



# Image Compression using GMM

---

## Submitted By –

Ahire Abhay Madhukar	20112008
Golu Jatav	20112045
Adarsh	20112006
Aditya Dawar	20112007
Chandan Meena	18112017

## The motivation of the problem –

The Internet is loaded up with an enormous amount of data as images. People upload millions of pictures every day on social media sites such as Instagram and Facebook and cloud storage platforms such as google drive, etc. With such large amounts of data, image compression techniques become essential to compress the images and reduce storage space.

In this project, we have performed image compression using the GMM algorithm, an unsupervised learning algorithm.



## Methodology –

- First, the image is uploaded to the python notebook and then resized to required dimensions.

```
In [2]: img = io.imread('./original.jpg')  
img.shape
```

```
Out[2]: (360, 640, 3)
```

```
In [3]: plt.imshow(img)  
plt.axis('off')
```

```
Out[3]: (-0.5, 639.5, 359.5, -0.5)
```



```
In [5]: imgnew = img.reshape(img.shape[0]*img.shape[1],img.shape[2])  
imgnew.shape
```

```
Out[5]: (230400, 3)
```

- Preparing the model

```
In [6]: from sklearn.mixture import GaussianMixture  
gmm_5 = GaussianMixture(n_components=5, max_iter=10000, random_state = 10)  
gmm_5.fit(imgnew)  
labels_gmm_5 = gmm_5.predict(imgnew)
```

- GMM will group similar colors together into 'k' clusters (say k=5) of different colors (RGB values). Therefore, each cluster centroid is the representative of the color vector in RGB color space of its respective cluster.
- Now, these 'k' cluster centroids will replace all the color vectors in their respective clusters. Thus, we need to only store the label for each pixel which tells the cluster to which this pixel belongs. Additionally, we keep the record of color vectors of each cluster center.

```

In [8]: Ccentres_gmm_5 = gmm_5.means_.astype(int)
        Ccentres_gmm_5

Out[8]: array([[249, 204, 191],
               [ 73, 116, 130],
               [188, 135, 139],
               [ 93, 150, 152],
               [ 88, 49, 70]])

In [9]: newimage_gmm_5 = []
        for i in range(len(labels_gmm_5)):
            newimage_gmm_5.append(Ccentres_gmm_5[labels_gmm_5[i]])

In [10]: newimage_gmm_5 = np.array(newimage_gmm_5)
         newimage_gmm_5, type(newimage_gmm_5)

Out[10]: (array([[ 73, 116, 130],
                  [ 73, 116, 130],
                  [ 73, 116, 130],
                  ...,
                  [ 73, 116, 130],
                  [ 73, 116, 130],
                  [ 88, 49, 70]]),
          numpy.ndarray)

In [11]: newimage_gmm_5 = newimage_gmm_5.reshape(img.shape[0],img.shape[1],img.shape[2])
         newimage_gmm_5.shape

Out[11]: (360, 640, 3)

```

original



Compressed @ k=5



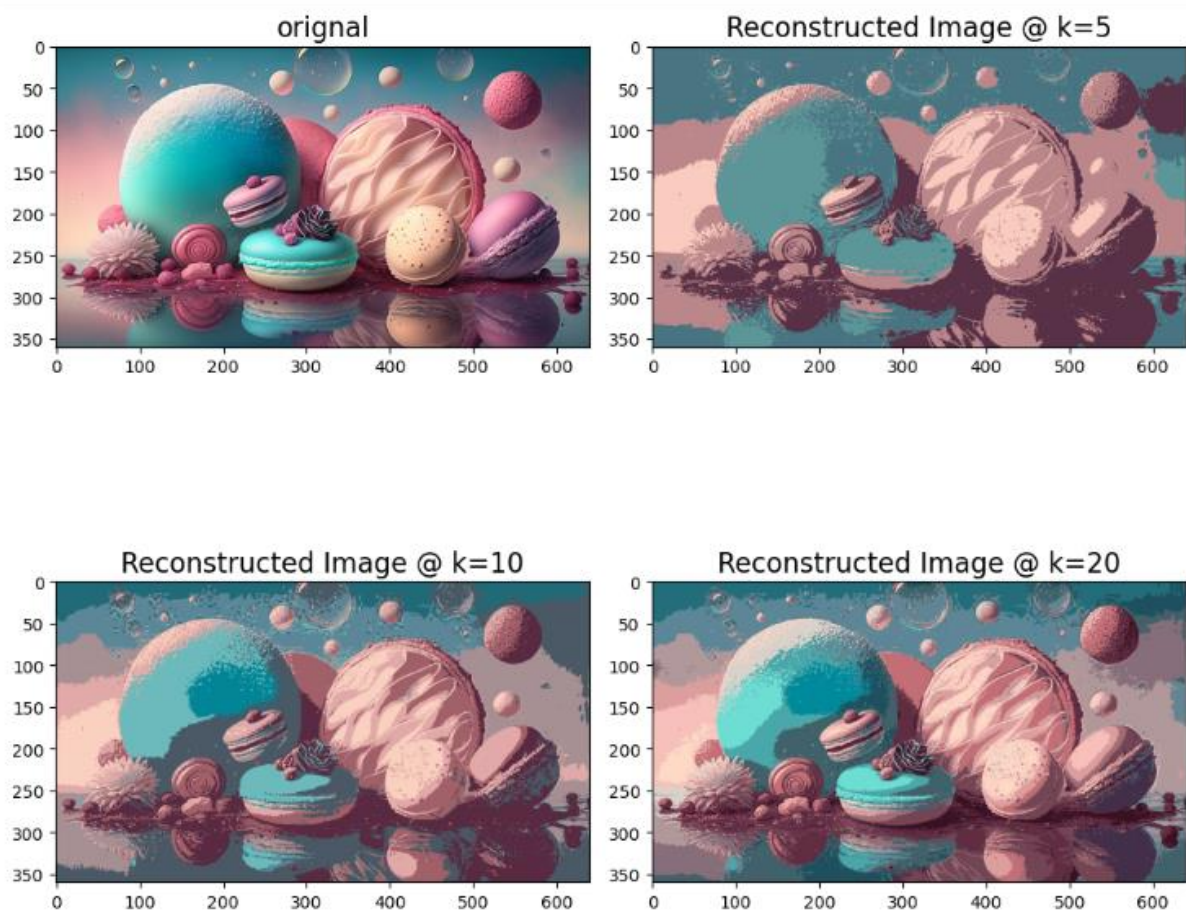


# Results and Observation –





- Original Image



- Compressed Image for different values of 'k'



- Details of all images

 original	Type: JPG File Dimensions: 640 x 360	Size: 79.7 KB
 compressed_image_5	Type: JPG File Dimensions: 640 x 360	Size: 49.4 KB
 compressed_image_10	Type: JPG File Dimensions: 640 x 360	Size: 48.3 KB
 compressed_image_20	Type: JPG File Dimensions: 640 x 360	Size: 49.7 KB

### Observations from above results –

- Dimensions of all the compressed images are same as that of input image (640 x 360).
- The size of compressed image decreases as k decreases.
- For value k=5, the compressed image lost most of its colours and the details of images are also lost.
- For k=10, the output compressed image still loses many colours and the details missed are easily visible to human eye.
- For k=20 the output compressed image seems reasonably good and the lose colours and highlights are seldomly visible to human eye.