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

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
```

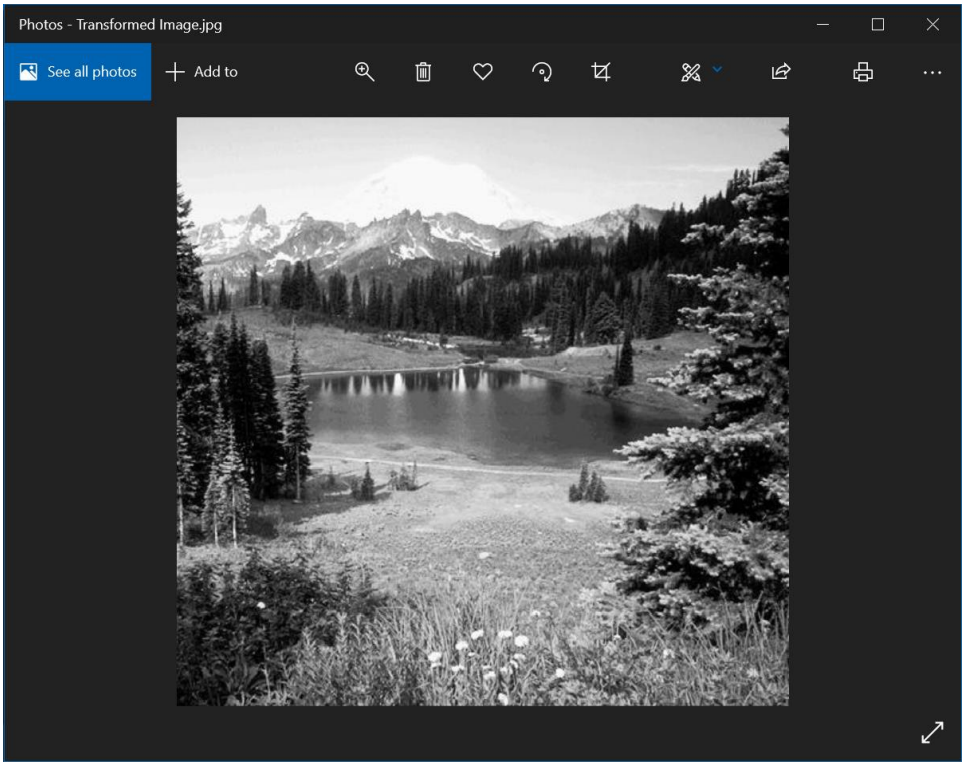
```
image = Image.open("lab07_img.png").convert('L')

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")
```

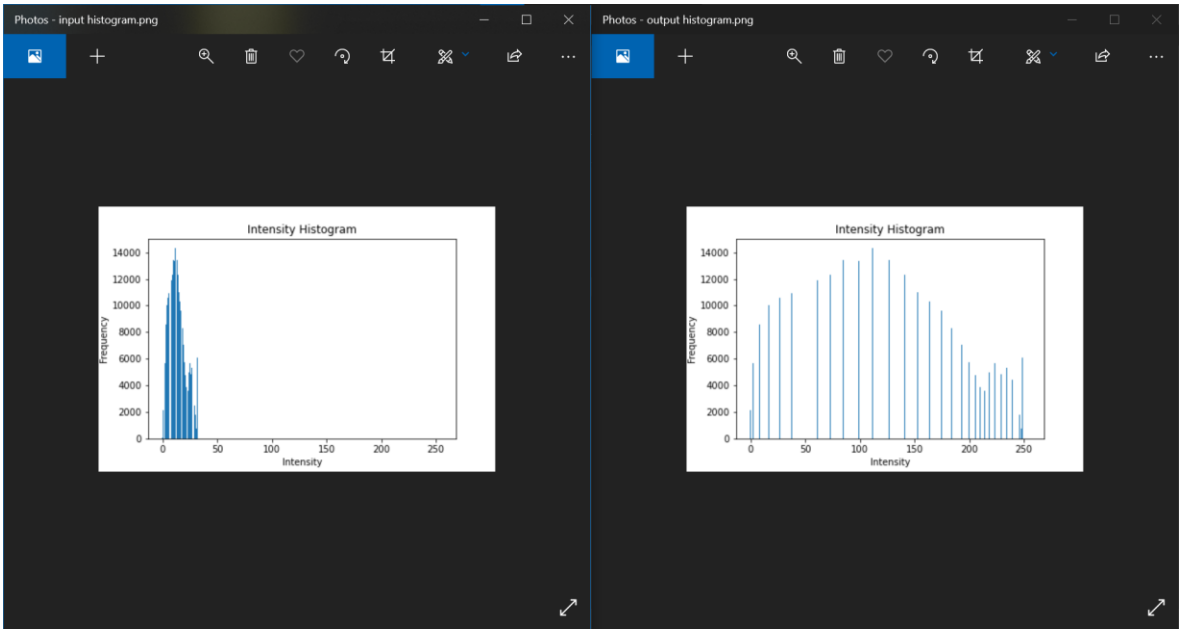
```python
from PIL import Image
from matplotlib import pyplot as plt

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

```

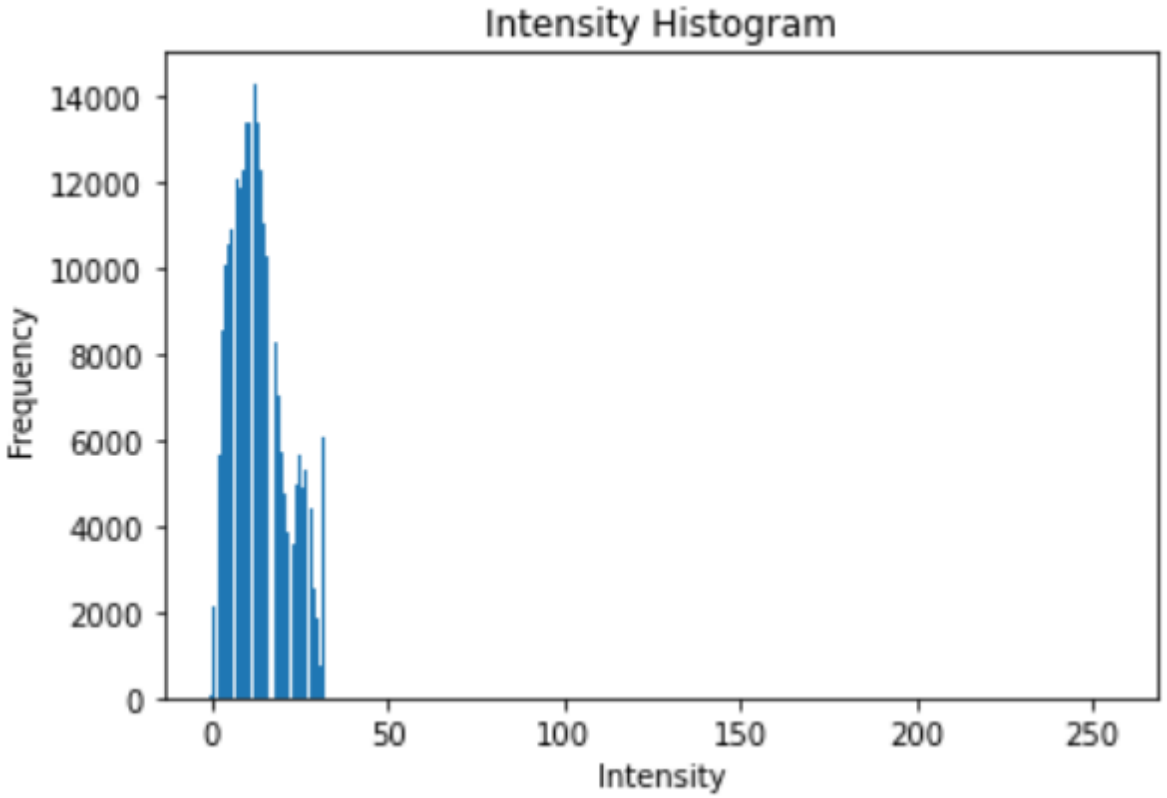
```python
# map cdf to intensity values
def transformation(array):
    transformed = [0] * 256
    for i in range(len(array)):
        transformed[i] = round(array[i] * (L - 1))
    return transformed

# 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
```

```
77
78
79  # open image
80  image = Image.open("lab07_img.png").convert('L')
81  histArray = populateCountArray(image)
82  pdfArray = pdf(histArray, image)
83  cdfArray = cdf(pdfArray)
84  transformed = transformation(cdfArray)
85  # save and show the histogram for the input image
86  showAndSaveHistogram(histArray, "input")
87  # get output image
88  output = outputImage(transformed, image)
89  histOutput = populateCountArray(image)
90  # save and show the histogram for the output image
91  showAndSaveHistogram(histOutput, "output")
92  # save output image
93  output.save("Transformed Image.jpg")
```

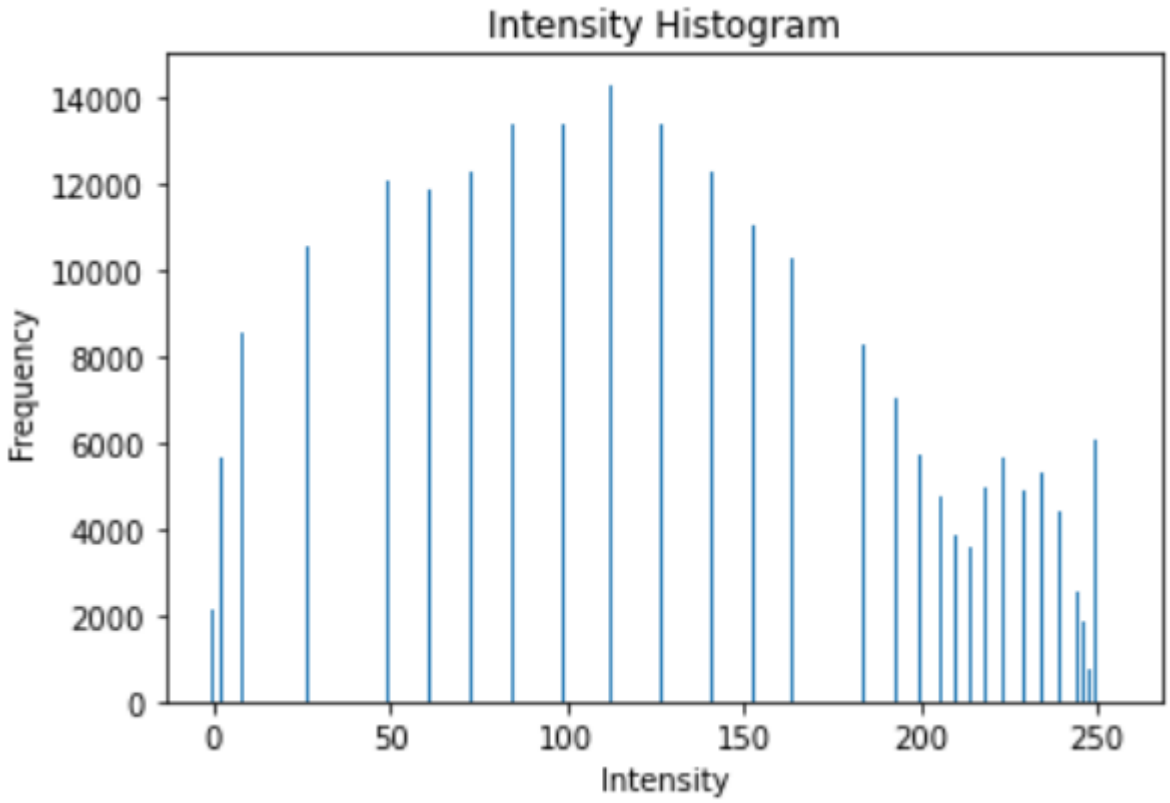




Before:



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

```

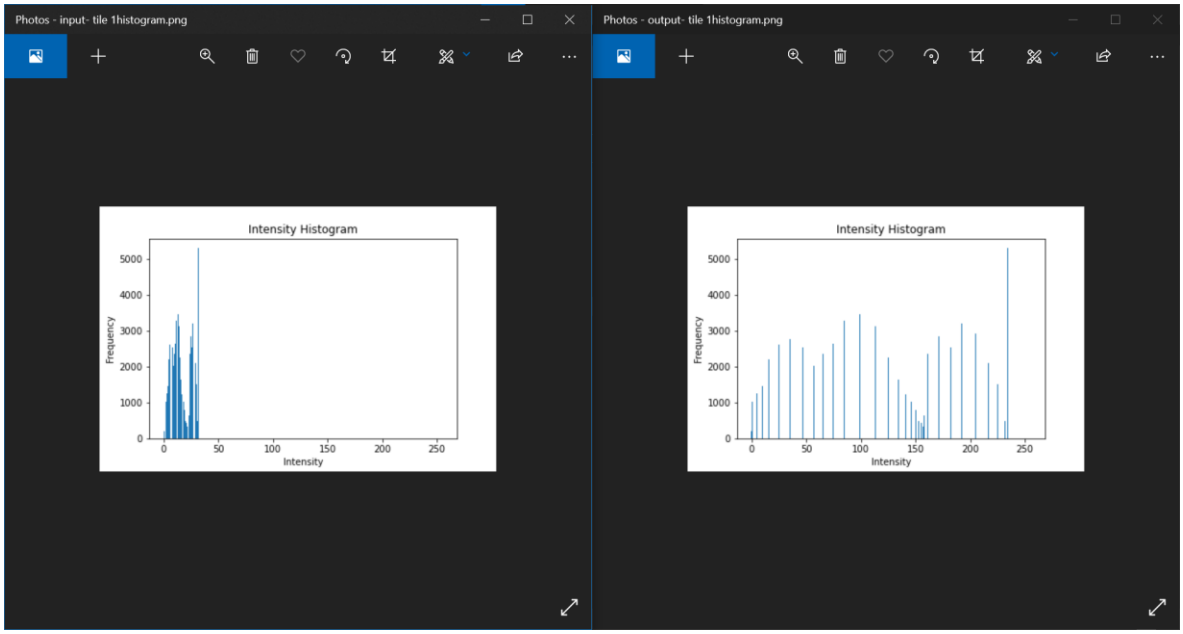
```python
# probability distribution function
def pdf(array, width, height):
    pdfArray = [0] * 256

    # 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
```

```
45
46  # map cdf to intensity values
47  def transformation(array):
48      transformed = [0]*256
49      for i in range(len(array)):
50          transformed[i] = round(array[i] * (L - 1))
51      return transformed
52
53  # plot the histogram
54  def showAndSaveHistogram(array, string, tileNo):
55      plt.title("Intensity Histogram")
56      plt.xlabel("Intensity")
57      plt.ylabel("Frequency")
58      plt.bar([i for i in range(256)], array)
59      plt.savefig(string + "- tile " + str(tileNo) + "histogram")
60      plt.show()
61
62  # apply histogram equalization to the input image
63  def outputImage(array, img, startX,startY,endX,endY):
64      pix = img.load()
65
66      # iterate the pixels
67      for x in range(startX,endX):
68          for y in range(startY,endY):
69              # change the intensity of the pixel to the transformed one
70              pix[x, y] = array[pix[x, y]]
71
72      return img
```

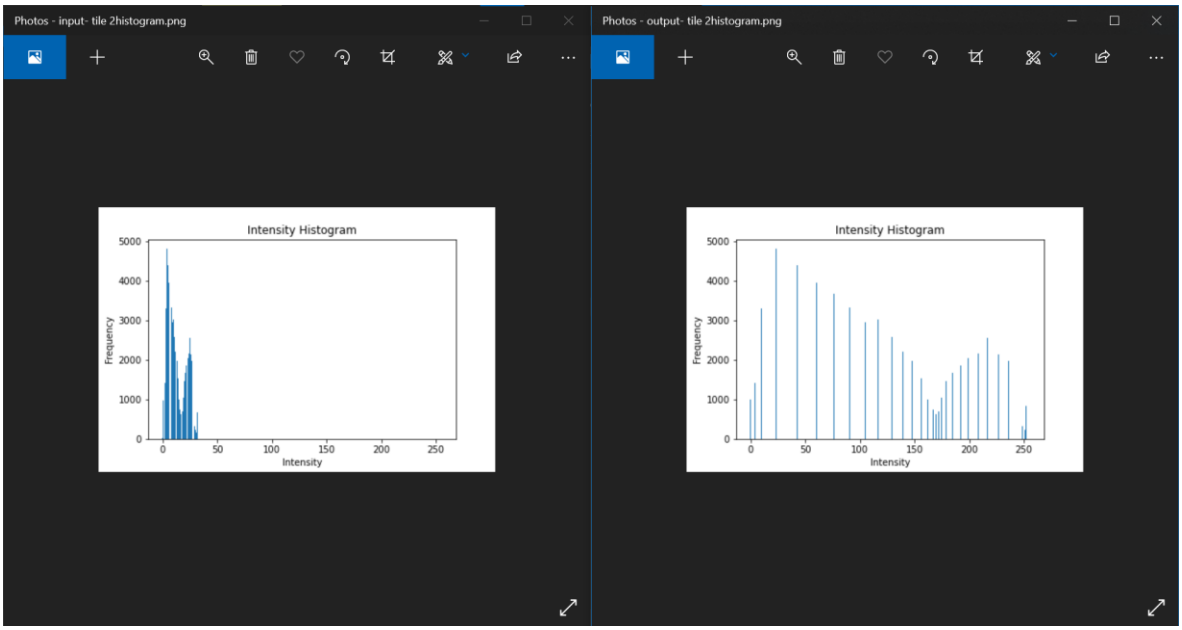
```
73
74
75  # open image
76  image = Image.open("Lab07_img.png").convert('L')
77  M, N = image.size
78
79  currentTile = 0
80
81  for i in range(2):
82      for j in range(2):
83          # histogram of input tile
84          tile = populateCountArray(image, j*int(M/2), i*int(N/2), (j+1)*int(M/2), (i+1)*int(N/2))
85          pdfArray = pdf(tile,M/2,N/2)
86          cdfArray = cdf(pdfArray)
87          transformed = transformation(cdfArray)
88          # for naming reasons
89          currentTile += 1
90          # showing and saving the input tile's histogram
91          showAndSaveHistogram(tile, "input", currentTile)
92          # map the transformed array to an output image of that tile
93          output = outputImage(transformed, image, j*int(M/2), i*int(N/2), (j+1)*int(M/2), (i+1)*int(N/2))
94           # histogram of output tile
95          outputHist = populateCountArray(output, j*int(M/2), i*int(N/2), (j+1)*int(M/2), (i+1)*int(N/2))
96          # showing and saving the output tile's histogram
97          showAndSaveHistogram(outputHist, "output", currentTile)
98
99  output.save("Task2.png")
```

HISTOGRAMS

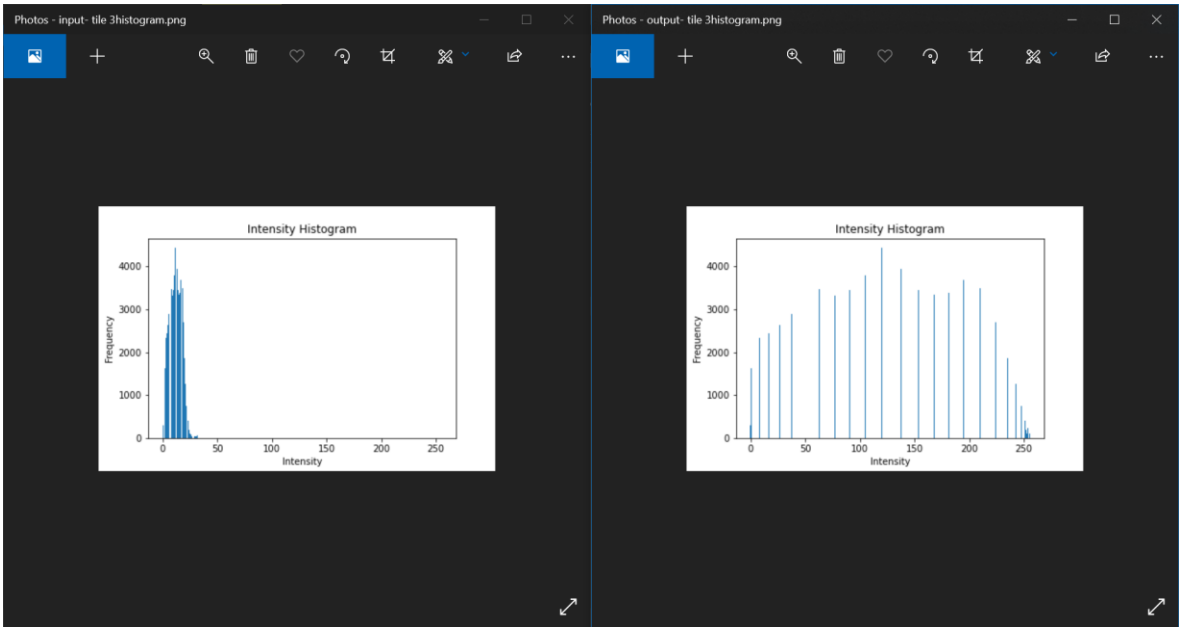
Tile 1


Photos - input- tile 1histogram.png
Intensity Histogram
Frequency
Intensity
Photos - output- tile 1histogram.png
Intensity Histogram
Frequency
Intensity

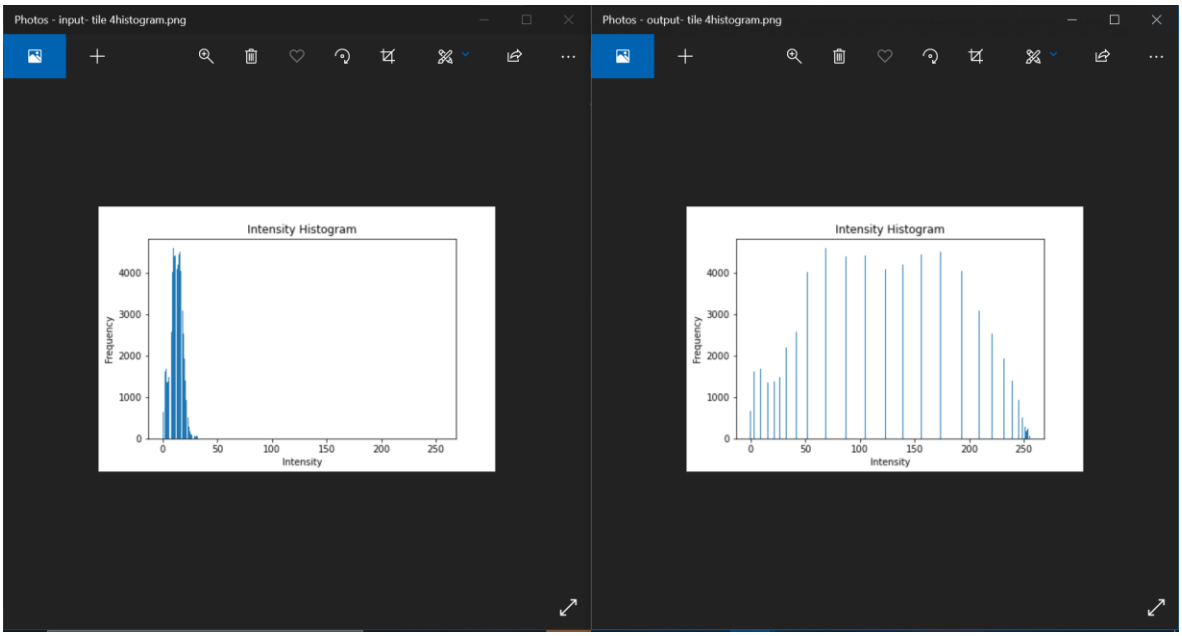
Tile 2


Photos - input- tile 2histogram.png
Intensity Histogram
Frequency
Intensity
Photos - output- tile 2histogram.png
Intensity Histogram
Frequency
Intensity

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
```

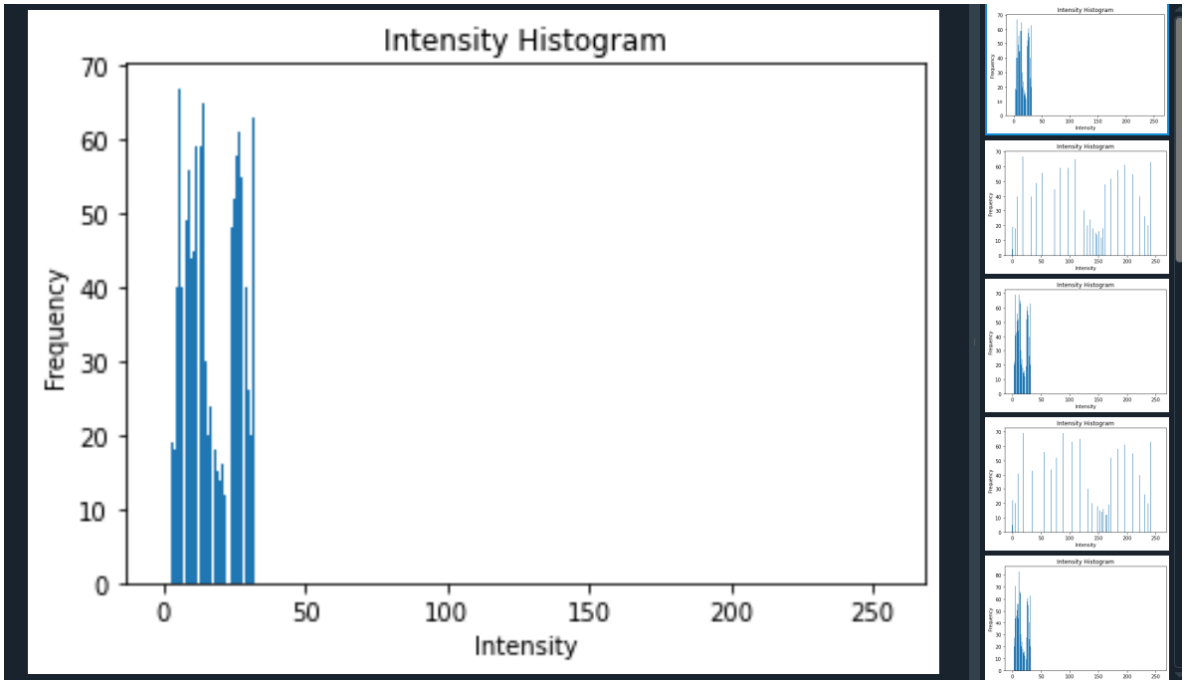
```python

# 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

```

```
53  # plot the histogram
54  def showAndSaveHistogram(array, string):
55      plt.title("Intensity Histogram")
56      plt.xlabel("Intensity")
57      plt.ylabel("Frequency")
58      plt.bar([i for i in range(256)], array)
59      plt.savefig(string + " histogram")
60      plt.show()
61
62  # apply histogram equalization to the input image
63  def outputImage(array, img, startX,startY,endX,endY):
64      pix = img.load()
65
66      # iterate the pixels
67      for x in range(startX,endX):
68          for y in range(startY,endY):
69              # change the intensity of the pixel to the transformed one
70              pix[x, y] = array[pix[x, y]]
71
72      return img
73
74  def histogramFromArray(new_arr):
75      new_img = Image.fromarray(new_arr)
76      countArray = [0] * 256
77
78      # load the pixels
79      pix = new_img.load()
80      (M,N) = new_img.size
81
82      # iterate through the pixels
83      for x in range(M):
84          for y in range(N):
85              # add 1 to the countArray whenever a particular intensity pixel is found
86              countArray[pix[x, y]] += 1
87      return countArray
```

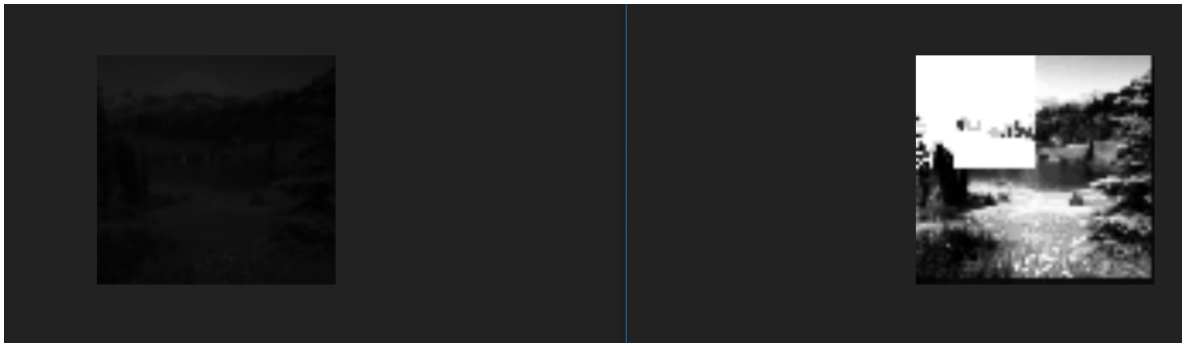
```
88
89
90  # open image
91  image = Image.open("lab07_img_resized.png").convert('L')
92  M, N = image.size
93
94
95  pix = image.load()
96  new_arr = np.asarray(image).copy()
97
```

```
98  for i in range(int(M/2)):
99      for j in range(int(N/2)):
100         histArray = populateCountArray(image, i, i + int(M/2), j, j + int(N/2))
101         pdfArray = pdf(histArray, int(M/2), int(N/2))
102         cdfArray = cdf(pdfArray)
103         transformed = transformation(cdfArray)
104         # save and show the histogram for the input image
105         showAndSaveHistogram(histArray, "input")
106
107         # get output image
108         output = outputImage(transformed, image, i, j, i + int(M/2), j + int(N/2))
109         histOutput = populateCountArray(output, i, i + int(M/2), j, j + int(N/2))
110         showAndSaveHistogram(histOutput, "output")
111
112         for x in range(i,i+int(M/2)):
113             for y in range(j,j+int(N/2)):
114                 pix_val=pix[x,y]
115                 transform_val=transformed[pix_val]
116                 new_arr[y][x]=transform_val
117
118         image = Image.open("lab07_img_resized.png").convert('L')
119
120 arr2 = histogramFromArray(new_arr)
121 showAndSaveHistogram(arr2, "full_image")
122
123 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()
```

```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
```

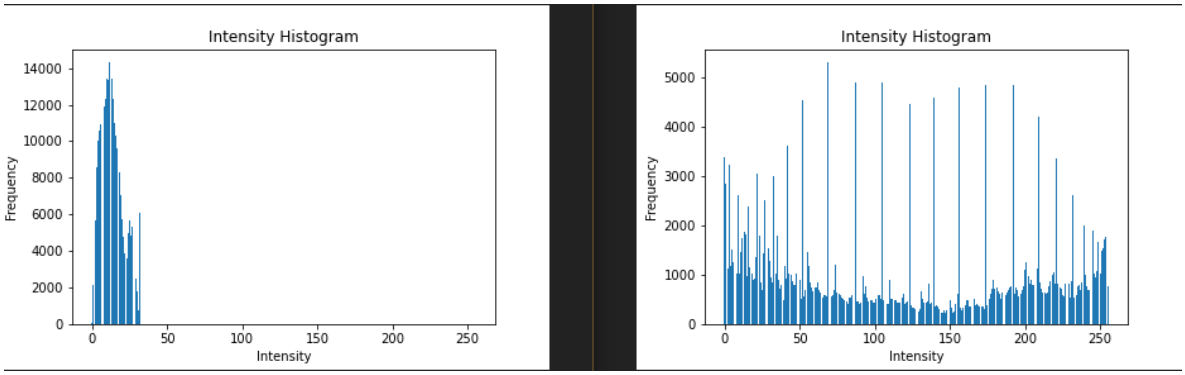
```
24
25  # probability distribution function
26  def pdf(array, width, height):
27      pdfArray = [0] * 256
28
29      # pdf = frequency / total no. of pixels
30      for i in range(len(array)):
31          # width * height is the total no. of pixels in that window
32          pdfArray[i] = array[i] / (width * height)
33
34      return pdfArray
35
36  # cummulative frequency distribution function
37  def cdf(array):
38      cdfArray = [0] * 256
39
40      # at every index, find the sum of current and all previous pdfs
41      for i in range(len(array)):
42          for j in range(i):
43              cdfArray[i] += array[j]
44      return cdfArray
45
46  # map cdf to intensity values
47  def transformation(array):
48      transformed = [0] * 256
49      for i in range(len(array)):
50          transformed[i] = round(array[i] * (L - 1))
51      return transformed
```

```
52
53  # plot the histogram
54  def showAndSaveHistogram(array, string):
55      plt.title("Intensity Histogram")
56      plt.xlabel("Intensity")
57      plt.ylabel("Frequency")
58      plt.bar([i for i in range(256)], array)
59      plt.savefig(string + " histogram")
60      plt.show()
61
62  # apply histogram equalization to the input image
63  def outputImage(array, img, startX,startY,endX,endY):
64      pix = img.load()
65
66      # iterate the pixels
67      for x in range(startX,endX):
68          for y in range(startY,endY):
69              # change the intensity of the pixel to the transformed one
70              pix[x, y] = array[pix[x, y]]
71
72      return img
```

```
73
74  def histogramFromArray(new_arr):
75      new_img = Image.fromarray(new_arr)
76      countArray = [0] * 256
77
78      # load the pixels
79      pix = new_img.load()
80      (M,N) = new_img.size
81
82      # iterate through the pixels
83      for x in range(M):
84          for y in range(N):
85              # add 1 to the countArray whenever a particular intensity pixel is found
86              countArray[pix[x, y]] += 1
87      return countArray
88
89
90  # open image
91  image = Image.open("lab07_img.png").convert('L')
92  M, N = image.size
93
94  # histogram before equalization
95  inputHistArray = populateCountArray(image, 0, M, 0, N)
96  showAndSaveHistogram(inputHistArray, "input")
97
98  pix = image.load()
99  new_arr = np.asarray(image).copy()
```

```
100
101  for i in range(int(M/2)):
102      for j in range(int(N/2)):
103          histArray = populateCountArray(image, i, i + int(M/2), j, j + int(N/2))
104          pdfArray = pdf(histArray, int(M/2), int(N/2))
105          cdfArray = cdf(pdfArray)
106          transformed = transformation(cdfArray)
107
108          for x in range(i,i+int(M/2)):
109              for y in range(j,j+int(N/2)):
110                  pix_val=pix[x,y]
111                  transform_val=transformed[pix_val]
112                  new_arr[y][x]=transform_val
113
114  # histogram after equalization
115  arr2 = histogramFromArray(new_arr)
116  showAndSaveHistogram(arr2, "full_image")
117
118  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.