
Emma Civello's Computer Vision Portfolio**Smith College CSC370 ~ Professor N. Howe**

Exercise 2: Histogram Equalization**Description**

A few weeks ago, a team brought up the idea of using a histogram's data to manipulate an image. Initially, I was confused by this idea, and so I did the histogram equalization exercise from Szeliski's textbook (3.6 on page 172 in my hardcover copy) to learn more. I found the following parts of the process especially challenging and interesting:

1. Converting from color to grayscale → I did this manually by taking the average of the three color channels, but there are *many* ways to do it (like taking a weighted average that favors green because people are more sensitive to green variations and using existing cv2 functions).
2. Performing the actual "equalization" → The idea was to find the total number of pixels and then loop through each individually. At each pixel, I found the number of pixels whose luminance value was less than the current one's value and divided this number by the total number of pixels; this process told me which percentile the current pixel fell into. The percentile translated directly into the pixel's new luminance value (for example, if the number of pixels whose value was less, when divided by the total, gave 0.25, its new luminance value was 0.25).

3. The idea of adding “punch” → The textbook suggested creating a threshold x so that the bottom $x\%$ of pixels were all floored to 0 and the top $x\%$ were all bumped up to 1.
4. The conversion back to color → I learned that you can use the ratio of each color channel at a pixel over the original luminance at that pixel to find the new value of the color channel at the new luminance of the pixel.

I liked this exercise a lot because of how dramatically different some of the outputs were from the original images. In the underexposed example below, I can barely see the people and the door / opening at the back until the image is rebalanced. I would be interested in looking at the color-handling more closely because I think that a lot of greens in the overexposed example image were lost in the final output – the output feels more flat to me than the input (I think that using a weighted average to convert between luminance and color might help this issue).

Finally, this exercise helped me discover more numpy functions – ones for calculating and displaying histograms and for finding the array index of the value that is closest to the intended value (useful for floating point numbers that are not represented precisely).





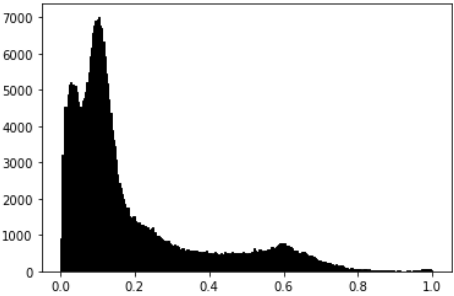
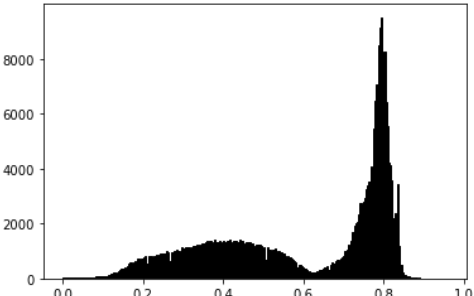
Image Results

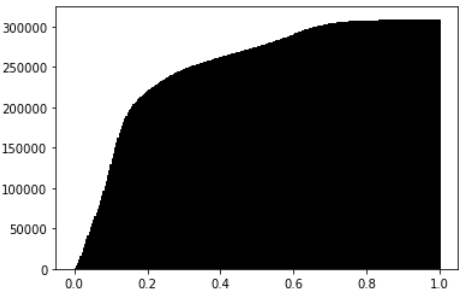
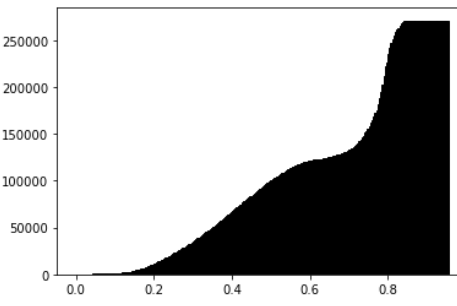
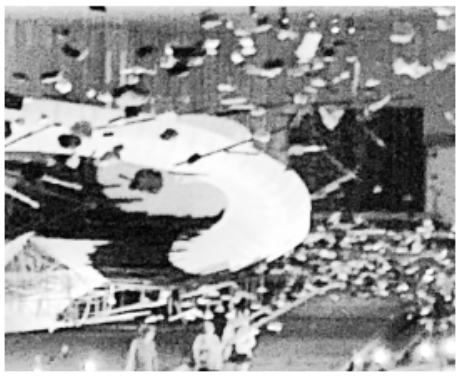

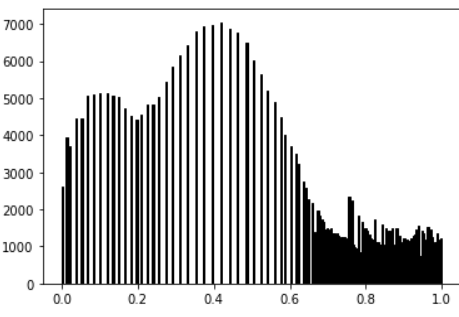
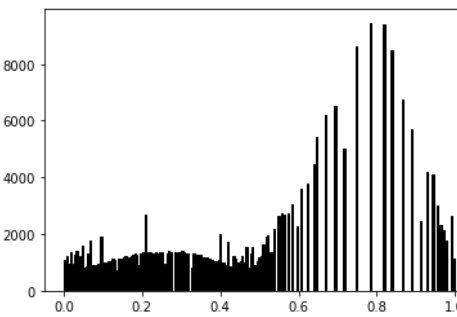
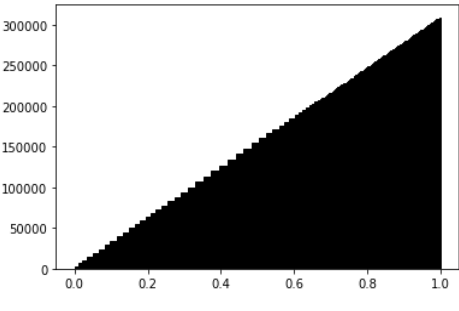
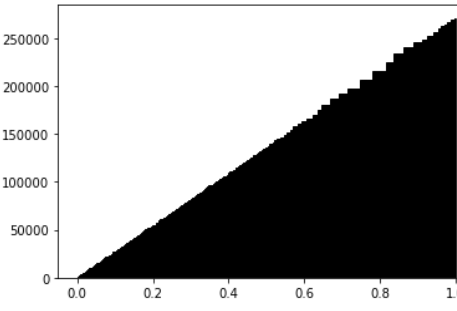
The original tree image came from here:

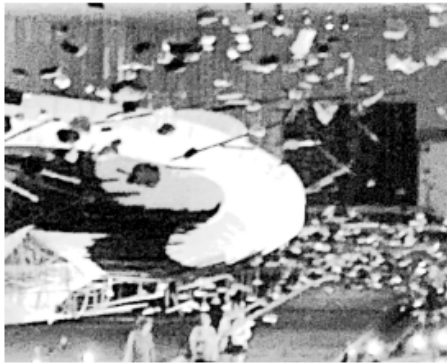

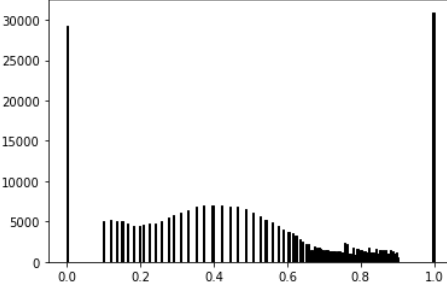
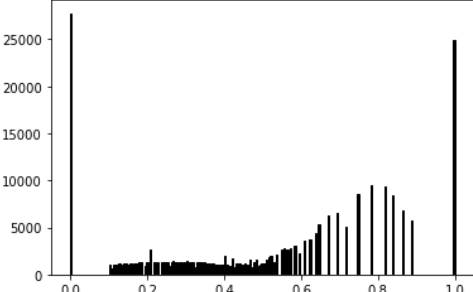
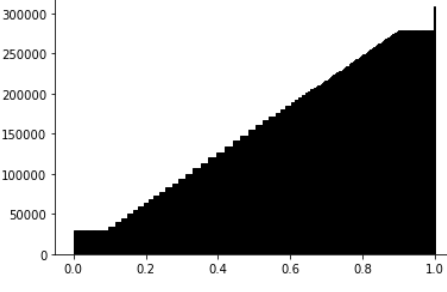
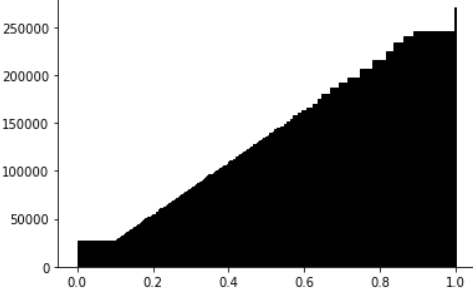


Siemens, Johann. 2014. “Green tree on grassland during daytime.” *Unsplash*.

<https://unsplash.com/photos/EPy0gBJzzZU>

(I took the underexposed image myself at MASS MoCA.)

Step Name	Step Output (underexposed example)	Step Output (overexposed example)
Initial image		
Converted to luminance / grayscale		
Grayscale histogram		

Grayscale cumulative distribution		
Equalized image		
Equalized histogram		
Equalized cumulative distribution		

<p>“Punched” image</p>		
<p>“Punched” histogram</p>		
<p>“Punched” cumulative distribution</p>		
<p>Re-converted to color</p>		

Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Sep 25 12:29:04 2022

@author: emmacivello
"""
import numpy
from matplotlib import image as matImg
from matplotlib import pyplot as plt
import math

# =====
# Basic-purpose Functions
# =====

#show image
def showImage(data):
    plt.imshow(data)
    plt.axis('off')
    plt.show()

def showImageBW(data):
    plt.imshow(data, cmap='Greys_r') #r = reversed (otherwise, black = white
    and vice versa)
    plt.axis('off')
    plt.show()

#import image
def getImgData(name):
    image = matImg.imread(name, format="PNG")
    data = numpy.asarray(image)

    #transform to range 0-1
    if(numpy.issubdtype(numpy.uint8, data[0][0][0])):
        data = data.astype(float) / 255

    return data

# =====
# Luminance conversion
# =====
#convert color image to luminance (aka gray scale) - approaches for this
explained here (I used method 2):
https://muthu.co/converting-color-images-to-grayscale-using-numpy-and-some-mathematics/
def colorToLuminance(data):
    newData = []
    width = len(data)
    height = len(data[0])

    blueChannel = data[:, :, 0]
    greenChannel = data[:, :, 1]
    redChannel = data[:, :, 2]

    for x in range(width):
```

```

        newData.append([])
        for y in range(height):
            blue = blueChannel[x][y]
            green = greenChannel[x][y]
            red = redChannel[x][y]
            newData[x].append((red+green+blue)/3)

    return newData

# =====
# Histograms
# =====
#compute the histogram
def computeHistogram(array):
    newArr = numpy.ndarray.flatten(numpy.array(array))
    return numpy.histogram(array, bins=256) #images go from range 0-255 to
range 0-1, so seems to make sense to make 256 bins

#show a histogram (also works for showing cumulative distribution)
def showHistogram(data):

#https://stackoverflow.com/questions/44003552/matplotlib-histogram-from-numpy-h
istogram-output
    frq, edges = data
    fig, ax = plt.subplots()
    ax.bar(edges[:-1], frq, width=numpy.diff(edges), color="black",
edgecolor="black", align="edge")
    plt.show()

# =====
# Cumulative distribution (here, I used built-in numpy function, but an
equation is also given on p95 of textbook, which is better to use)
# =====

def computeCumulativeDist(data):
    return numpy.cumsum(data)

#function from:
https://stackoverflow.com/questions/2566412/find-nearest-value-in-numpy-array
def find_nearest(array, value):
    array = numpy.asarray(array)
    idx = (numpy.abs(array - value)).argmin()
    return array[idx]

def findIndexOfBinValue(histogram, val):

#https://stackoverflow.com/questions/18079029/index-of-element-in-numpy-array
    i, = numpy.where(numpy.isclose(histogram, find_nearest(histogram, val)))
    return i

def computePartialCumulativeDist(histogram, intensity):
    binIndex = findIndexOfBinValue(histogram[1], intensity)[0]
    return numpy.sum(histogram[0][0:binIndex])

# =====

```

```

# Equalization
# =====
def applyEqualization(bwData, bwHist, bwCDist):
    newData = []
    width = len(bwData)
    height = len(bwData[0])

    for x in range(width):
        newData.append([])
        for y in range(height):
            partialCumulativeDist = computePartialCumulativeDist(bwHist,
bwData[x][y])
            newData[x].append(partialCumulativeDist / bwCDist[-1])

    return newData

# =====
# "Punch" / threshold values above 0.9 -> 1 and below 0.1 -> 0
# =====
def addPunch(arr):
    width = len(arr)
    height = len(arr[0])
    arr = numpy.asarray(arr)
    arr.flatten()
    arr[arr > 0.9] = 1
    arr[arr < 0.1] = 0
    numpy.reshape(arr, (width,height))
    return arr

# =====
# Go back to color after equalization, using original color ratios
# =====
def recolorImage(initImage, bwImage):
    newData = []
    width = len(bwImage)
    height = len(bwImage[0])

    for x in range(width):
        newData.append([])
        for y in range(height):
            red = initImage[x][y][2]
            green = initImage[x][y][1]
            blue = initImage[x][y][0]
            total = (red + green + blue)/3
            newTotal = bwImage[x][y]
            if(total != 0):
                newRed = (red/total) * newTotal
                newGreen = (green/total) * newTotal
                newBlue = (blue/total) * newTotal
            else:
                newRed = 0
                newGreen = 0
                newBlue = 0
            newData[x].append([newBlue,newGreen,newRed])

    return newData

```



```

def main():
    #starting point
    treeData = getImgData('tree.png')
    showImage(treeData)

    #convert image to luminance (and show)
    treeDataBW = colorToLuminance(getImgData('tree.png'))
    showImageBW(treeDataBW)

    #compute histogram (and show)
    bwHist = computeHistogram(treeDataBW)
    showHistogram(bwHist)

    #compute cumulative distribution (and show)
    bwCDist = computeCumulativeDist(bwHist[0])
    showHistogram([bwCDist,bwHist[1]])

    #compute new image and show it
    newData = applyEqualization(treeDataBW, bwHist, bwCDist)
    showImageBW(newData)

    #compute new histogram and show it
    bwHist2 = computeHistogram(newData)
    showHistogram(bwHist2)

    #compute new cumulative distribution (and show it)
    bwCDist2 = computeCumulativeDist(bwHist2[0])
    showHistogram([bwCDist2,bwHist2[1]])

    #add "punch" to image -> anything in bottom 5% rounded to 0 / black, in top
5% to 1 / white
    punchedData = addPunch(newData)
    showImageBW(punchedData)

    #compute new histogram and show it
    bwHist3 = computeHistogram(punchedData)
    showHistogram(bwHist3)

    #compute new cumulative distribution (and show it)
    bwCDist3 = computeCumulativeDist(bwHist3[0])
    showHistogram([bwCDist3,bwHist3[1]])

    #change image back into color
    coloredImg = recolorImage(treeData, punchedData)
    # print(len(coloredImg))
    # print(len(coloredImg[0]))
    # print(len(coloredImg[0][0]))
    # print(coloredImg[0][0])
    showImage(coloredImg)

main()

```