A Minor Project Report on

Image Processing/Image Enhancement

Submitted to Manipal University, Jaipur

Towards the partial fulfillment for the Award of the Degree of

BACHELORS OF TECHNOLOGY

In Computer Science Engineering
2017-2021

By
Arnav Sachdev
179301043

MANIPAL UNIVERSITY JAIPUR

INSPIRED BY LIFE

Under the guidance of
Dr.Ankit Shrivastava

Department of Computer Science Engineering,
School of Computing ad Information Technology
Manipal University Jaipur
Jaipur, Rajasthan

Table Of Contents

Introduction

Motivation

Literature Review

Problem Statement

Methodology

Work Done

- Overview
- Code
- Output/Results

Requirements

Bibilography

Introduction

The field of Digital Image Processing deals with the processing and analysis of digital images. It is a vast field encompassing various techniques such as thinning, binarization, enhancement and recognition.

It involves applying functions and filters on an image by adjusting the values of its constituent pixels in order to achieve a certain goal depending on the use case.

It includes the various techniques used to analyze images such as face detection, biometric recognition, etc which have important real world applications.

It allows for a broad range of algorithms to be applied to the input image to reduce undesired noise or distortion to make the image clearer.

The aim of Digital Image Enhancement is to adjust the pixels of a digital image so that the output image is better suited for further analysis. This can include increasing the contrast, removing noise, sharpening the features of an image, brightening elements of an image, etc.

Motivation

The motivation of this project is to create a flexible, reusable and extendable module which implements various Image Enhancement techniques which can be used in conjunction to other similar modules to form a suite of Image Processing tools.

Current research papers explain mathematical techniques and concepts to achieve the goal of Image Enhancement but lack an ***easy to use*** **practical implementation**. The goal of this project is to provide such an implementation.

The module is extendible via its use of classes to keep all related data and functions encapsulated, making it easier to add new functionality.

The module supports all popular images types like **JPEG, PNG, TIFF, BMP,** etc.

The module has a streamlined export feature which exports output images with verbose naming at each stage of the processing.

The module supports command line arguments and integration with the Linux shell.

The module allows exporting of histograms at each stage of processing so we can know how effective the enhancements have been in the form of a mathematical graph.

Literature Review

Contrast stretching is a point operation, i.e it is an operation applied to every pixel in the image. It uses a linear scaling function to allow the image to take a wider range of pixel intensity values, thus improving it's contrast.

The linear scaling function used is

$$P_{out} = (P_{in} - c)\left(\frac{b-a}{d-c}\right) + a$$

source: https://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm

The issue with an approach like this is that outlying pixel values can push the values of c and d approaching towards a and b which can give the result of unrepresentative scaling.

To solve this problem, we use a histogram to define a custom cutoff percentile and scale the image according to images in the corrected percentile range.

Although the literature recommends cutoff percentile values in the range of 5% to 95%, I have found that most images I have tested benefit from lower cutoff values of the order of **0**.***1% to 99.9%*** and ***1% to 99%***.

Generally, contrast stretching is performed on single band images, but I have applied the algorithm to multi band RGB images as well, stretching each band individually.

The **YCrCb** color space represents colors as **luminance, red-difference and blue-difference** just as **RGB** represents colors as Red, Green and Blue components.

I have utilized the **YCrCb** color space since literature suggests that contrast stretching the luminance band provides more color accurate results than individually stretching the red, green and blue bands of an image.

Problem Statement

To research and develop a module to Enhance a given digital image by improving it's

Sharpness
Contrast

using various Spatial Enhancement Techniques.

The existing method to improve image contrast is *histogram equalization.* The pros and cons of this method are:-

Pros	Cons
Increases overall global contrast of the image.	Produces unrealistic effects in photographs.
Works well with higher color depth images (16bit color depth and above).	This technique reduces the bit color depth and so is unsuitable for low color depth images (ex – 8bit color depth images). Further causes visible change in the image gradient of low color bit depth images.
Works well with false-color images such as X-Ray, thermal, satellite Images.	

Methodology

To achieve the goal of creating an enhanced image this project will use contrast stretching to improve the range the intensity values of the pixels of the image take.

The module is written to support 3 types of image formats, **GREYSCALE**, **RGB COLOR** and **YCrCb COLOR**.

First we find the minimum and maximum intensity values of the image, called the upper and lower limits.

Then we find the maximum and minimum intensity values actually present in the given digital image and use a scaling function to transform the pixels of the image which results in an image which better utilizes the intensity range possible for it's pixels.

The scaling function is given by

```
self.img[:,:] = (self.img[:,:] -
local_min)*((gmax-gmin)/(local_max-local_min)) + gmin
```

where gmin = 0
gmax = 255
local_min and local_max are calculated as per histogram (top n2%ile and bottom n1%ile)

```
    local_min = numpy.percentile(self.img[:,:],low)
    local_max = numpy.percentile(self.img[:,:],high)
```

Contrast stretching attempts to use a larger, better distributed range of values of pixel intensities in order to improve image perception.

Work Done

Overview

The project code consists of 3 classes, each with the following functions:-

colImg: class to handle RGB images

greyImg: class to handle greyscale images

lumImg: class to handle YCrCb images

All the classes implement at least the following methods:-

- **init:** constructor
- **info:** returns height, width and bit depth of image
- **conStretch:** performs contrast stretching
- **genHists:** generate histograms
- **write:** write to output

The main.py file handles command line I/O and arguments and calls all necessary functions as per the user's requirement.

Code

main.py

```
#!/usr/bin/python3

import cv2
import numpy
import PIL
from matplotlib import pyplot
import argparse
from lib import *
import sys

parser = argparse.ArgumentParser(description='contrast stretching
image enhancement module')
parser.add_argument('input',help='input file name')
#parser.add_argument('output',help='output file name')
parser.add_argument('--color',dest='color',action='store_true')
parser.add_argument('--grey',dest='grey',action='store_true')
parser.add_argument('--lum',dest='lum',action='store_true')
parser.add_argument('--split',dest='split',action='store_true')
parser.add_argument('--histogram',dest='hist',action='store_true')
parser.add_argument('n1',type=float)
parser.add_argument('n2',type=float)
args = parser.parse_args()

inputFile = args.input
outputFile = str(inputFile.split('.')[0]+'_output')
n1 = args.n1
n2 = args.n2
```

```
if __name__ == '__main__':

    if args.color:
      col = colImg(inputFile)
      col.info()
      if args.hist:

col.genHists(inputFile.split('.')[0]+'_rgb_before_histogram')
      col.conStretch(n1,n2)
      if args.hist:

col.genHists(inputFile.split('.')[0]+'_rgb_after_histogram')
      col.write(outputFile+'_color.png')
      sys.exit(0)

    elif args.grey:
      gry = greyImg(inputFile)
      gry.info()
      if args.hist:

gry.genHists(inputFile.split('.')[0]+'_grey_before_histogram')
      gry.conStretch(n1,n2)
      if args.hist:

gry.genHists(inputFile.split('.')[0]+'_grey_after_histogram')
      gry.write(outputFile+'_grey.png')
      sys.exit(0)

    elif args.lum:
      lum = lumImg(inputFile)
      lum.info()
      if args.hist:
           lum.genHists(inputFile.split('.')[0]+'_before_Y_stretch')
      lum.conStretchY(n1,n2)
      if args.hist:
           lum.genHists(inputFile.split('.')[0]+'_after_Y_stretch')
      lum.write(outputFile+'_ycrcb.png')
      sys.exit(0)

    elif args.split:
      print('feature not supported')
      sys.exit(0)
```

lib.py

```
import cv2
import numpy
import PIL
from matplotlib import pyplot


class colImg():

      def __init__(self,inputFilename):
      self.img = cv2.imread(inputFilename)

      def info(self):

print('shape:{0}\ndepth:{1}'.format(self.img.shape,self.img.dtype))

      def conStretch(self,low,high):
      for i in range(0,3):
            local_min = numpy.percentile(self.img[:,:,i],low)
            local_max = numpy.percentile(self.img[:,:,i],high)
            gmax = 255
            gmin = 0
            print('running for channel {0}\n\t{1}th
percentile:{2}\n\t{3}th
percentile:{4}'.format(i,low,local_min,high,local_max))
            self.img[:,:,i] = (self.img[:,:,i] -
local_min)*((gmax-gmin)/(local_max-local_min)) + gmin


      def genHists(self,histName):
      histlist = []
      for i in range(0,3):

histlist.append(cv2.calcHist([self.img],[i],None,[256],[0,256]))
            for i in range(0,len(histlist)):
                  pyplot.plot(histlist[i])
                  pyplot.savefig(str(histName+'_channel{}').format(i))
                  pyplot.clf()


      def write(self,outputFilename):
      cv2.imwrite(outputFilename,self.img)

      def con2gray(self):
```

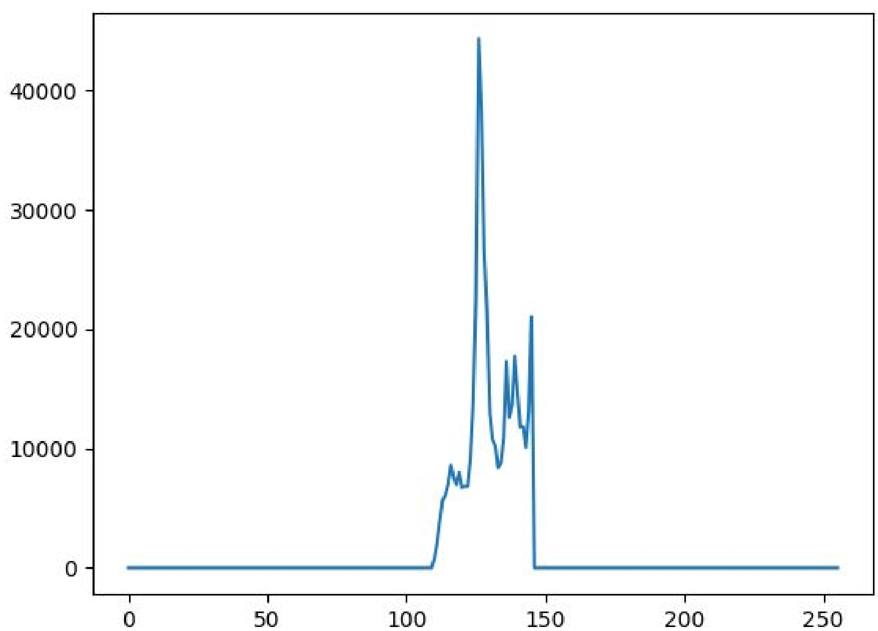
```
      pass


class greyImg():

      def __init__(self,inputFilename):
      temp = cv2.imread(inputFilename)
      self.img = cv2.cvtColor(temp,cv2.COLOR_BGR2GRAY)

      def info(self):

print('shape:{0}\ndepth:{1}'.format(self.img.shape,self.img.dtype))

      def conStretch(self,low,high):
      local_min = numpy.percentile(self.img[:,:],low)
      local_max = numpy.percentile(self.img[:,:],high)
      gmin=0
      gmax=255
      print('running for channel 0\n\t{0}th percentile:{1}\n\t{2}th
percentile:{3}'.format(low,local_min,high,local_max))
      self.img[:,:] = (self.img[:,:] -
local_min)*((gmax-gmin)/(local_max-local_min)) + gmin

      def genHists(self,histName):
      histogram = cv2.calcHist([self.img],[0],None,[256],[0,256])
      pyplot.plot(histogram)
      pyplot.savefig(histName)
      pyplot.clf()

      def write(self,outputFilename):
      cv2.imwrite(outputFilename,self.img)

      def colorize(self):
      self.img = cv2.cvtColor(self.img,cv2.COLOR_GRAY2BGR)


class lumImg():

      def __init__(self,inputFilename):
      temp = cv2.imread(inputFilename)
      self.img = cv2.cvtColor(temp,cv2.COLOR_BGR2YCR_CB)

      def info(self):

print('shape:{0}\ndepth:{1}'.format(self.img.shape,self.img.dtype))
```

```
	def write(self,outputFilename):
	self.img = cv2.cvtColor(self.img,cv2.COLOR_YCrCb2BGR)
	cv2.imwrite(outputFilename,self.img)

	def genHists(self,histName):
	histogram = cv2.calcHist([self.img],[0],None,[256],[0,256])
	pyplot.plot(histogram)
	pyplot.savefig(str(histName))
	pyplot.clf()


	def conStretchY(self,low,high):
	local_min = numpy.percentile(self.img[:,:,0],low)
	local_max = numpy.percentile(self.img[:,:,0],high)
	gmin=0
	gmax=255
	print('running for channel Y\n\t{0}th percentile:{1}\n\t{2}th
percentile:{3}'.format(low,local_min,high,local_max))
	#self.img[:,:,0] = (self.img[:,:,0] + 25)%256
	self.img[:,:,0] = (self.img[:,:,0] -
local_min)*((gmax-gmin)/(local_max-local_min)) + gmin
```

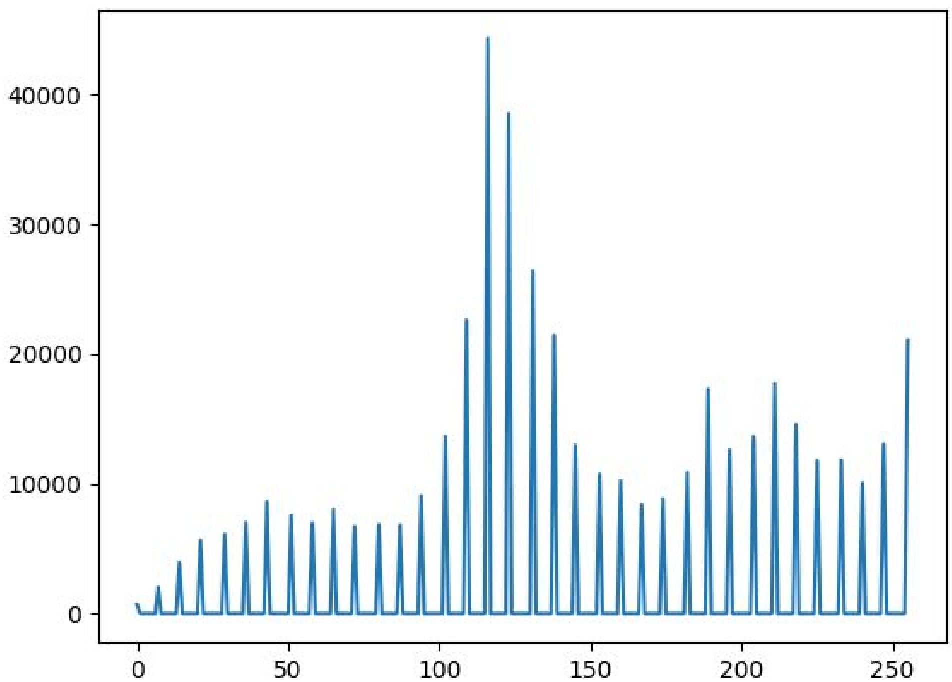
Examples

Example 1: Greyscale Image

before

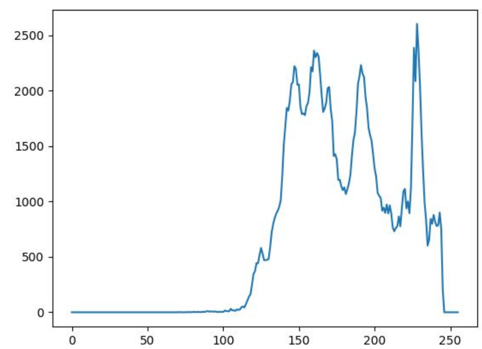


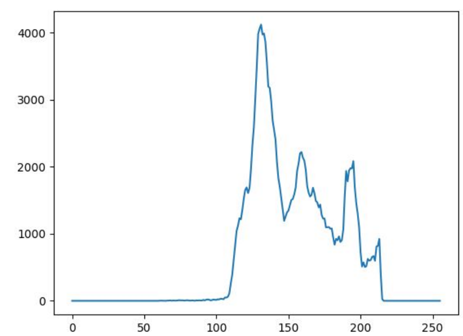
after

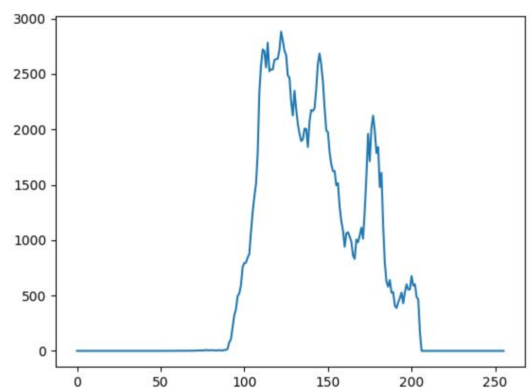


Example 2: Colored Image using RGB Contrast Stretching

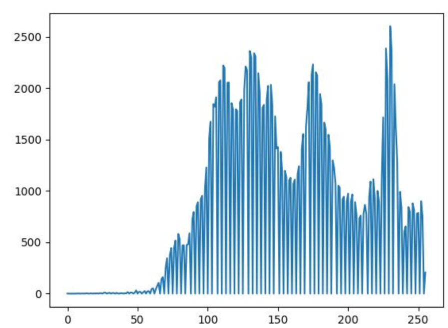
before



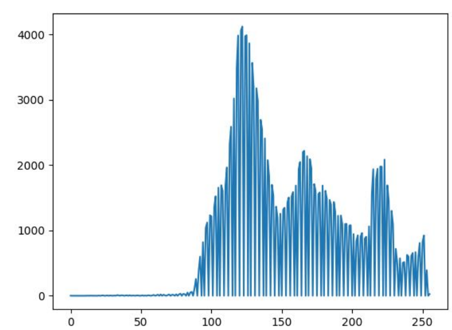




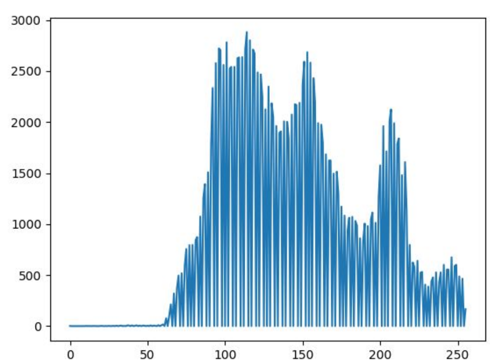
after



2500
2000
1500
1000
500
0
0
50
100
150
200
250



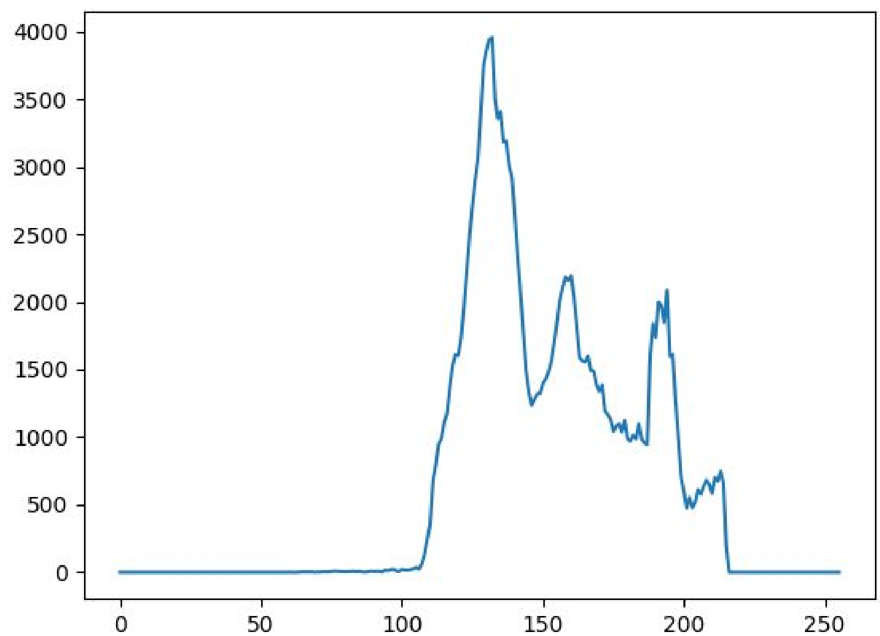
4000
3000
2000
1000
0
0
50
100
150
200
250



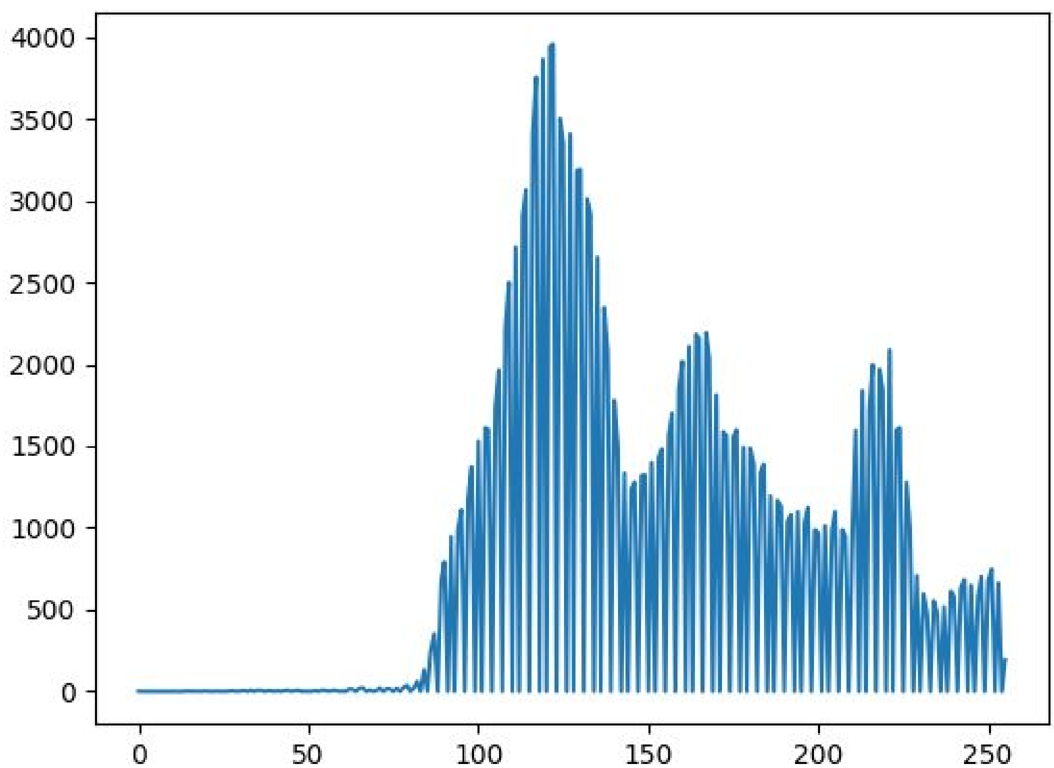
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250

Example 3: Colored Image using YCrCb Luminance Stretching

before

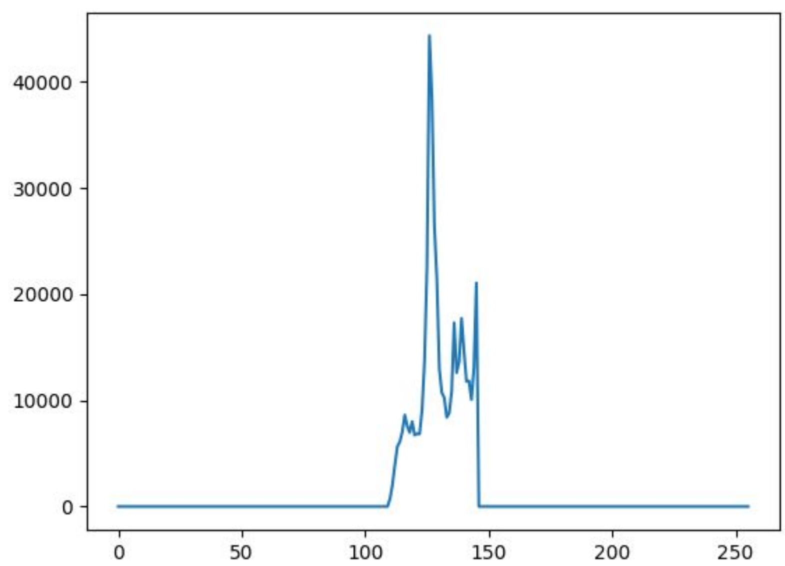


after

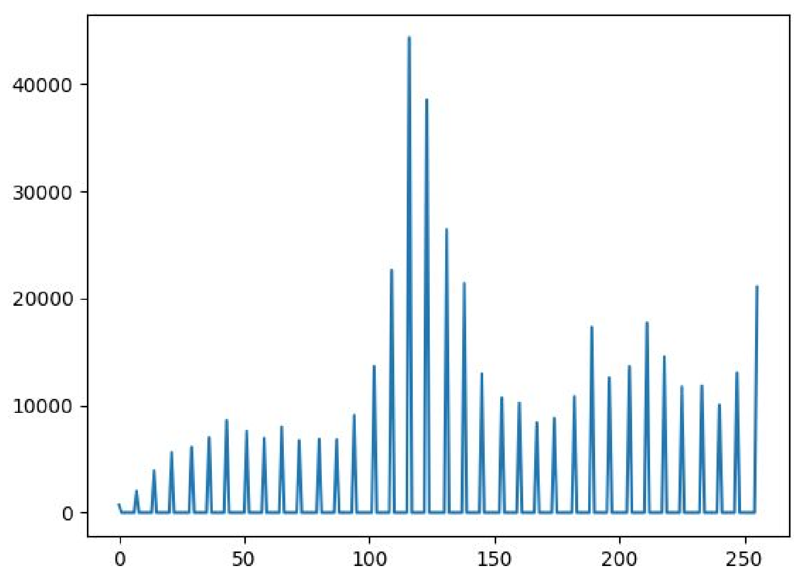


Example 4; Greyscale Image using Single Band Stretching

before



after



Requirements

This project will be implemented in Python3 and make use of the following supporting software libraries provided for the language

PythonImagingLibrary/PIL
OpenCV-python

The development machine is a modern Linux based system (Linux predator 5.4.6-2-MANJARO #1 SMP PREEMPT Tue Dec 24 15:55:20 UTC 2019 x86_64 GNU/Linux)

Bibliography/References

https://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm

Kim, Yeong-Taeg. "Contrast enhancement using brightness preserving bi-histogram equalization." IEEE transactions on Consumer Electronics 43.1 (1997): 1-8.

www.cse.iitd.ernet.in/~pkalra/col783/week-2.pdf