# DIP Assignment #1

Muhammad Noman Shafqat SE5A 111572

## Problem 1:

Absolute difference seems to be the best option since the values can’t be negative and everything will remain in the image space. Whereas, in pixel by pixel subtraction difference will get lesser than zero so there is a loss of data. In division most of the data is lost, Unless the the pixel value of a pixel in image2 is at least 2 times the one in image1 there will be no change.In XOR there is also a chance of data loss since bitwise XOR between 1000 and 0100 gives us 1100 which is not the actual difference.

## Problem 2:

1. The picture will get more whitish.
2. The picture will get more whitish.
3. The picture will get more whitish but the impact will be more than that of (a).
4. The picture will be black because all the values will be 1.
5. The picture will be white because all the values will be 1.0 which is the maximum.

## Problem 3:

1. Picture will get darker all the pixel at 120 (grey) will move to 0(black) and white will move to 135(grey).
2. The picture will have a more range of colors than the original.
3. The picture will have a more range of colors than the original but the distribution will be according to the pixel value density.

## Problem 4:

The image got brighter after the first stretching but there was no change after second stretching.

### Code

|  |
| --- |
| def strech(image, cutoff):  b = 255  a = 0  # create histogram  hist, bins = np.histogram(image.flatten(), 256, [0, 256])  # calcualte cumulative sum of the histogram  cumsum = hist.cumsum()  # total number of pixels  sum = hist.sum()  # the pixels which lies at cuttoff  start = sum \* cutoff  end = sum \* (1 - cutoff)  print(start, end)  # the index out of 255 where the cumulative sum is greater than the cuttoffs  c = np.where(cumsum.flatten() > start)[0][0]  d = np.where(cumsum.flatten() > end)[0][0]  print(a, b, c, d)  # formulae for stretching  temp = cv2.subtract(image, int(c))  temp = cv2.multiply(temp, (255 / (d - c)))  temp = cv2.add(temp, a)  return temp.astype("uint8")  def problem4():  ein = cv2.imread("Einstein.PNG", 0)  # strech the image  result = strech(ein, .03)  # strech the rsultant image  result2 = strech(result, .03)  # plot  plt.subplot(311)  plt.hist(ein.ravel(), 256, [0, 255])  plt.subplot(312)  plt.hist(result.ravel(), 256, [0, 255])  plt.subplot(313)  plt.hist(result2.ravel(), 256, [0, 255])  #save  cv2.imwrite("problem4Strechingresult1.png", result)  cv2.imwrite("problem4Strechingresult2.png", result2)  plt.savefig('problem4StrechingHist', bbox\_inches='tight')  # cv2.imshow("In", ein)  # cv2.imshow("out", result)  # cv2.imshow("out2", result2)  # plt.show()  cv2.waitKey(0)  cv2.destroyAllWindows() |

### Output

|  |  |
| --- | --- |
| **problem4Strechingresult1.png** | **problem4Strechingresult2.png** |

|  |
| --- |
| **problem4StrechingHist** |

## Problem 5:

### Code

|  |
| --- |
| def brightness\_correction(image, percent, type):  factor = 1  # if you are to darken the image bt 0.15% then original will be multiplied bt .85  if (type == "dark"):  factor = 1 - percent  elif type == "brighten":  factor = 1 + percent  else:  print("Please send third parameter as dark or brighten")  return  # multiply the image with a factor of itself  image = cv2.multiply(image, factor)  return image  def problem5():  #read images  chld1 = cv2.imread("Child\_1.PNG", 0)  chld2 = cv2.imread("Child\_2.PNG", 0)  #perform correction  correctchld1 = brightness\_correction(chld1, .8, "dark")  correctchld2 = brightness\_correction(chld2, 2, "brighten")  #save  cv2.imwrite("problem5\_correction\_darken.png",correctchld1)  cv2.imwrite("problem5\_correction\_brighten.png",correctchld2)  '''  cv2.imshow("in1", chld1)  cv2.imshow("in2", chld2)  cv2.imshow("out1", correctchld1)  cv2.imshow("out2", correctchld2)  cv2.waitKey(0)  cv2.destroyAllWindows()  ''' |
|  |

### Output

|  |  |
| --- | --- |
| **Brightened** | **Darkened** |

## Problem 6:

### Code

|  |
| --- |
| def power\_law\_transform(image, c):  temp = cv2.pow(image.astype(float), c)  temp = np.clip(temp, 0, 255)  temp1 = temp.astype("uint8")  return temp1  def log\_transform(image, c):  # 1 added to avoid the log(0)  add1 = cv2.add(image, 1).astype(float)  # log taken  log = cv2.log(add1)  # multiplied with the constant  temp = cv2.multiply(log, c)  return temp.astype("uint8")  def problem6():  chld1 = cv2.imread("Child\_1.PNG", 0)  powerchild = power\_law\_transform(chld1, 1.1)  logchild = log\_transform(chld1, 30)  cv2.imwrite("problem6-logchild.png", logchild)  cv2.imwrite("problem6-powerchild.png", powerchild)  '''  cv2.imshow("problem6-logchild", logchild)  cv2.imshow("problem6-powerchild", powerchild)  cv2.waitKey(0)  cv2.destroyAllWindows()  ''' |
|  |

### Output

|  |  |
| --- | --- |
| Log transform with constant 30 | Power law transform with lambda 1.1 and c=1 |

## Problem 7:

### Code

|  |
| --- |
| def connected\_components(image):  # thresh the mask  ret, thresh = cv2.threshold(image, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  # find the connected compoenents  output = cv2.connectedComponentsWithStats(thresh, 4, cv2.CV\_32S)  # return (no\_of\_components,labeledmatrix)  return (output[0] - 1, output[1])  def find\_avg\_pixels\_per\_circle(n, matrix):  matrix = np.array(matrix)  # count non zero points in the foreground and divide them by number of (red or any) circles  nonzero = matrix.nonzero()  return (len(nonzero[0]) / n)  def problem7():  chld1 = cv2.imread("blobs.PNG", 1)  # areas of interest where respective circles are located  blueMask = cv2.inRange(chld1, np.array([200, 0, 0]), np.array([255, 100, 100]))  redMask = cv2.inRange(chld1, np.array([0, 0, 200]), np.array([100, 100, 255]))  yellowMask = cv2.inRange(chld1, np.array([0, 200, 200]), np.array([100, 255, 255]))  # finding the number of componenrts and average pixels  noBlue, matrix = connected\_components(blueMask)  avgBluePix = find\_avg\_pixels\_per\_circle(noBlue, matrix)  noRed, matrix = connected\_components(redMask)  avgRedPix = find\_avg\_pixels\_per\_circle(noRed, matrix)  noYellow, matrix = connected\_components(yellowMask)  avgYellowPix = find\_avg\_pixels\_per\_circle(noYellow, matrix)  # taking suareroot and diplaying the results  print("Red: \t\tNo of circles = ", noRed, "\t\t Average area(pixels^2)", np.math.sqrt(avgRedPix))  print("Blue: \t\tNo of circles = ", noBlue, "\t\t Average area(pixels^2)", np.math.sqrt(avgBluePix))  print("Yellow:\t\tNo of circles = ", noYellow, "\t\t Average area(pixels^2)", np.math.sqrt(avgYellowPix))  cv2.imwrite("problem7-blueMask.png", blueMask)  cv2.imwrite("problem7-redMask.png", redMask)  cv2.imwrite("problem7-yellowMask.png", yellowMask)  '''cv2.imshow("problem7",chld1)  cv2.imshow("blueMask",blueMask)  cv2.imshow("redMask",redMask)  cv2.imshow("problem7yellowMask",yellowMask)  cv2.waitKey(0)  cv2.destroyAllWindows()  ''' |

### Output:

|  |
| --- |
| BLUE MASK |
| Red Mask |
| Yellow Mask |
| Console Output Red: No of circles = 25 Average area(pixels^2) 28.939937802282852  Blue: No of circles = 26 Average area(pixels^2) 27.849872254920392  Yellow: No of circles = 20 Average area(pixels^2) 27.160633276858626 |

## Problem 8:

### Code

|  |
| --- |
| boxPoints = [] # points of the selected box  currmouse = [] # curr mouse position  isCropping = False  # function called whenever mouse is moved  def mouseEvent(event, x, y, flags, param):  global boxPoints, isCropping, currmouse  currmouse = [(x, y)]  # when button down update first box point when up update the last  if event == cv2.EVENT\_LBUTTONDOWN:  boxPoints = [(x, y)]  isCropping = True  print("Croppping True")  elif event == cv2.EVENT\_LBUTTONUP:  boxPoints.append((x, y))  isCropping = False  print("Croppping False Area Selected= ", boxPoints)  def equalize(img):  img2 = np.array(img)  img = cv2.equalizeHist(img2)  return img  def problem8():  global boxPoints, isCropping, currmouse  #read and make a copy  image = cv2.imread("child.png", 0)  original = image.copy()  #create a window and set a mouse callback function  cv2.namedWindow("image")  cv2.setMouseCallback("image", mouseEvent)  # keep looping unless escapre key is pressed  while True:  # if user is cropping show a bax around the selected area  if isCropping == True:  # show slecting box  image = original.copy()  print(currmouse)  cv2.rectangle(image, boxPoints[0], currmouse[0], (255, 0, 0), 2)  cv2.imshow("image", image)  pkey = cv2.waitKey(33)  # if user presses enter crop the selected area  if pkey == 13:  if len(boxPoints) == 2:  # fetch Region of Interest from the boxPoints  roi = original[boxPoints[0][1]:boxPoints[1][1], boxPoints[0][0]:boxPoints[1][0]]  # equalize the image  roi = equalize(roi)  # replace in the original  original[boxPoints[0][1]:boxPoints[1][1], boxPoints[0][0]:boxPoints[1][0]] = roi  #reset  boxPoints = []  image = original.copy()  cv2.imshow("image", original)  cv2.setMouseCallback("image", mouseEvent)  elif pkey == 27:  break  else:  continue  time.sleep(.1)  cv2.destroyAllWindows() |

### Output:

Interactive screen where you can choose region of interest where normalization will be applied.

## Problem 9:

### Code

|  |
| --- |
| def problem9():  image = cv2.imread("script.PNG", 0)  # invert the image  image = np.invert(image)  # Blurr and sharpen the image three times  blurr = cv2.GaussianBlur(image, (25, 25), 10.0)  image = cv2.add(image, cv2.subtract(image, blurr))  blurr = cv2.GaussianBlur(image, (19, 19), 10.0)  image = cv2.add(image, cv2.subtract(image, blurr))  blurr = cv2.GaussianBlur(image, (15, 15), 10.0)  image = cv2.add(image, cv2.subtract(image, blurr))  # Blur the images to remove the isolated dots.  image = cv2.GaussianBlur(image, (5, 5), 10.0)  # threshhold the pictures  blackmask = cv2.inRange(image, np.array([200]), np.array([255]))  ret, thresh = cv2.threshold(blackmask, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  # find connected components  no, matrix = connected\_components(thresh)  # crop each connected component and save  for i in range(1, no):  temp = np.where(matrix == i)  a = np.min(temp[0])  c = np.max(temp[0])  b = np.min(temp[1])  d = np.max(temp[1])  roi = thresh[a :c , b :d ]  if (len(roi) > 0):  cv2.imwrite("Problem 10-img" + str(i) + ".png", roi)  cv2.imwrite("problem10-result.png", thresh)  '''cv2.imshow("thresh", thresh)  cv2.imshow("sharpened", image)  cv2.waitKey(0)  cv2.destroyAllWindows() ''' |

### Output:

|  |
| --- |
|  |
|  |