# Assignment 2

Efe Özden, 21946495
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
b21946495@cs.hacettepe.edu.tr

November 18, 2022

## 1 Introduction

In this assignment, our goal is to implement the color transfer method of Reinhard et al. and analyze RGB histograms of before and after images. Moreover, we will improve this method by transferring the color between the similar regions of source and target images.

## 2 Experiment

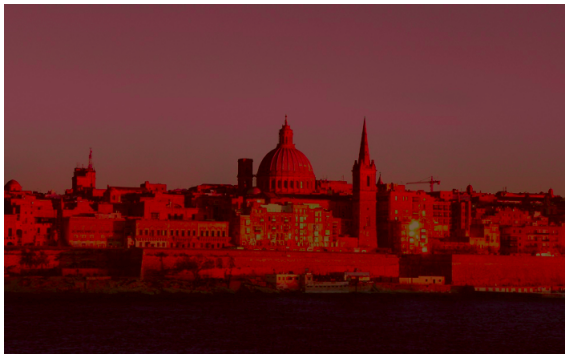
### 2.1 Part 1

I wrote a function which called image-stats for obtaining mean standard deviation for l,a,b values.

```
[4]: import numpy as np
     import cv2
     import os

[5]: def image_stats(image):
         # compute the mean and standard deviation of each channel
         (l, a, b) = cv2.split(image)
         (lMean, lStd) = (l.mean(), l.std())
         (aMean, aStd) = (a.mean(), a.std())
         (bMean, bStd) = (b.mean(), b.std())
         # return the color statistics
         return (lMean, lStd, aMean, aStd, bMean, bStd)
```

```
•[7]: #PART 1 ------------------------------------------------------------
files = [f for f in os.listdir('./data/')]

for file in files:
    if not file.endswith(".png"):
        continue

    name = file.split('.')[0]
    if "in" not in name:
        continue

    label,number = name.split('_')
    source = cv2.imread('./data/'+name + '.png')
    source = cv2.cvtColor(source,cv2.COLOR_BGR2LAB).astype("float32")
    target = cv2.imread('./data/tar_'+ number + '.png')
    target = cv2.cvtColor(target,cv2.COLOR_BGR2LAB).astype("float32")

    (lMeanSrc, lStdSrc, aMeanSrc, aStdSrc, bMeanSrc, bStdSrc) = image_stats(source)
    (lMeanTar, lStdTar, aMeanTar, aStdTar, bMeanTar, bStdTar) = image_stats(target)


    # subtract the means from the target image
    (l, a, b) = cv2.split(source)
    l -= lMeanSrc
    a -= aMeanSrc
    b -= bMeanSrc
    # scale by the standard deviations
    l = (lStdTar / lStdSrc) * l
    a = (aStdTar / aStdSrc) * a
    b = (bStdTar / bStdSrc) * b
    # add in the source mean
    l += lMeanTar
    a += aMeanTar
    b += bMeanTar
    # clip the pixel intensities to [0, 255] if they fall outside
    # this range
    l = np.clip(l, 0, 255)
    a = np.clip(a, 0, 255)
    b = np.clip(b, 0, 255)
    # merge the channels together and convert back to the RGB color
    # space, being sure to utilize the 8-bit unsigned integer data
    # type
    result = cv2.merge([l, a, b])
    result = cv2.cvtColor(result.astype("uint8"), cv2.COLOR_LAB2BGR)

    #save image
    cv2.imwrite('result_part1/res_'+ number + '.png',result)
```

Results of Part 1:



## 2.2 Part 2

For Part 2, First I resize target images acording to input images. Then I divided source and target images in to regions.

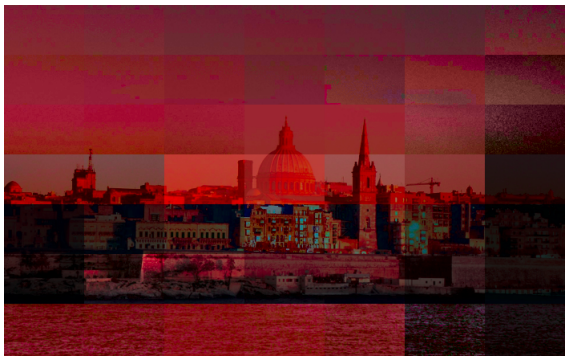
```
#PART 2 ------------------------------------------------------------
files = [f for f in os.listdir('./data/')]

for file in files:
    if not file.endswith(".png"):
        continue

    name = file.split('.')[0]
    if "in" not in name:
        continue

    label,number = name.split('_')
    source = cv2.imread('./data/'+name + '.png')
    source = cv2.cvtColor(source,cv2.COLOR_BGR2LAB).astype("float32")
    target = cv2.imread('./data/tar_'+ number + '.png')
    target = cv2.cvtColor(target,cv2.COLOR_BGR2LAB).astype("float32")


    (hs, ws) = source.shape[:2]

    #resize target image acording to source image
    target =  cv2.resize(target, (ws, hs))

    c = 0
    v = 0

    if hs > ws:
        c = hs // ws
        v = hs // 100
    else:
        c = ws // hs
        v = ws // 100



    w, h = (ws // v * c), (hs // v)

    result = np.zeros((hs,ws,3), np.uint8)
```

```
  w, h = (ws // v * c), (hs // v)

  result = np.zeros((hs,ws,3), np.uint8)

  for x in range(0, ws , w):
      for y in range(0, hs, h):

          source_p = source[y : y + h, x : x + w]
          target_p = target[y : y + h, x : x + w]

          (lMeanSrc, lStdSrc, aMeanSrc, aStdSrc, bMeanSrc, bStdSrc) = image_stats(source_p)
          (lMeanTar, lStdTar, aMeanTar, aStdTar, bMeanTar, bStdTar) = image_stats(target_p)


          # subtract the means from the target image
          (l, a, b) = cv2.split(source_p)
          l -= lMeanSrc
          a -= aMeanSrc
          b -= bMeanSrc
          # scale by the standard deviations
          l = (lStdTar / lStdSrc) * l
          a = (aStdTar / aStdSrc) * a
          b = (bStdTar / bStdSrc) * b
          # add in the target mean
          l += lMeanTar
          a += aMeanTar
          b += bMeanTar
          # clip the pixel intensities to [0, 255] if they fall outside
          # this range
          l = np.clip(l, 0, 255)
          a = np.clip(a, 0, 255)
          b = np.clip(b, 0, 255)
          # merge the channels together and convert back to the RGB color
          # space, being sure to utilize the 8-bit unsigned integer data
          # type
          result[y : y + h, x : x + w] = cv2.merge([l, a, b])
```

I obtained height/width ratio and assigned it in to c variable. Then I divided largest value(width or height) by 100 and assigned the result to the variable v.

Finally I divided width to v*c regions and height to c regions.

Results of Part 2:



## 3 Conclusion

Some images for Part 1 were not very close to the desired result. Although I have obtained results closer to the desired result in terms of color in Part 2, the results are still not satisfactory. Even smoother results can be obtained by dividing images into more parts.