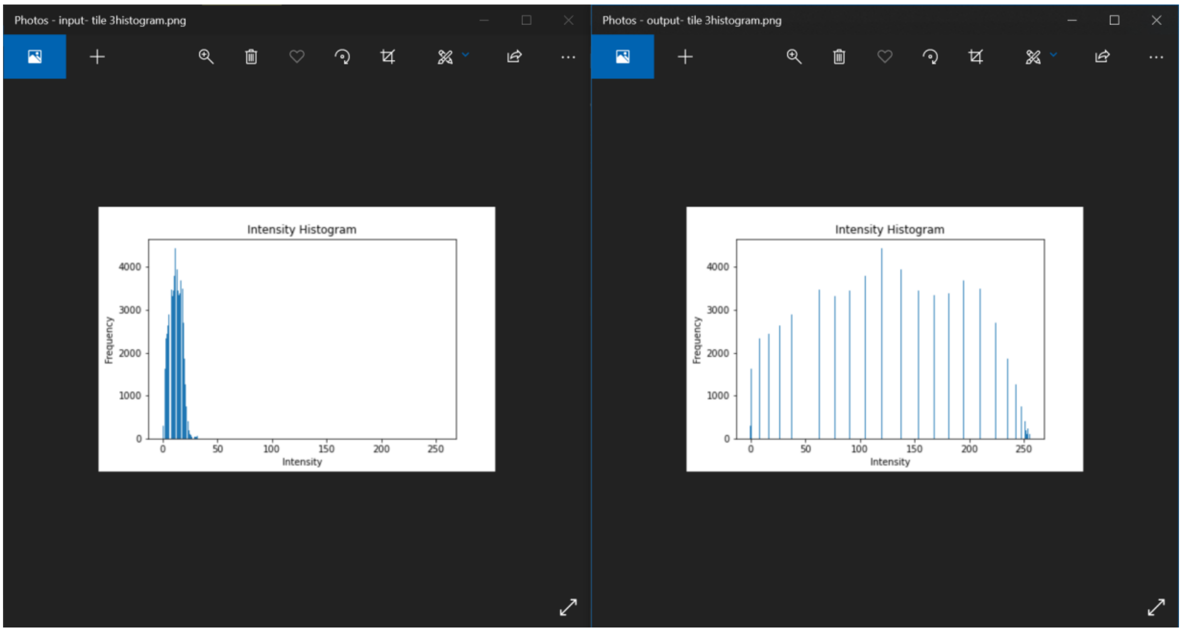
Tile 1


Photos - input- tile 1histogram.png
Intensity Histogram
Frequency
Intensity
0
1000
2000
3000
4000
5000
0
50
100
150
200
250
Photos - output- tile 1histogram.png
Intensity Histogram
Frequency
Intensity
0
1000
2000
3000
4000
5000
0
50
100
150
200
250

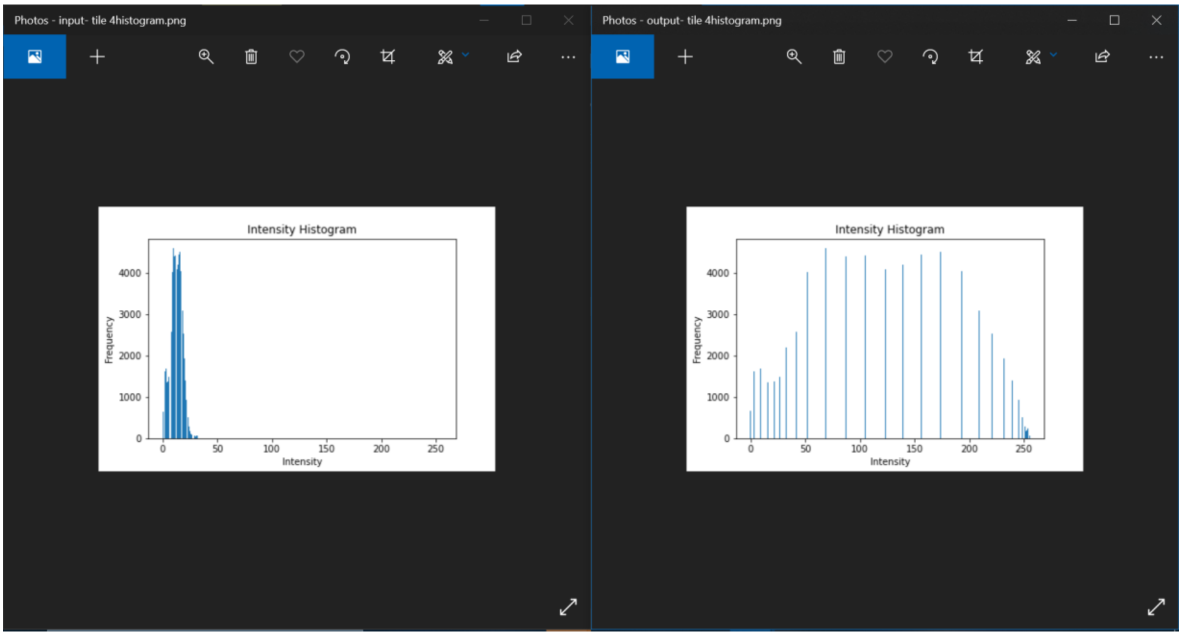
Tile 2


Photos - input- tile 2histogram.png
Intensity Histogram
Frequency
Intensity
0
1000
2000
3000
4000
5000
0
50
100
150
200
250
Photos - output- tile 2histogram.png
Intensity Histogram
Frequency
Intensity
0
1000
2000
3000
4000
5000
0
50
100
150
200
250

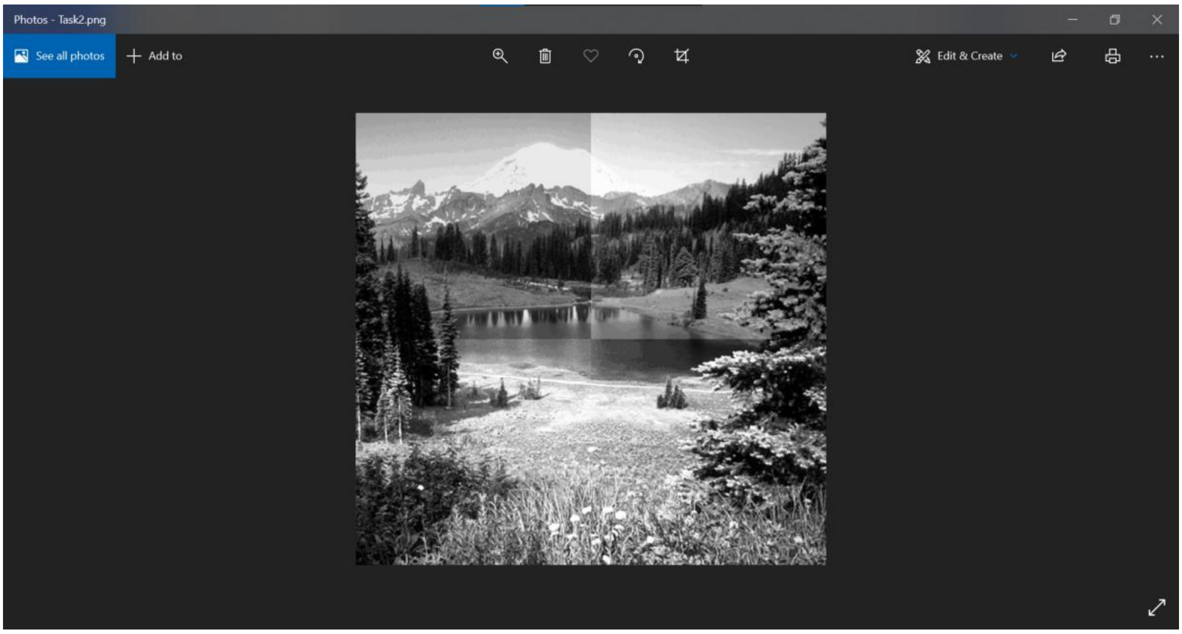
Tile 3



Tile 4



OUTPUT

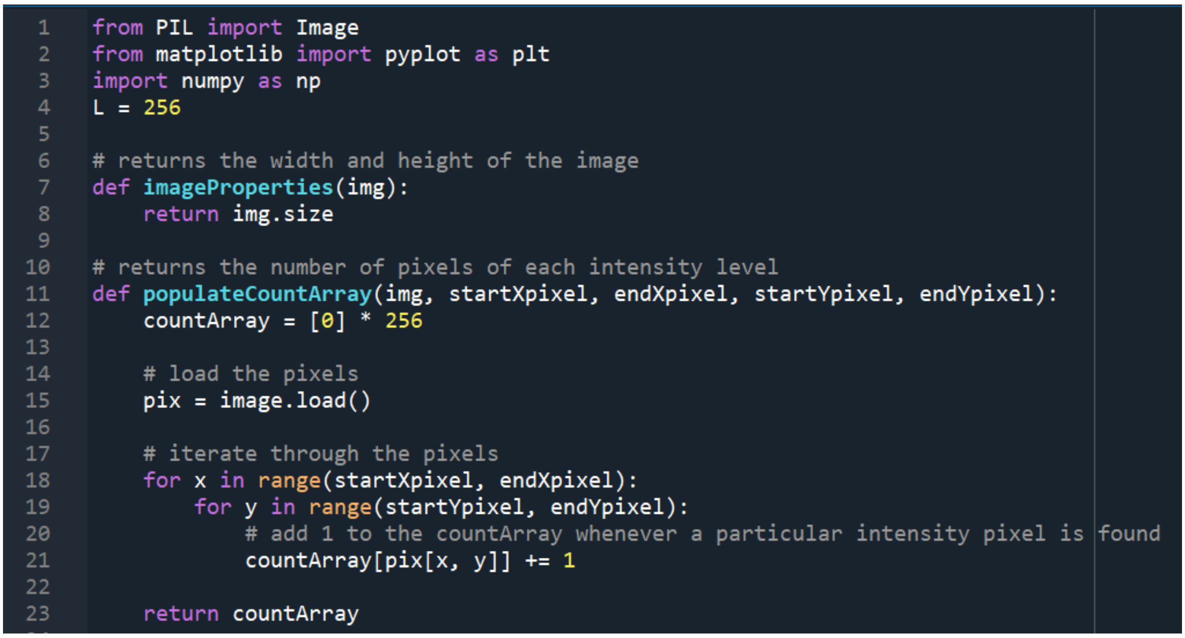


Sliding Window

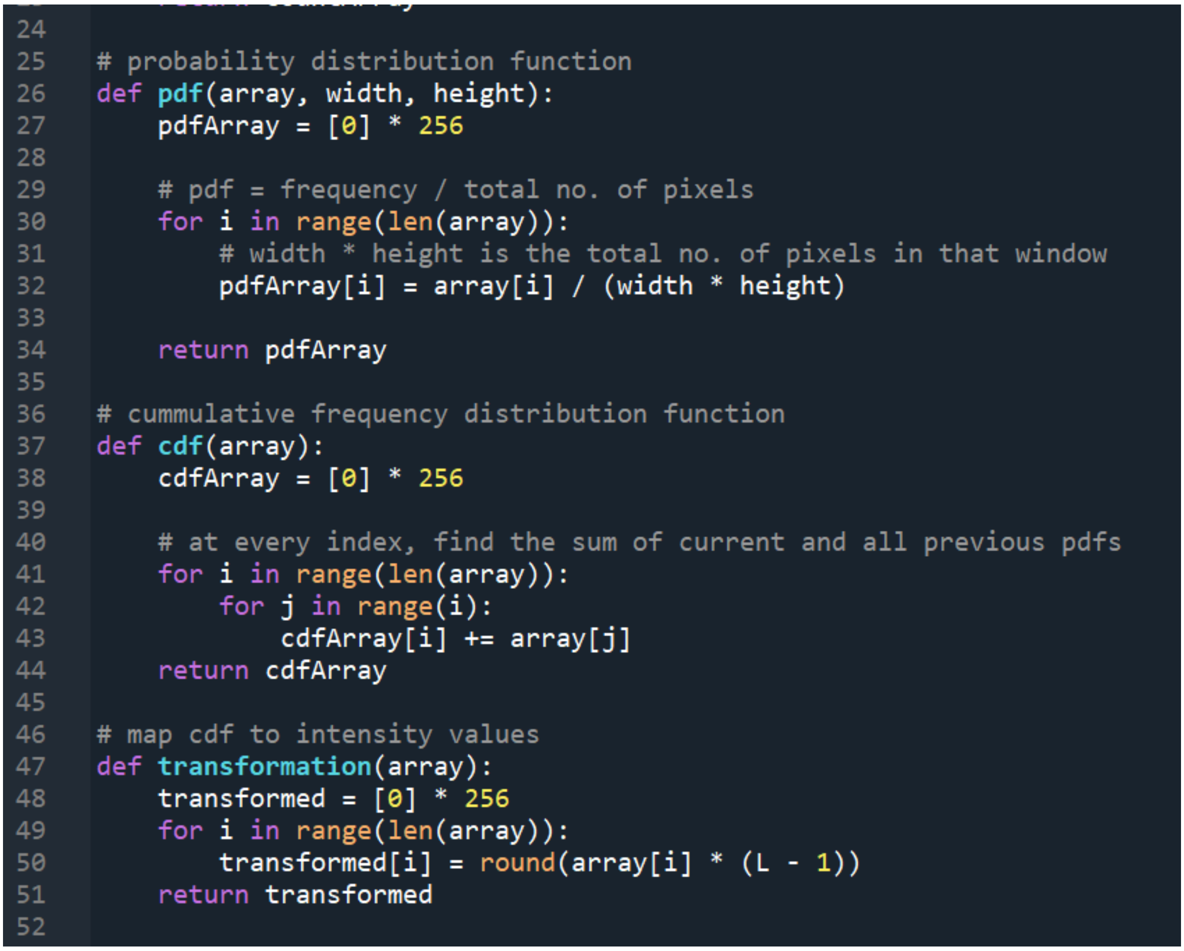
Saving and showing histograms for so many windows takes a lot of time. So we resized the image and saved and showed the histograms for the windows. For the larger image, we displayed two histograms, one at the start and one at the end.

CODE (to save and show histogram for every window)

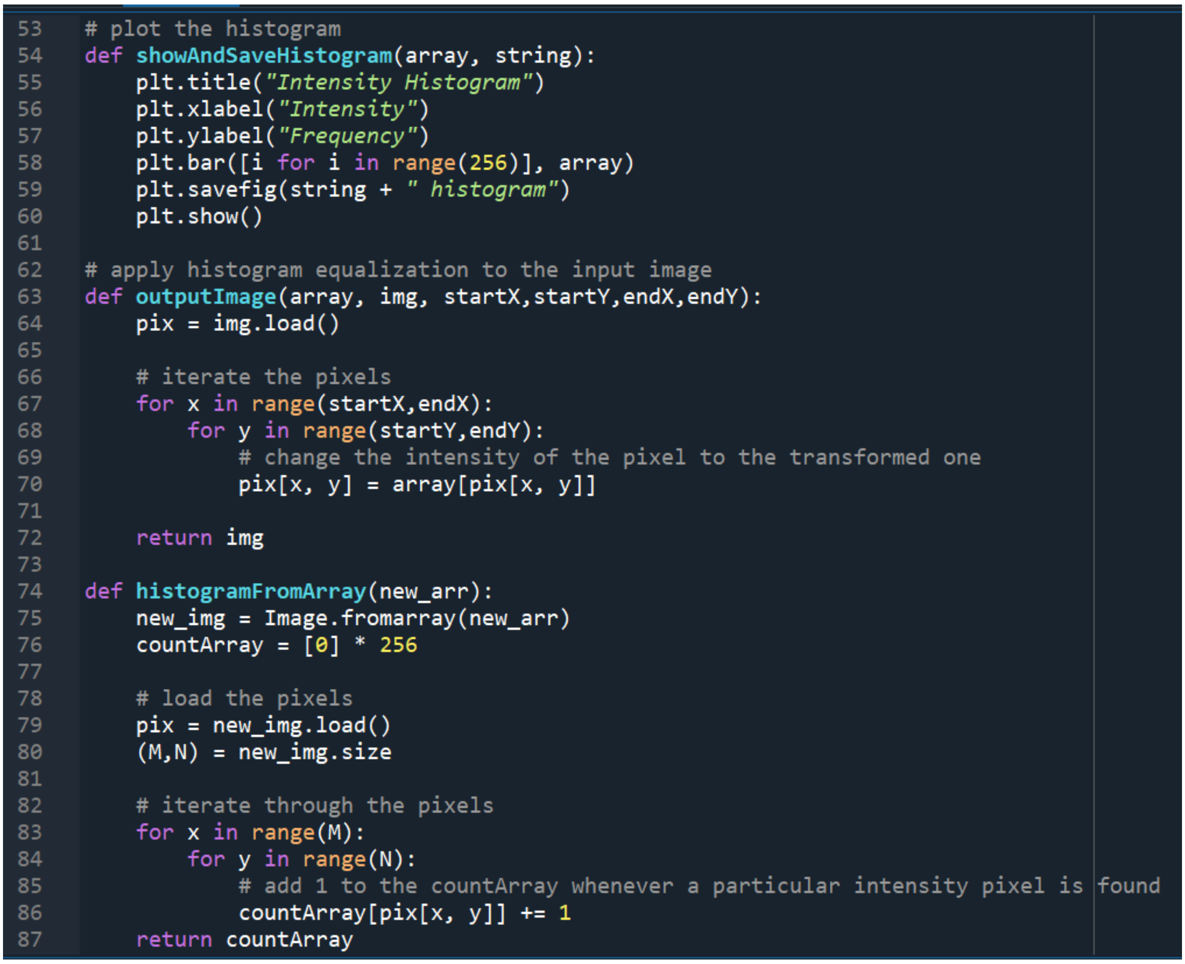
```
from PIL import Image

from matplotlib import pyplot as plt

import numpy as np

L = 256


# returns the width and height of the image

def imageProperties(img):

    return img.size


# returns the number of pixels of each intensity level

def populateCountArray(img, startXpixel, endXpixel, startYpixel, endYpixel):

    countArray = [0] * 256
    # load the pixels
    pix = image.load()

    # iterate through the pixels
    for x in range(startXpixel, endXpixel):
        for y in range(startYpixel, endYpixel):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1

    return countArray

# probability distribution function
def pdf(array, width, height):
    pdfArray = [0] * 256

    # pdf = frequency / total no. of pixels
    for i in range(len(array)):
        # width * height is the total no. of pixels in that window
        pdfArray[i] = array[i] / (width * height)

    return pdfArray

# cummulative frequency distribution function
def cdf(array):
    cdfArray = [0] * 256

    # at every index, find the sum of current and all previous pdfs
    for i in range(len(array)):
        for j in range(i):
            cdfArray[i] += array[j]
    return cdfArray

# map cdf to intensity values
def transformation(array):
    transformed = [0] * 256
    for i in range(len(array)):
        transformed[i] = round(array[i] * (L - 1))
    return transformed

# plot the histogram
def showAndSaveHistogram(array, string):
    plt.title("Intensity Histogram")
    plt.xlabel("Intensity")
    plt.ylabel("Frequency")
    plt.bar([i for i in range(256)], array)
    plt.savefig(string + " histogram")
    plt.show()

# apply histogram equalization to the input image
def outputImage(array, img, startX,startY,endX,endY):
    pix = img.load()

    # iterate the pixels
    for x in range(startX,endX):
        for y in range(startY,endY):
            # change the intensity of the pixel to the transformed one
            pix[x, y] = array[pix[x, y]]
    return img


def histogramFromArray(new_arr):
    new_img = Image.fromarray(new_arr)
    countArray = [0] * 256

    # load the pixels
    pix = new_img.load()
    (M,N) = new_img.size

    # iterate through the pixels
    for x in range(M):
        for y in range(N):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1
    return countArray



# open image
image = Image.open("lab07_img_resized.png").convert('L')
M, N = image.size



pix = image.load()
new_arr = np.asarray(image).copy()


for i in range(int(M/2)):
    for j in range(int(N/2)):
        histArray = populateCountArray(image, i, i + int(M/2), j, j + int(N/2))

        pdfArray = pdf(histArray, int(M/2), int(N/2))

        cdfArray = cdf(pdfArray)

        transformed = transformation(cdfArray)

        # save and show the histogram for the input image

        showAndSaveHistogram(histArray, "input")


        # get output image

        output = outputImage(transformed, image, i, j, i + int(M/2), j + int(N/2))

        histOutput = populateCountArray(output, i, i + int(M/2), j, j + int(N/2))

        showAndSaveHistogram(histOutput, "output")


        for x in range(i,i+int(M/2)):

            for y in range(j,j+int(N/2)):

                pix_val=pix[x,y]

                transform_val=transformed[pix_val]

                new_arr[y][x]=transform_val


        image = Image.open("lab07_img_resized.png").convert('L')


arr2 = histogramFromArray(new_arr)

showAndSaveHistogram(arr2, "full_image")


Image.fromarray(new_arr).show()
```



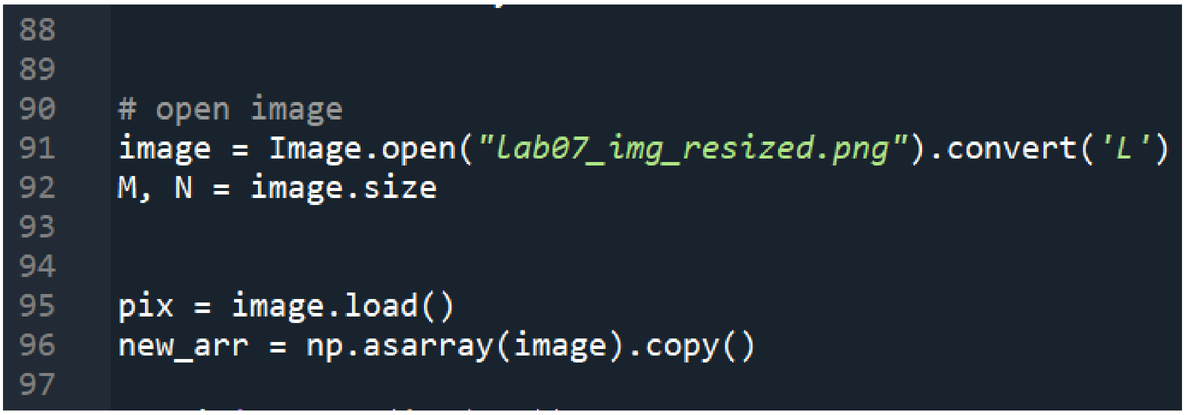
```python
from PIL import Image
from matplotlib import pyplot as plt
import numpy as np
L = 256

# returns the width and height of the image
def imageProperties(img):
    return img.size

# returns the number of pixels of each intensity level
def populateCountArray(img, startXpixel, endXpixel, startYpixel, endYpixel):
    countArray = [0] * 256

    # load the pixels
    pix = image.load()

    # iterate through the pixels
    for x in range(startXpixel, endXpixel):
        for y in range(startYpixel, endYpixel):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1

    return countArray
```



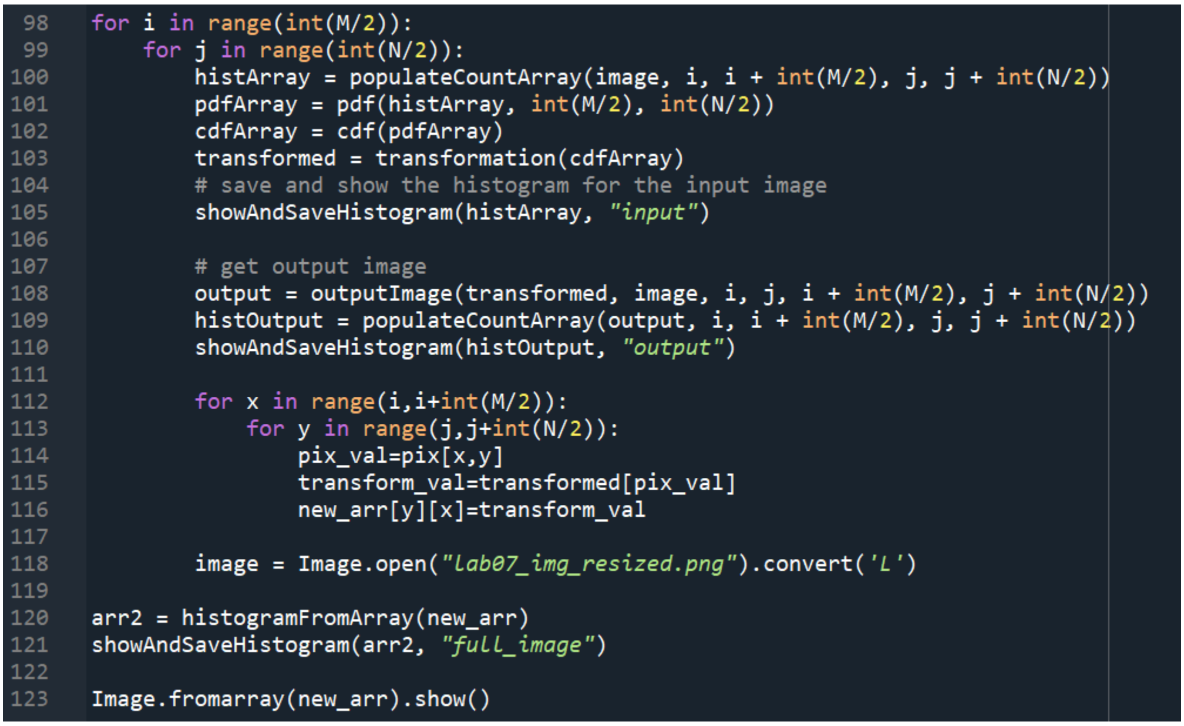
```python

# probability distribution function
def pdf(array, width, height):
    pdfArray = [0] * 256

    # pdf = frequency / total no. of pixels
    for i in range(len(array)):
        # width * height is the total no. of pixels in that window
        pdfArray[i] = array[i] / (width * height)

    return pdfArray

# cummulative frequency distribution function
def cdf(array):
    cdfArray = [0] * 256

    # at every index, find the sum of current and all previous pdfs
    for i in range(len(array)):
        for j in range(i):
            cdfArray[i] += array[j]
    return cdfArray

# map cdf to intensity values
def transformation(array):
    transformed = [0] * 256
    for i in range(len(array)):
        transformed[i] = round(array[i] * (L - 1))
    return transformed

```



```
# plot the histogram
def showAndSaveHistogram(array, string):
    plt.title("Intensity Histogram")
    plt.xlabel("Intensity")
    plt.ylabel("Frequency")
    plt.bar([i for i in range(256)], array)
    plt.savefig(string + " histogram")
    plt.show()

# apply histogram equalization to the input image
def outputImage(array, img, startX,startY,endX,endY):
    pix = img.load()

    # iterate the pixels
    for x in range(startX,endX):
        for y in range(startY,endY):
            # change the intensity of the pixel to the transformed one
            pix[x, y] = array[pix[x, y]]

    return img

def histogramFromArray(new_arr):
    new_img = Image.fromarray(new_arr)
    countArray = [0] * 256

    # load the pixels
    pix = new_img.load()
    (M,N) = new_img.size

    # iterate through the pixels
    for x in range(M):
        for y in range(N):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1
    return countArray
```

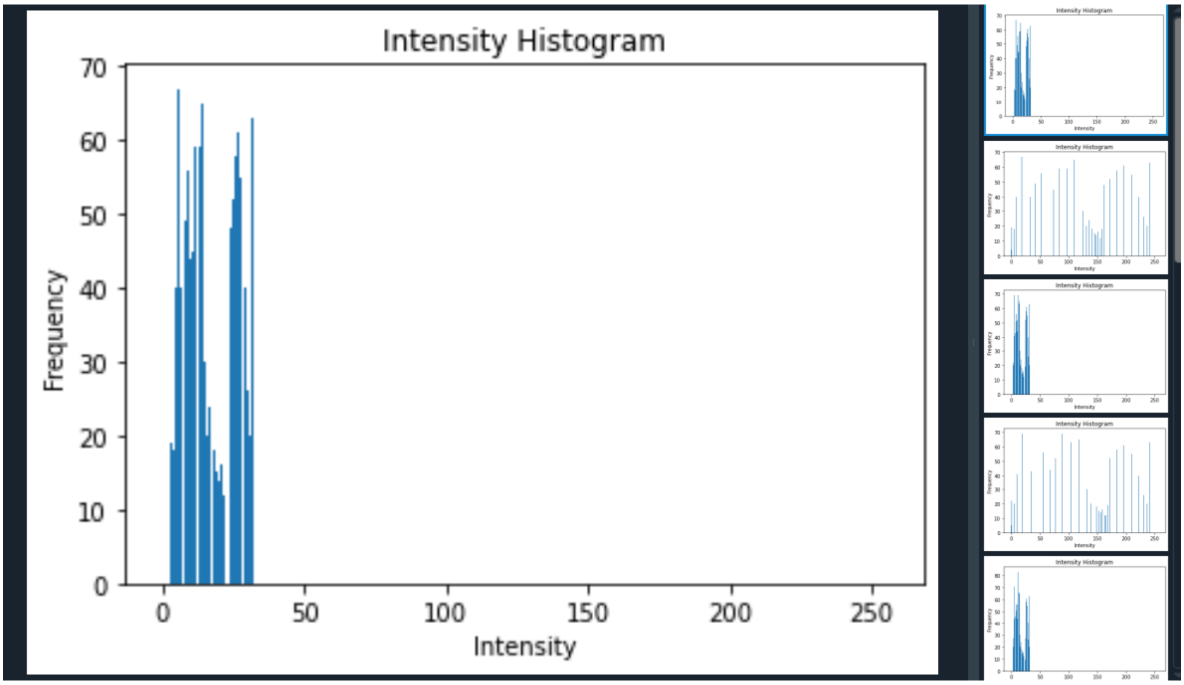


```


# open image
image = Image.open("Lab07_img_resized.png").convert('L')
M, N = image.size


pix = image.load()
new_arr = np.asarray(image).copy()

```

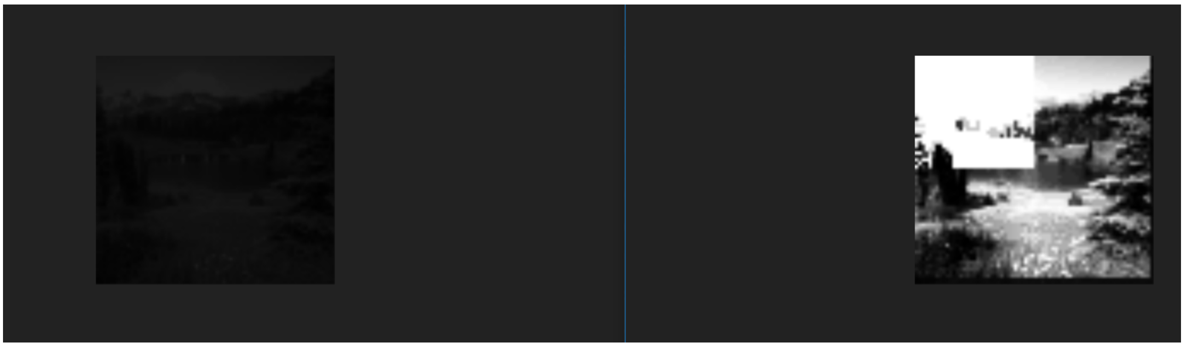


```
for i in range(int(M/2)):
    for j in range(int(N/2)):
        histArray = populateCountArray(image, i, i + int(M/2), j, j + int(N/2))
        pdfArray = pdf(histArray, int(M/2), int(N/2))
        cdfArray = cdf(pdfArray)
        transformed = transformation(cdfArray)
        # save and show the histogram for the input image
        showAndSaveHistogram(histArray, "input")

        # get output image
        output = outputImage(transformed, image, i, j, i + int(M/2), j + int(N/2))
        histOutput = populateCountArray(output, i, i + int(M/2), j, j + int(N/2))
        showAndSaveHistogram(histOutput, "output")

        for x in range(i,i+int(M/2)):
            for y in range(j,j+int(N/2)):
                pix_val=pix[x,y]
                transform_val=transformed[pix_val]
                new_arr[y][x]=transform_val

        image = Image.open("Lab07_img_resized.png").convert('L')

arr2 = histogramFromArray(new_arr)
showAndSaveHistogram(arr2, "full_image")

Image.fromarray(new_arr).show()
```

SOME OF THE HISTOGRAMS (can see on the right)



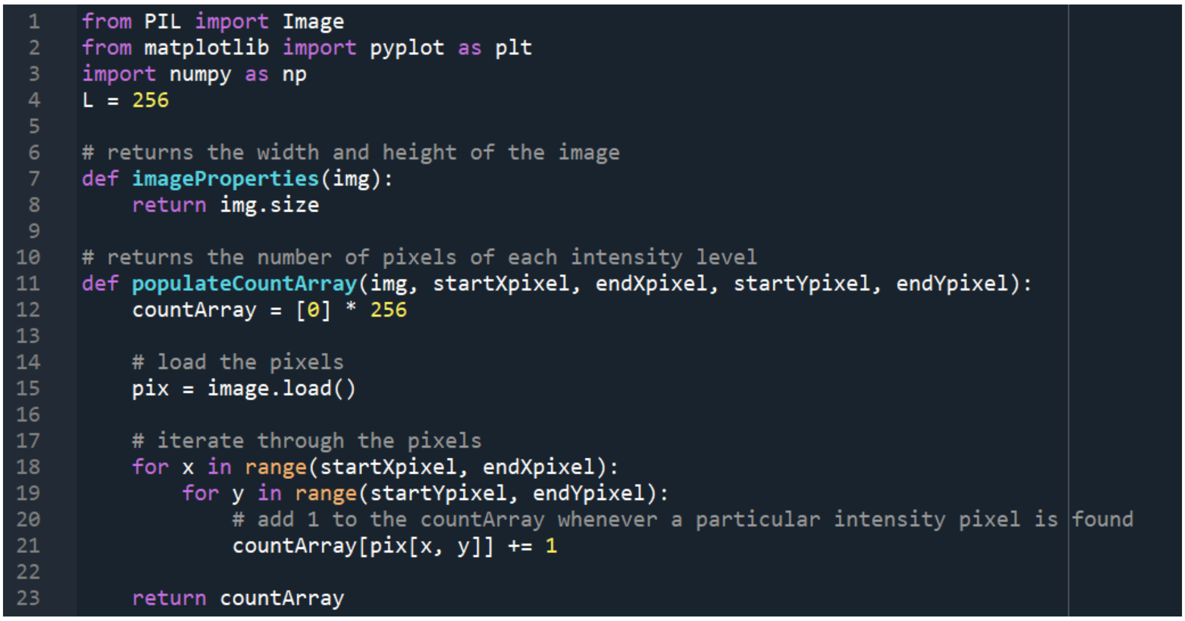
OUTPUT



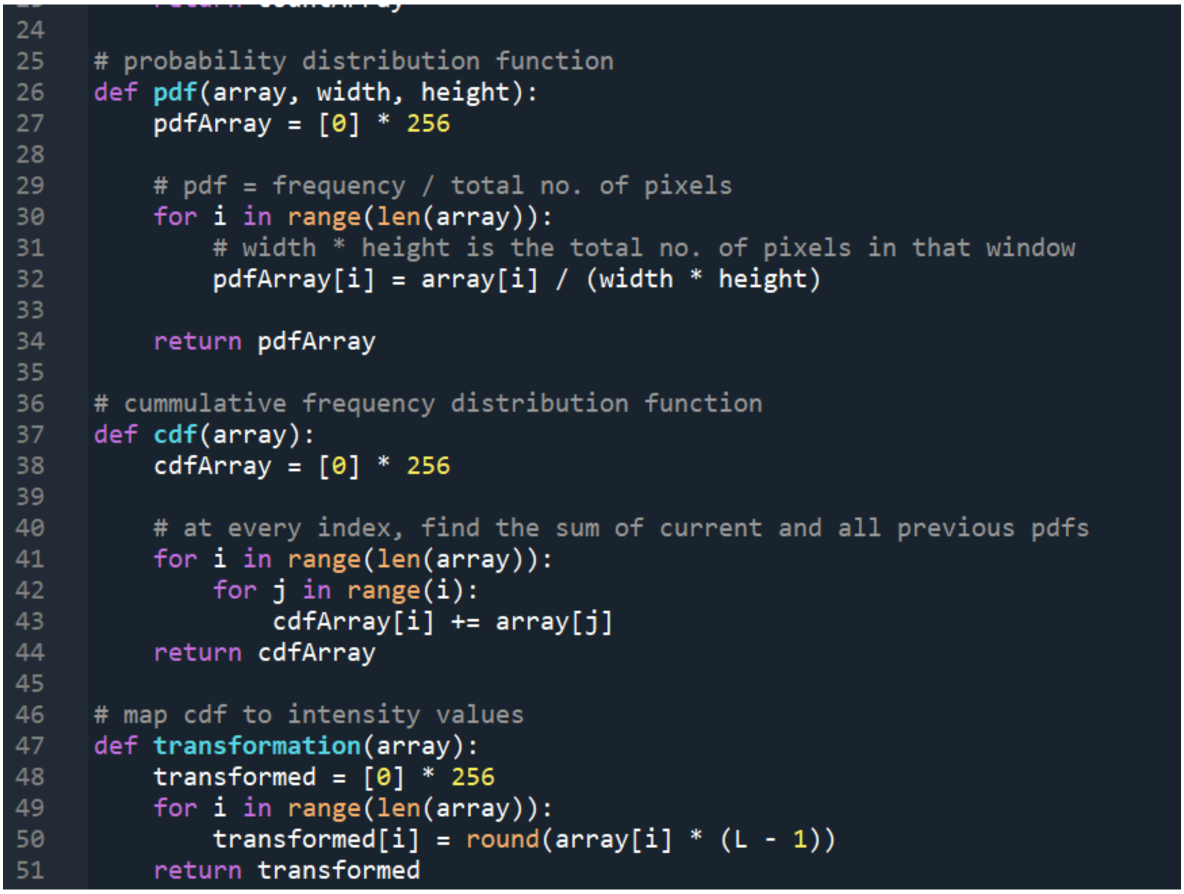
CODE (to save and show histogram only at the start and end)

```
from PIL import Image
from matplotlib import pyplot as plt
import numpy as np
L = 256

# returns the width and height of the image
def imageProperties(img):
    return img.size

# returns the number of pixels of each intensity level
def populateCountArray(img, startXpixel, endXpixel, startYpixel, endYpixel):
    countArray = [0] * 256

    # load the pixels
    pix = image.load()

    # iterate through the pixels
    for x in range(startXpixel, endXpixel):
        for y in range(startYpixel, endYpixel):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1
    return countArray


# probability distribution function
def pdf(array, width, height):
    pdfArray = [0] * 256


    # pdf = frequency / total no. of pixels
    for i in range(len(array)):
        # width * height is the total no. of pixels in that window
        pdfArray[i] = array[i] / (width * height)


    return pdfArray


# cummulative frequency distribution function
def cdf(array):
    cdfArray = [0] * 256


    # at every index, find the sum of current and all previous pdfs
    for i in range(len(array)):
        for j in range(i):
            cdfArray[i] += array[j]
    return cdfArray


# map cdf to intensity values
def transformation(array):
    transformed = [0] * 256
    for i in range(len(array)):
        transformed[i] = round(array[i] * (L - 1))
    return transformed


# plot the histogram
def showAndSaveHistogram(array, string):
    plt.title("Intensity Histogram")
    plt.xlabel("Intensity")
    plt.ylabel("Frequency")
    plt.bar([i for i in range(256)], array)
    plt.savefig(string + " histogram")
    plt.show()


# apply histogram equalization to the input image
def outputImage(array, img, startX,startY,endX,endY):
    pix = img.load()

    # iterate the pixels
    for x in range(startX,endX):
        for y in range(startY,endY):
            # change the intensity of the pixel to the transformed one
            pix[x, y] = array[pix[x, y]]

    return img


def histogramFromArray(new_arr):
    new_img = Image.fromarray(new_arr)
    countArray = [0] * 256

    # load the pixels
    pix = new_img.load()
    (M,N) = new_img.size


    # iterate through the pixels
    for x in range(M):
        for y in range(N):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1
    return countArray



# open image
image = Image.open("lab07_img.png").convert('L')
M, N = image.size

# histogram before equalization
inputHistArray = populateCountArray(image, 0, M, 0, N)
showAndSaveHistogram(inputHistArray, "input")

pix = image.load()
new_arr = np.asarray(image).copy()

for i in range(int(M/2)):
    for j in range(int(N/2)):
        histArray = populateCountArray(image, i, i + int(M/2), j, j + int(N/2))
        pdfArray = pdf(histArray, int(M/2), int(N/2))
        cdfArray = cdf(pdfArray)
        transformed = transformation(cdfArray)

        for x in range(i,i+int(M/2)):
```

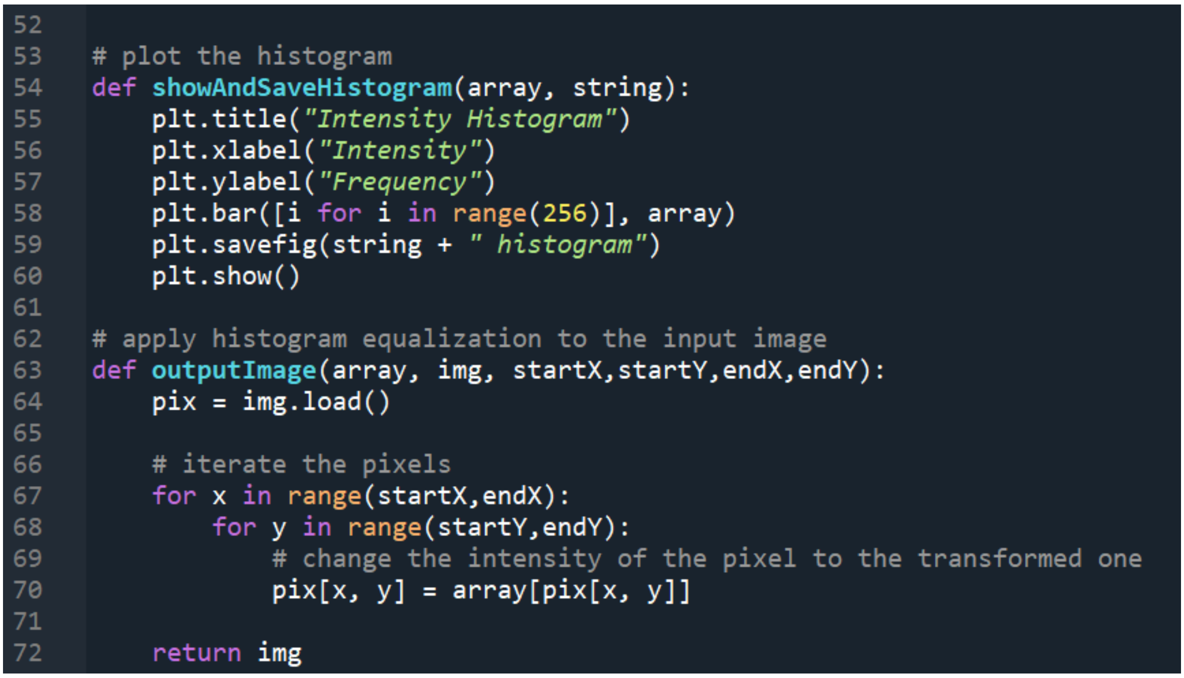
```
        for y in range(j,j+int(N/2)):
          pix_val=pix[x,y]
          transform_val=transformed[pix_val]
          new_arr[y][x]=transform_val


# histogram after equalization
arr2 = histogramFromArray(new_arr)
showAndSaveHistogram(arr2, "full_image")


Image.fromarray(new_arr).show()
```



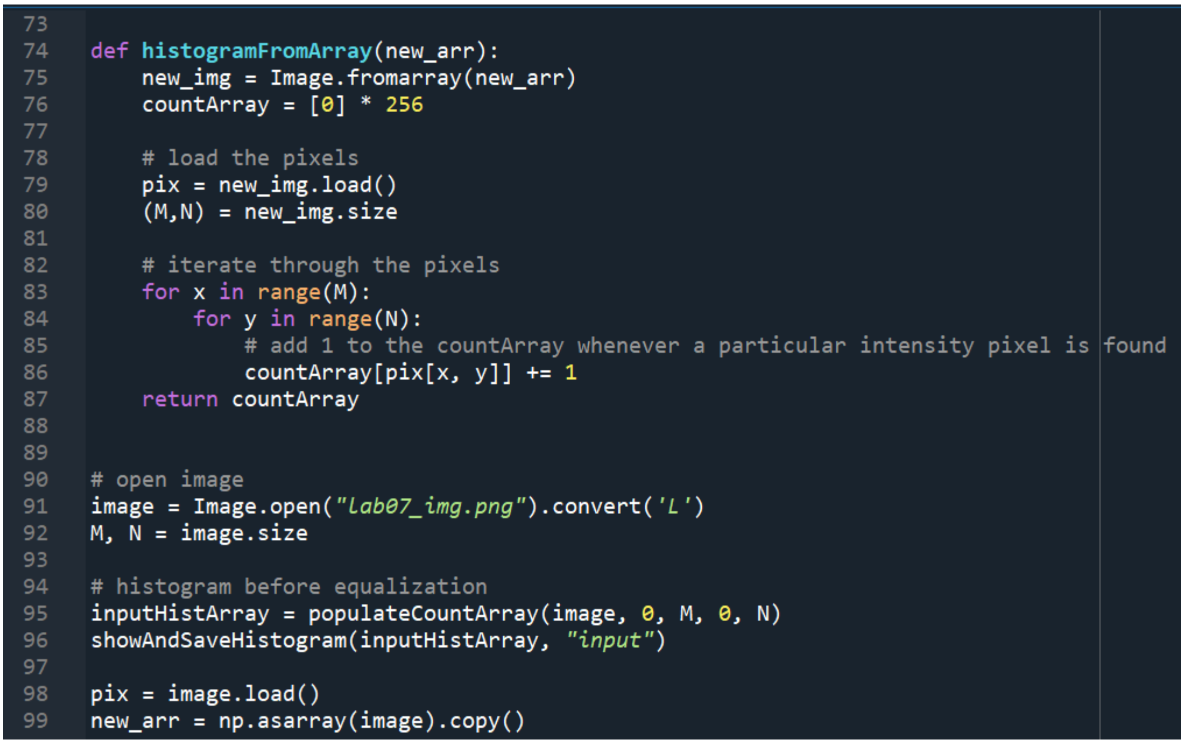
```
from PIL import Image
from matplotlib import pyplot as plt
import numpy as np
L = 256

# returns the width and height of the image
def imageProperties(img):
    return img.size

# returns the number of pixels of each intensity level
def populateCountArray(img, startXpixel, endXpixel, startYpixel, endYpixel):
    countArray = [0] * 256

    # load the pixels
    pix = image.load()

    # iterate through the pixels
    for x in range(startXpixel, endXpixel):
        for y in range(startYpixel, endYpixel):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1

    return countArray
```



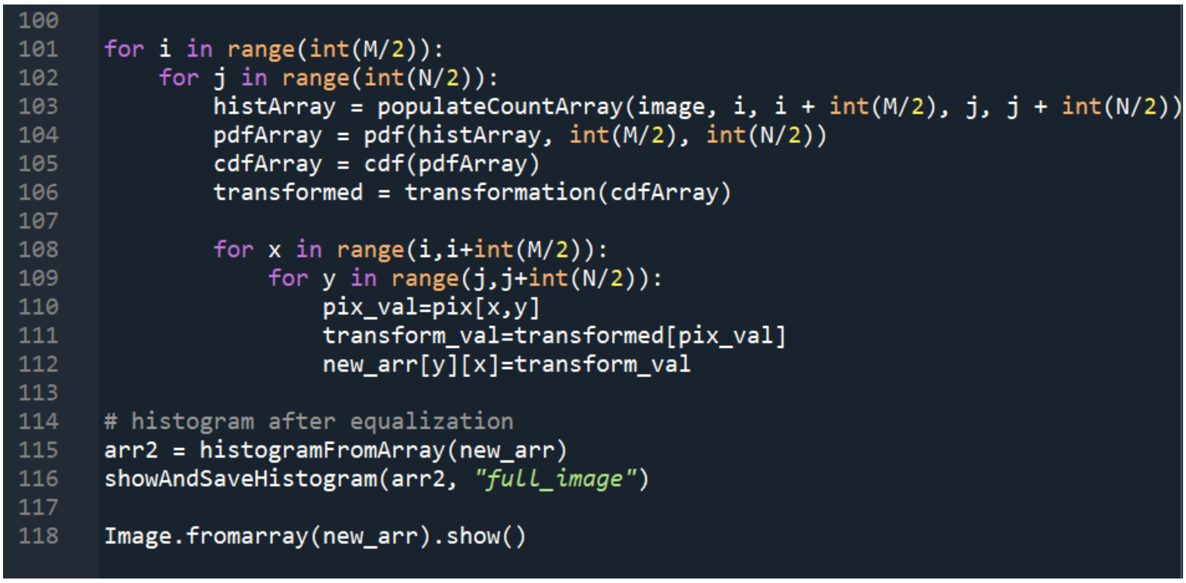
```
    return countArray

# probability distribution function
def pdf(array, width, height):
    pdfArray = [0] * 256

    # pdf = frequency / total no. of pixels
    for i in range(len(array)):
        # width * height is the total no. of pixels in that window
        pdfArray[i] = array[i] / (width * height)

    return pdfArray

# cummulative frequency distribution function
def cdf(array):
    cdfArray = [0] * 256

    # at every index, find the sum of current and all previous pdfs
    for i in range(len(array)):
        for j in range(i):
            cdfArray[i] += array[j]
    return cdfArray

# map cdf to intensity values
def transformation(array):
    transformed = [0] * 256
    for i in range(len(array)):
        transformed[i] = round(array[i] * (L - 1))
    return transformed
```



```

# plot the histogram
def showAndSaveHistogram(array, string):
    plt.title("Intensity Histogram")
    plt.xlabel("Intensity")
    plt.ylabel("Frequency")
    plt.bar([i for i in range(256)], array)
    plt.savefig(string + " histogram")
    plt.show()

# apply histogram equalization to the input image
def outputImage(array, img, startX,startY,endX,endY):
    pix = img.load()

    # iterate the pixels
    for x in range(startX,endX):
        for y in range(startY,endY):
            # change the intensity of the pixel to the transformed one
            pix[x, y] = array[pix[x, y]]

    return img
```

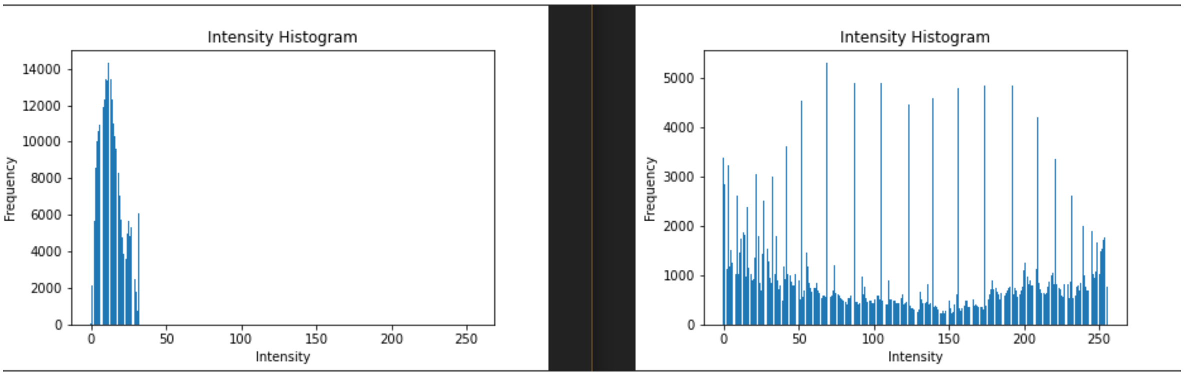


```python

def histogramFromArray(new_arr):
    new_img = Image.fromarray(new_arr)
    countArray = [0] * 256

    # load the pixels
    pix = new_img.load()
    (M,N) = new_img.size

    # iterate through the pixels
    for x in range(M):
        for y in range(N):
            # add 1 to the countArray whenever a particular intensity pixel is found
            countArray[pix[x, y]] += 1
    return countArray


# open image
image = Image.open("Lab07_img.png").convert('L')
M, N = image.size

# histogram before equalization
inputHistArray = populateCountArray(image, 0, M, 0, N)
showAndSaveHistogram(inputHistArray, "input")

pix = image.load()
new_arr = np.asarray(image).copy()
```

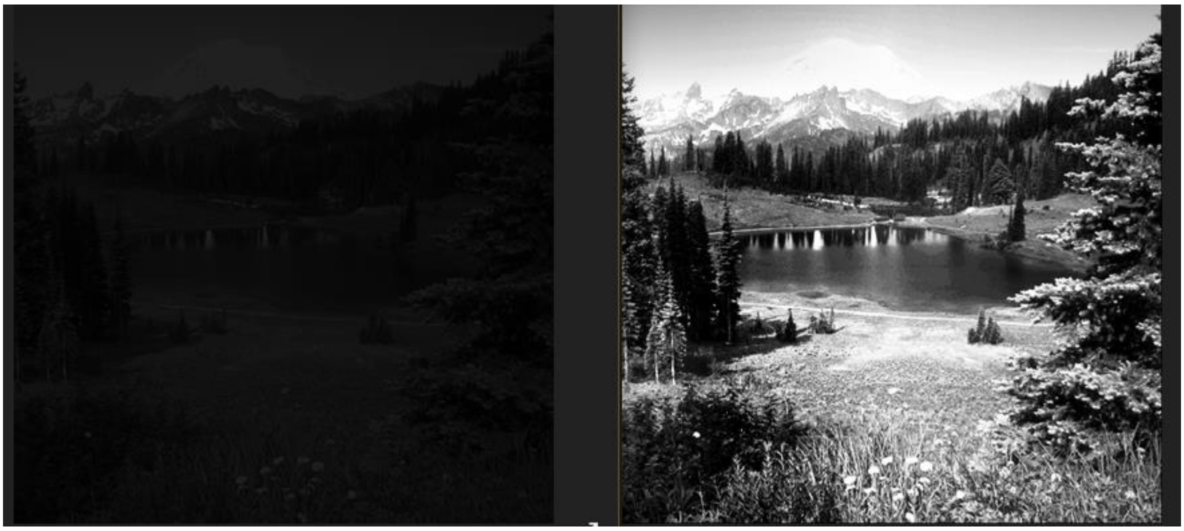


```python

for i in range(int(M/2)):
    for j in range(int(N/2)):
        histArray = populateCountArray(image, i, i + int(M/2), j, j + int(N/2))
        pdfArray = pdf(histArray, int(M/2), int(N/2))
        cdfArray = cdf(pdfArray)
        transformed = transformation(cdfArray)

        for x in range(i,i+int(M/2)):
            for y in range(j,j+int(N/2)):
                pix_val=pix[x,y]
                transform_val=transformed[pix_val]
                new_arr[y][x]=transform_val

# histogram after equalization
arr2 = histogramFromArray(new_arr)
showAndSaveHistogram(arr2, "full_image")

Image.fromarray(new_arr).show()
```

HISTOGRAMS



OUTPUT



Artifacts/ effects are observed when applying equalization at different levels in an image:

Using global histogram equalization, showed a result with good contrast.

In tiled histogram equalization, the edges/boundaries of the tiles are clearly visible making the image look like it’s divided into 4 tiles.

In sliding window histogram equalization, the image is clearer than with the global histogram equalization. Unlike tiled histogram equalization, windows cannot be seen using sliding window method.