CENG 562 HW2 QUESTION 1 REPORT

Burak Eren Dere*
Department of Computer Engineering, METU–2098929
(DECEMBER 12, 2020)

In this report, I am going to explain how I solved question-1 in hw2.

I. EXPLANATION

I have chosen to implement k-means clustering in language I like the most(c++). For image compression, the program I wrote only uses png images for compression.In my c++ project, for reading and manipulating pixels of png images, I have used stb_image_write and stb_image_ header files from Sean Barret's github repository[1]. On top of this basic png read and write functions, I have first implemented a wrapper class which I've chosen to call PngHandler, and then I have implemented another class called KMeansImageCompress. My code basically takes 4 arguments;

- path to input image
- path to output image
- C number (for dividing image into CxC pieces)
- K number (how many means we are going to use in k-means algorithm)

I have experimented with the results and calculated, errors and compression rates for different K and C numbers. As an example this is how I run my program compiled on my mingw64 windows machine;

```
C:\users\Eren\ImageCompress> mingw32-make // running Makefile
C:\users\Eren\ImageCompress> .\hw2.exe lenna.png lennaCompressed.png 32 512
```

similarly on my Ubuntu Focal machine;

```
eren@eren:~/ImageCompress make // running Makefile
eren@eren:~/ImageCompress ./hw2 lenna.png lennaCompressed.png 32 512
```

I wanted to run my code on our inek machines but external.ceng server unexpectedly keeps refusing my ssh and sftp attempts. I could not figure out the problem. So I unfortunately could not test it on our lab machines. However, I tested it on my home desktop, ubuntu 20.4 and win10 and it ran without errors.

A. Implementation Details

When reading png files with STB library, it stores all the pixels in that image on an allocated 8-bit unsigned integer array where r, g and b colors are stored in groups of 3. By using this knowledge, I could reach all the pixels in the images by just calculating the required index for that array. So, I did not need to store duplicate information. The only duplication I have is a deep copy of pixel array because I use it for error correction. When I need to divide my image into squares, I just store start and end indexes of pixels and get the colors from the 8-bit unsigned integer array. Since each part runs k-means on different parts of the image array, it is also easy to parallelize these jobs. I run each part on a separate thread to make it faster.

My k-means implementation first randomly creates K mean objects on pixel color space. And each mean object naively searches all available pixel colors and calculates euclidean distances. In every iteration of k-means algorithm, each mean object owns pixels that are more close to them than other mean objects around. At the end of the iteration, mean objects calculates the mean of their member pixels' colors and changes their means accordingly. In my implementation, I assumed that 10 k-means iterations are enough for this. I could also introduce a threshold to stop the iteration but I thought that 10 iterations would be enough. On the other hand, maybe my algorithm of

* eren.dere@metu.edu.tr

finding the means and distances may be a poor implementation. Because I experienced higher run times for increased K and image values. For small k values, complexity is nearly linear $O(n)$ but for increased K values it can even be close to a $O(n^2)$ algorithm. For accelerating it, we may use kd-trees like we did in $Ceng477$ back in bachelor's or some kind of 3D voronoi diagrams but I thought it would be overkill for this task.

At the end, I compare some kind of diff, pixel by pixel with the image I produced and initial image and calculate the error from it. I could not find a specific information on the internet how to take find error numerically so I came up with this idea. The euclidean distances Each pixels' colors in initial and final image are calculated and added. at the end that summation is divided to $height * width * 50$ and we get the error number.

B. Results

I experimented with image of Lena Forsén[2] which is a standard image processing test image according to what I've learned.

FIG. 1: lenna original image

Firstly, I kept increasing C values while keeping K's constant and calculated, compression rates and errors and examined the images. Here are the images I got;

(a) K is 4 and from left to right c=16,32,64

(b) K is 4 and from left to right c=128,256,512

FIG. 2: Increasing c while keeping k constant

As we can see by examining the images, image gets further away from the original one if we increase the C value while keeping K constant. The reason behind this that by increasing C, we are increasing the local pixel space so that mean object selects more general colors for their cluster members. As seen on the figure below, experiment I did by calculating errors validates our observations. When we increase C numbers, error increases.

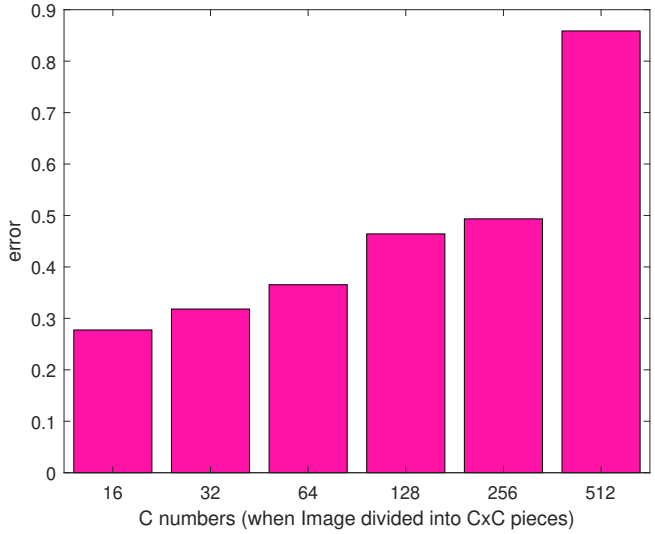


FIG. 3: increase C errors

On the other hand, when C is increased, I have found that compression tends to be better. According to my findings when $C = 256$ compression becomes worse but it does not change the fact that there is a trend going downwards. Here is my finding about compression;

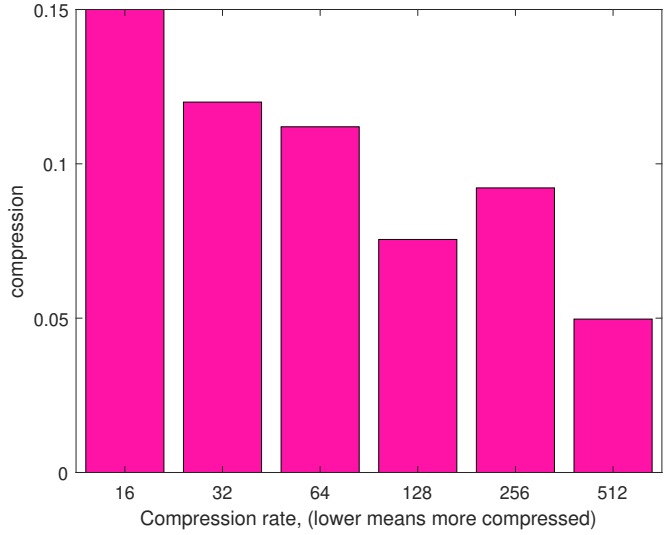
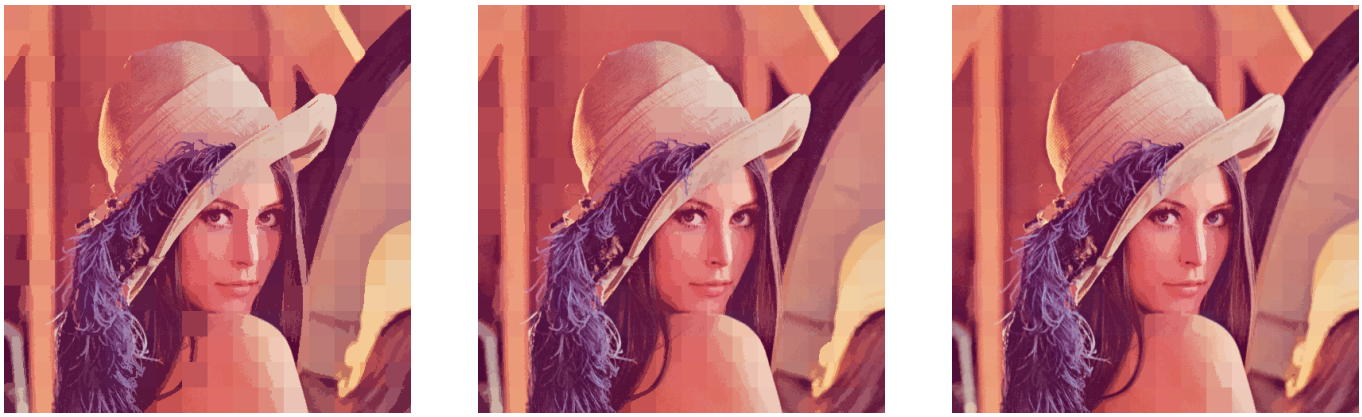


FIG. 4: increase C compression

Secondly, I started playing with K numbers and keeping C numbers constant. At first observation, While increasing k, output images started looking more similar to the input image. These are the images I got;



(a) c is 32 and from left to right k=8,16,32

(b) c is 32 and from left to right k=64,128,256

(c) c is 32 and from left to right k=128,256,512

FIG. 5: Increasing k while keeping c constant

As we can see by examining the images, image gets closer to the original one if we increase the K value while keeping C constant. The reason behind this that by increasing K, we are increasing the number of colors to be used in every piece so that k-means algorithm selects closer colors to the original image. As seen on the figure below, experiment I did by calculating errors validates our observations. When we increase K numbers, error decreases.

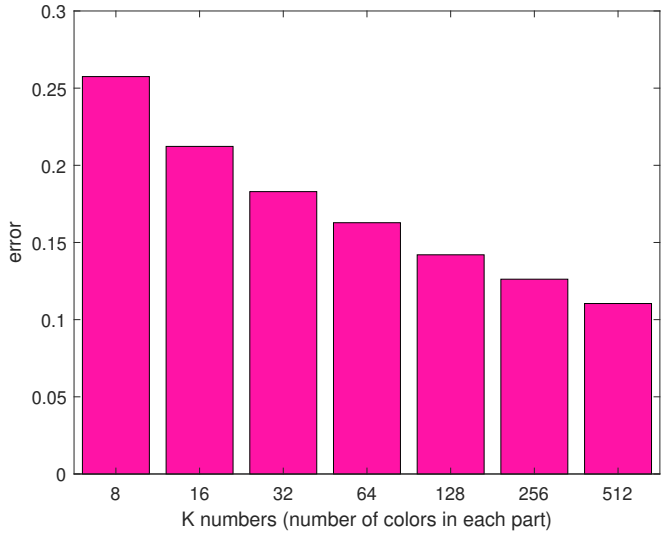


FIG. 6: increase K errors

On the other hand, when K is increased, I have found that compression tends to be worse. As I understood, more color means more space in png images. Here is my finding about compression;

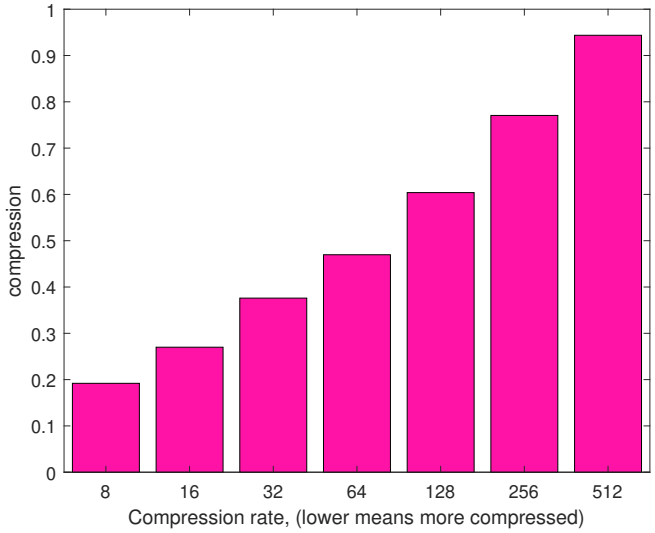


FIG. 7: increase K compression

C. Conclusion

In this report, I have explained how I implemented question 1 for Hw2. From this part of the homework, I have learned how to implement a k-means algorithm from scratch using c++ and how to make image compression. I have also understood that, higher k means lower error and worse compression and higher c means higher error and better compression.

---

[1] S. Barret, Image loader and image writer libraries, public domain (2013).
[2] wiki, lenna (2020).