Homework Three:

Problem 01:

Original Image:

A picture containing tree, sky, outdoor

Description automatically generated

Code:

#%%

#importing cv2

import cv2

# Using cv2.imread() method

img1 = cv2.imread('3\_1.bmp')

# Displaying the image using cv2.imshow()

cv2.imshow('3\_1.bmp', img1)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

Blue\_Band = img1[:, :, 0]

Green\_Band = img1[:, :, 1]

Red\_Band = img1[:, :, 2]

# %%

BB\_rows = Blue\_Band.shape[0]

GB\_rows = Green\_Band.shape[0]

RB\_rows = Red\_Band.shape[0]

BB\_coloumns = Blue\_Band.shape[1]

GB\_coloumns = Green\_Band.shape[1]

RB\_coloumns = Red\_Band.shape[1]

# %%

#Algorithm

new\_BB= np.zeros((int(BB\_rows), int(BB\_coloumns)))

#gamma = 3

gama\_correction = 1.3

for i in range(0, BB\_rows):

    for j in range(0, BB\_coloumns):

        new\_BB[i, j] = 255\*(Blue\_Band[i, j]/255)\*\*gama\_correction

Fin\_BB = np.uint8(new\_BB)

#%%

new\_GB= np.zeros((int(GB\_rows), int(GB\_coloumns)))

#gamma = 3

gama\_correction = 0.7

for i in range(0, GB\_rows):

    for j in range(0, GB\_coloumns):

        new\_GB[i, j] = 255\*(Green\_Band[i, j]/255)\*\*gama\_correction

Fin\_GB = np.uint8(new\_GB)

#%%

new\_RB= np.zeros((int(RB\_rows), int(RB\_coloumns)))

#gamma = 3

gama\_correction = 0.9

for i in range(0, RB\_rows):

    for j in range(0, RB\_coloumns):

        new\_RB[i, j] = 255\*(Red\_Band[i, j]/255)\*\*gama\_correction

Fin\_RB = np.uint8(new\_RB)

#%%

image\_final = cv2.merge((Fin\_BB, Fin\_GB, Fin\_RB))

cv2.imshow('Natural Image', image\_final)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

Natural Image:

A close up of a flower garden

Description automatically generated

Problem 2:

Original Image 01:

A picture containing person, baseball, player, outdoor

Description automatically generated

Original Image 01 Histogram:

A screenshot of a cell phone

Description automatically generated

Original Image 01 CDF:

A screenshot of a cell phone

Description automatically generated

Original Image 02:

A picture containing floor, building

Description automatically generated

Histogram original image 02:

A screenshot of a social media post

Description automatically generated

CDF original image 02:

A screenshot of a cell phone

Description automatically generated

Linear Stretching:

Original Image 01

A picture containing person, baseball, player, outdoor

Description automatically generated

Code:

#%%

#Problem 2

#Linear Streatching

#importing cv2

import cv2

# Using cv2.imread() method

img1 = cv2.imread('3\_2.jpg')

# Displaying the image using cv2.imshow()

cv2.imshow('3\_2.jpg', img1)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

hsv\_image1 = cv2.cvtColor(img1, cv2.COLOR\_BGR2HSV)

# Displaying the image using cv2.imshow()

cv2.imshow('HSV Image', hsv\_image1)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

import numpy as np

value1 = hsv\_image1[:, :, 2]

minimum = np.amin(value1)

maximum = np.amax(value1)

#the minimum range is 1 and maximum range is 157

r = value1.shape[0]

c = value1.shape[1]

new\_value1= np.zeros((int(r), int(c)))

# %%

#Choosing Appropriate value for x1, x2, y1, y2

r1 = 60

r2 = 120

s1 = 40

s2 = 70

slope1 = s1/r1

slope2 = (s2-s1)/(r2-r1)

slope3= (255-s2)/(255-r2)

"""

intersect1 = y1-slope2\*x1

intersect2 = y2-slope3\*x2

"""

for i in range(0, r):

    for j in range (0, c):

        if 0<=value1[i, j]<=r1:

            new\_value1[i, j]=(slope1\*value1[i, j])

        elif r1<=value1[i, j]<=r2:

            new\_value1[i, j]=(slope2\*(value1[i, j]-r1))+s1

        elif r2<value1[i, j]<255:

            new\_value1[i, j]=(slope2\*(value1[i, j]-r2))+s2

v1 = cv2.normalize(src=new\_value1, dst=None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

#%%

h1= hsv\_image1[:, :, 0]

s1= hsv\_image1[:, :, 1]

new\_hsv= cv2.merge((h1,s1,v1))

final\_rgb = cv2.cvtColor(new\_hsv, cv2.COLOR\_HSV2BGR)

# Displaying the image using cv2.imshow()

cv2.imshow('Linearly\_Streatched\_Image1', final\_rgb)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

#Collecting values from 2D array

image\_first\_band = final\_rgb[:,:,2]

rows = image\_first\_band.shape[0]

coloumns= image\_first\_band.shape[1]

values = []

for i in range(0, rows):

    for j in range(0, coloumns):

        values.append(image\_first\_band[i, j])

frequencies = {x:values.count(x) for x in values}

import collections

od = collections.OrderedDict(sorted(frequencies.items()))

#%%

#Histogram

import matplotlib.pyplot as plt

ax = plt.subplot(111)

w = 0.3

ax.bar(list(od.keys()), list(od.values()) , width=w, color='b', align='center')

ax.autoscale(tight=True)

plt.title("Histrogram of linearly\_streatched\_image01")

plt.ylim([0, 1200])

plt.xlabel("Pixel Intensity")

plt.ylabel("Pixel Frequency")

plt.show()

#%%

#CDF

import numpy as np

probability = []

for item in list(od.values()):

    probability.append(item/sum(list(od.values())))

cp = np.cumsum(probability).tolist()

od\_list = list(od.keys())

amin, amax = min(od\_list), max(od\_list)

for i, val in enumerate(od\_list):

    od\_list[i] = (val-amin) / (amax-amin)

plt.xlabel("Pixel Intensity")

plt.ylabel("Cumulative Probability")

plt.title("Cumulative Distribution Function of linearly\_streatched\_image01")

plt.plot(od\_list, cp, c='blue')

plt.show()

Resulting Image:

A group of baseball players that are standing in front of a crowd

Description automatically generated

Linearly Stretched Image 01 histogram:

A screenshot of a cell phone

Description automatically generated

Linearly Stretched Image 01 CDF:

A screenshot of a cell phone

Description automatically generated

Original Image 02:

A picture containing floor, building

Description automatically generated

Linearly stretched image 02:

A person standing in a room

Description automatically generated

Linearly Stretched Image 02 histogram:

A screenshot of a social media post

Description automatically generated

Linearly Stretched Image 01 CDF:

A screenshot of a cell phone

Description automatically generated

Histogram Equalization:

#%%

#importing cv2

import cv2

# Using cv2.imread() method

img1 = cv2.imread('3\_3.jpg')

# Displaying the image using cv2.imshow()

cv2.imshow('3\_3.jpg', img1)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

hsv\_image1 = cv2.cvtColor(img1, cv2.COLOR\_BGR2HSV)

# Displaying the image using cv2.imshow()

cv2.imshow('HSV Image', hsv\_image1)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

value\_band = hsv\_image1[:, :, 2]

rows = value\_band .shape[0]

coloumns= value\_band.shape[1]

values = []

for i in range(0, rows):

    for j in range(0, coloumns):

        values.append(value\_band [i, j])

frequencies = {x:values.count(x) for x in values}

import collections

od = collections.OrderedDict(sorted(frequencies.items()))

import numpy as np

probability = []

for item in list(od.values()):

    probability.append(item/sum(list(od.values())))

cp = np.cumsum(probability).tolist()

od\_list = list(od.keys())

res = {od\_list[i]: cp[i] for i in range(len(od\_list))}

# %%

import math

r = value\_band .shape[0]

c = value\_band.shape[1]

new\_band= np.zeros((int(r), int(c)))

for i in range (0, r):

    for j in range (0, c):

        new\_band[i, j]= math.floor(res[(value\_band [i, j])]\*255)

v1 = cv2.normalize(src=new\_band, dst=None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

#%%

h1= hsv\_image1[:, :, 0]

s1= hsv\_image1[:, :, 1]

new\_hsv= cv2.merge((h1,s1,v1))

final\_rgb = cv2.cvtColor(new\_hsv, cv2.COLOR\_HSV2BGR)

# Displaying the image using cv2.imshow()

cv2.imshow('Histogram\_Equalization\_Image1', final\_rgb)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

#Collecting values from 2D array

image\_first\_band = final\_rgb[:,:,2]

rows = image\_first\_band.shape[0]

coloumns= image\_first\_band.shape[1]

values = []

for i in range(0, rows):

    for j in range(0, coloumns):

        values.append(image\_first\_band[i, j])

frequencies = {x:values.count(x) for x in values}

import collections

od = collections.OrderedDict(sorted(frequencies.items()))

#%%

#Histogram

import matplotlib.pyplot as plt

ax = plt.subplot(111)

w = 0.3

ax.bar(list(od.keys()), list(od.values()) , width=w, color='b', align='center')

ax.autoscale(tight=True)

plt.title("Histrogram of\_histogram\_equalized\_\_image02")

plt.xlabel("Pixel Intensity")

plt.ylabel("Pixel Frequency")

plt.show()

#%%

#CDF

import numpy as np

probability = []

for item in list(od.values()):

    probability.append(item/sum(list(od.values())))

cp = np.cumsum(probability).tolist()

od\_list = list(od.keys())

amin, amax = min(od\_list), max(od\_list)

for i, val in enumerate(od\_list):

    od\_list[i] = (val-amin) / (amax-amin)

plt.xlabel("Pixel Intensity")

plt.ylabel("Cumulative Probability")

plt.title("Cumulative Distribution Function of histogram\_equalized\_\_image02")

plt.plot(od\_list, cp, c='blue')

plt.show()

# %%

Resulting Image 01:

A person standing in front of a crowd

Description automatically generated

Histogram of histogram equalized image01:

A screenshot of a cell phone

Description automatically generated

CDF of histogram equalized image01:

A screenshot of a cell phone

Description automatically generated

Resulting Image 02:



Histogram of histogram equalized image02:

A screenshot of a cell phone

Description automatically generated

CDFof histogram equalized image02:

A screenshot of a cell phone

Description automatically generated

Histogram Specification:

#%%

#importing cv2

import cv2

# Using cv2.imread() method

img1 = cv2.imread('3\_2.jpg')

# Displaying the image using cv2.imshow()

cv2.imshow('3\_2.jpg', img1)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

hsv\_image1 = cv2.cvtColor(img1, cv2.COLOR\_BGR2HSV)

# Displaying the image using cv2.imshow()

cv2.imshow('HSV Image', hsv\_image1)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

# %%

#Creating normally distributed transformation function

import numpy as np

original\_value\_band = hsv\_image1[:, :, 2]

target\_value\_band= np.random.binomial(n=255, p=0.5, size=(228, 300))

target\_value\_band1 = np.uint8(target\_value\_band)

#%%

#Algorithm

import collections

def takeClosest(num,collection):

   return min(collection,key=lambda x:abs(x-num))

intensity1 = []

intensity2 = []

probability1 =[]

probability2 = []

values1 = []

values2 = []

intensity= []

final\_list =[]

rows1 = original\_value\_band .shape[0]

coloumns1= original\_value\_band.shape[1]

for i in range(0, rows1):

    for j in range(0, coloumns1):

        values1.append(original\_value\_band[i, j])

frequencies1 = {x1:values1.count(x1) for x1 in values1}

rows2 = target\_value\_band1.shape[0]

coloumns2= target\_value\_band1.shape[1]

for i in range(0, rows2):

    for j in range(0, coloumns2):

        values2.append(target\_value\_band1[i, j])

frequencies2 = {x2:values2.count(x2) for x2 in values2}

for item in range(0, 256):

    if item in frequencies1:

        intensity1.append(frequencies1[item])

    else:

        intensity1.append(0)

for item in range(0, 256):

    intensity.append(item)

    if item in frequencies2:

        intensity2.append(frequencies2[item])

    else:

        intensity2.append(0)

for item in intensity1:

    probability1.append(item/sum(intensity1))

for item in intensity2:

    probability2.append(item/sum(intensity2))

cp1 = np.cumsum(probability1).tolist()

cp2 = np.cumsum(probability2).tolist()

res = {cp2[i]: intensity[i] for i in range(len(cp2))}

for item in cp1:

    final\_list.append(res[takeClosest(item,cp2)])

#intensity is mapped to final list

# %%

resf= {intensity[i]: final\_list[i] for i in range(len(intensity))}

new\_original\_band= np.zeros((int(rows1), int(coloumns1)))

for i in range(0, rows1):

    for j in range(0,coloumns1 ):

        new\_original\_band[i, j] = resf[(original\_value\_band[i, j])]

v1 = cv2.normalize(src=new\_original\_band, dst=None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

#%%

h1= hsv\_image1[:, :, 0]

s1= hsv\_image1[:, :, 1]

new\_hsv= cv2.merge((h1,s1,v1))

final\_rgb = cv2.cvtColor(new\_hsv, cv2.COLOR\_HSV2BGR)

# Displaying the image using cv2.imshow()

cv2.imshow('Histogram\_Equalization\_Image1', final\_rgb)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

# %%

#Collecting values from 2D array

image\_first\_band = final\_rgb[:,:,2]

rows = image\_first\_band.shape[0]

coloumns= image\_first\_band.shape[1]

values = []

for i in range(0, rows):

    for j in range(0, coloumns):

        values.append(image\_first\_band[i, j])

frequencies = {x:values.count(x) for x in values}

import collections

od = collections.OrderedDict(sorted(frequencies.items()))

#%%

#Histogram

import matplotlib.pyplot as plt

ax = plt.subplot(111)

w = 0.3

ax.bar(list(od.keys()), list(od.values()) , width=w, color='b', align='center')

ax.autoscale(tight=True)

plt.title("Histrogram of\_histogram\_specified\_image01")

plt.xlabel("Pixel Intensity")

plt.ylim(0, 7000)

plt.ylabel("Pixel Frequency")

plt.show()

#%%

#CDF

import numpy as np

probability = []

for item in list(od.values()):

    probability.append(item/sum(list(od.values())))

cp = np.cumsum(probability).tolist()

od\_list = list(od.keys())

amin, amax = min(od\_list), max(od\_list)

for i, val in enumerate(od\_list):

    od\_list[i] = (val-amin) / (amax-amin)

plt.xlabel("Pixel Intensity")

plt.ylabel("Cumulative Probability")

plt.title("Cumulative Distribution Function of histogram\_specified\_image01")

plt.plot(od\_list, cp, c='blue')

plt.show()

# %%

Resulting Image 01: #%%

#importing cv2

import cv2

# Using cv2.imread() method

img1 = cv2.imread('3\_2.jpg')

# Displaying the image using cv2.imshow()

cv2.imshow('3\_2.jpg', img1)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

#%%

hsv\_image1 = cv2.cvtColor(img1, cv2.COLOR\_BGR2HSV)

# Displaying the image using cv2.imshow()

cv2.imshow('HSV Image', hsv\_image1)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

# %%

#Creating normally distributed transformation function

import numpy as np

original\_value\_band = hsv\_image1[:, :, 2]

target\_value\_band= np.random.binomial(n=255, p=0.5, size=(228, 300))

target\_value\_band1 = np.uint8(target\_value\_band)

#%%

#Algorithm

import collections

def takeClosest(num,collection):

   return min(collection,key=lambda x:abs(x-num))

intensity1 = []

intensity2 = []

probability1 =[]

probability2 = []

values1 = []

values2 = []

intensity= []

final\_list =[]

rows1 = original\_value\_band .shape[0]

coloumns1= original\_value\_band.shape[1]

for i in range(0, rows1):

    for j in range(0, coloumns1):

        values1.append(original\_value\_band[i, j])

frequencies1 = {x1:values1.count(x1) for x1 in values1}

rows2 = target\_value\_band1.shape[0]

coloumns2= target\_value\_band1.shape[1]

for i in range(0, rows2):

    for j in range(0, coloumns2):

        values2.append(target\_value\_band1[i, j])

frequencies2 = {x2:values2.count(x2) for x2 in values2}

for item in range(0, 256):

    if item in frequencies1:

        intensity1.append(frequencies1[item])

    else:

        intensity1.append(0)

for item in range(0, 256):

    intensity.append(item)

    if item in frequencies2:

        intensity2.append(frequencies2[item])

    else:

        intensity2.append(0)

for item in intensity1:

    probability1.append(item/sum(intensity1))

for item in intensity2:

    probability2.append(item/sum(intensity2))

cp1 = np.cumsum(probability1).tolist()

cp2 = np.cumsum(probability2).tolist()

res = {cp2[i]: intensity[i] for i in range(len(cp2))}

for item in cp1:

    final\_list.append(res[takeClosest(item,cp2)])

#intensity is mapped to final list

# %%

resf= {intensity[i]: final\_list[i] for i in range(len(intensity))}

new\_original\_band= np.zeros((int(rows1), int(coloumns1)))

for i in range(0, rows1):

    for j in range(0,coloumns1 ):

        new\_original\_band[i, j] = resf[(original\_value\_band[i, j])]

v1 = cv2.normalize(src=new\_original\_band, dst=None, alpha=0, beta=255, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

#%%

h1= hsv\_image1[:, :, 0]

s1= hsv\_image1[:, :, 1]

new\_hsv= cv2.merge((h1,s1,v1))

final\_rgb = cv2.cvtColor(new\_hsv, cv2.COLOR\_HSV2BGR)

# Displaying the image using cv2.imshow()

cv2.imshow('Histogram\_Equalization\_Image1', final\_rgb)

#Maintain output window until user presses a key

cv2.waitKey(0)

cv2.destroyAllWindows()

# %%

#Collecting values from 2D array

image\_first\_band = final\_rgb[:,:,2]

rows = image\_first\_band.shape[0]

coloumns= image\_first\_band.shape[1]

values = []

for i in range(0, rows):

    for j in range(0, coloumns):

        values.append(image\_first\_band[i, j])

frequencies = {x:values.count(x) for x in values}

import collections

od = collections.OrderedDict(sorted(frequencies.items()))

#%%

#Histogram

import matplotlib.pyplot as plt

ax = plt.subplot(111)

w = 0.3

ax.bar(list(od.keys()), list(od.values()) , width=w, color='b', align='center')

ax.autoscale(tight=True)

plt.title("Histrogram of\_histogram\_specified\_image01")

plt.xlabel("Pixel Intensity")

plt.ylim(0, 7000)

plt.ylabel("Pixel Frequency")

plt.show()

#%%

#CDF

import numpy as np

probability = []

for item in list(od.values()):

    probability.append(item/sum(list(od.values())))

cp = np.cumsum(probability).tolist()

od\_list = list(od.keys())

amin, amax = min(od\_list), max(od\_list)

for i, val in enumerate(od\_list):

    od\_list[i] = (val-amin) / (amax-amin)

plt.xlabel("Pixel Intensity")

plt.ylabel("Cumulative Probability")

plt.title("Cumulative Distribution Function of histogram\_specified\_image01")

plt.plot(od\_list, cp, c='blue')

plt.show()

# %%

Resulting Image 01:

A person holding a basketball

Description automatically generated

Histogram

A screenshot of a cell phone

Description automatically generated

CDF:

A screenshot of a cell phone

Description automatically generated

Resulting Image 02:

A picture containing wall, indoor

Description automatically generated

Histogram histogram specified\_image 02:

A screenshot of a cell phone

Description automatically generated

CDF of histogram specified image:

A screenshot of a map

Description automatically generated

Between these three techniques of linear stretching, histogram equalization and histogram specification, it depends on image type and our desired output to find out which technique is good. In this case for first image, linear stretching technique worked well. For second image, again linear stretching technique worked better. After histogram equalization both images were too bright. So, I used a 2d array which is normally distributed as a target transformation function in order to implement histogram specification. But this time both images were too dark. I got better results when I didn’t use the normally distributed array but an array from a target image. Then histogram specification worked.

Histogram Specification while used target image’s array as transformation function:

A picture containing floor, building, ground

Description automatically generatedA baseball player in the middle of a crowd

Description automatically generated