# CENG 414

## Special Topics in Ceng: Introduction to Data Mining

Spring 2020-2021

Programming HW2

Görkem Özer
gorkem@ceng.metu.edu.tr

Due date: June 27, 2021 Sunday, 23:59

## 1 Overview

In this assignment, you are going to do **color quantization** for images, by implementing K-means algorithm by yourself in Python 3. Color quantization is done for reducing the number of colors used in an image. It is widely used in daily life. Camera lenses can detect millions of colors, however, some sort of color quantization must be done before displaying images on the screen, due to video memory limitations.

(a) Original Image

(b) Quantized image with 6 colors, 20 iterations

(c) Quantized image with 24 colors, 20 iterations

Figure 1: Some images for color quantization

**Keywords:** *Clustering, K-means algorithm, color quantization, Python*

## 2 Tasks

You are expected to implement K-means algorithm first, then you will do extra coding for using your implementation for color quantization.

### 2.1 Task 1 - K-Means Implementation (70 Pts.)

You will implement basic 2 functions for the K-means algorithm. There will be **fit()** and **predict()** functions that you will code and use in color quantization. Here are the K-means model parameters (simplified for this homework):

1. **3rd party Library Usage:** You can only use **numpy**. No other 3rd party is allowed for K-means implementation, since that would be easy-peasy.
2. **Input list:** K-means model should take a numpy array. **This array must be in this dimension: (1 X N)**
3. **Number of clusters:** K-means model should know the number of clusters beforehand.
4. **Maximum number of iterations:** K-means model should know where to stop. There are 2 scenarios, and one of them is halting after some number of epochs (or iterations).
5. **Epsilon value:** K-means model should know where to stop and this is the 2nd scenario. Fitting should end when none of the cluster centers have moved significantly after an epoch. Significance will be given via epsilon value parameter. If you have 3 cluster centers and all 3 of them moved a distance smaller than *epsilon*, then your model should halt.

   For example, if maximum number of iterations is given as 50, but the model converges (check epsilon) after 34 epochs, then the model should halt.
6. **Distance function:** You will allow your K-means implementation to execute 2 types of distance functions: **Manhattan distance** and **Euclidian distance** functions. This should be provided in K-means model initialization phase, with a string "manhattan" or "euclidian".

Example call for custom K-means implementation is like the following:

```
model = kmeans.KMeans(
    X=np.array(inp),
    n_clusters=5,
    max_iterations=100,
    epsilon=0.001,
    distance_metric="euclidian"
)

model.fit()
cluster_index = model.predict( (10, 140, 250) )
```

Please note that cluster indices are between [**0, N-1**]. Also note that **predict()** function can take a single instance of input, which can be in any format (list, tuple, single numeric value).

### 2.2 Task 2 - Color Quantization (30 Pts.)

- Now consider your input as list of tuples containing 3 values (R, G, B) where each R, G, B value can take numbers between [**0-255**].

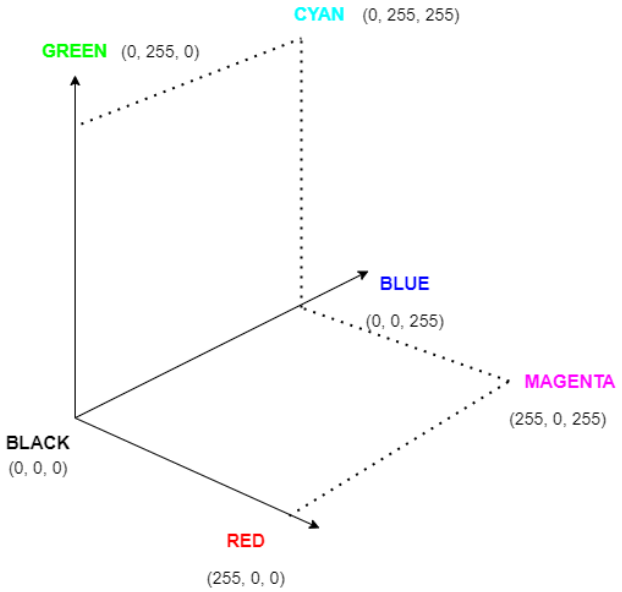


Figure 2: RGB Color Space

- You will use **python-pillow** library to read and process the image. Your color quantization implementation will base on the K-means implementation you did in part 1.

```
# read image pixels
# and have a list in 1 X (width*height) dimensions

model = kmeans.KMeans(
    X=np.array(inp),
    n_clusters=5,
    max_iterations=100,
    epsilon=0.001,
    distance_metric="euclidian"
)

model.fit()

# replace image pixels with color palette
# (cluster centers) found in the model

# save the final image
```

- Saved image name format should be:

```
Number of colors: 6
Max iterations: 20
Epsilon: 0.001
Image:  test.png
Output: test.png_6_colors_20_epochs_epsilon_0.001.png
```

- **python-pillow hints:** Use **getpixel()** for getting the RGB values for a pixel, **putpixel()** to replace the RGB value for a pixel, and **save()** to save the processed image.

# 3 Submission

1. You should submit all of your source files. If you have added an additional source file other than **main.py** and **kmeans.py**, make sure that everything works fine when the command below is executed:

```
>_ python main.py <image_file_name> <number_of_colors> <max_iterations> <epsilon>
```

2. Zip all task directories and name it as <ID> _ <FullNameSurname> and submit it through **ODTUClass**. **Replace with your ID number and your name.** For example:

`e1234567_GorkemOzer.zip`

3. Late submissions will **NOT** be accepted.