Image Compression using Kmeans algorithm with pyspark and map and reduce techniques

Nishan Singh Dhillon
Eric Jonsson School of Engineering and Computer Science
University of Texas at Dallas
Richardson, Texas
nsd160330@utdallas.edu

***Abstract*— Image compression is a critical aspect when it comes to storage or transmission of data. KMeans is an efficient algorithm that can be used for this purpose. Further, this paper will implement KMeans using pyspark and Map and reduce techniques. Compression ratio will be calculated for different values of K for each of the images and ultimately in reference to the image quality, appropriate value of K will be discussed.**

Keywords—Kmeans, machine learning, image compression, compression ratio

I. INTRODUCTION (*HEADING 1*)

Image compression is one of the most widely used technologies used today when it comes to storage or data transmission, since a lot of applications or submission forms online that require image submission have a limit to the size of image that could be uploaded, image compression becomes a critical tool.

There are several algorithms that maybe used for image compression such as Huffman Coding, Arithmetic Coding and many more. Kmeans is one such algorithm that could be applied for this purpose. Originally developed in the machine learning field, K means segregates a set of 'n' dimensions into 'k' clusters and thereby each observation is assigned to any one cluster that is closest to the mean. Basically, K means helps in color quantization which causes to reduce the quantity of unique colors present in an image and hence reduce image size.

In most of the situations image compression comes at the cost of ruining the image quality. This is further classified as lossless or lossy compression. Lossless is a type of compression that makes sure that each bit is recovered once it is uncompressed, hence the information is completely restored. Lossy compression eliminates certain bits of information which may not be recovered and will be lost forever. Although, lossy compression leads to a more extensive compression of data, but image quality is significantly reduced.

MapReduce is programming concept that is used extensively in parallel computing especially when it comes to data processing. KMeans algorithm can also be implemented using MapReduce and in this paper, we will discuss the results. During the map phase, data points are assigned to the nearest centroids, while reduce phase involves calculating new centroid. This approach makes this algorithm useful for handling large data volumes.

II. KMEANS ALGORITHM

In this section, we will describe KMeans algorithm and how it could be implemented for image compression using map and reduce techniques. KMeans algorithm uses a K value which basically specifies the number of clusters. For instance, assume K = 2 for an image compression program, then any two pixels from the image will be the centroids and each of the other pixels will be classified to one of those two centroids. Similarly, number of centroids could be increased with increase in the value of K. The centroids are initially initialized randomly, and each of the other pixels are classified to the respective centroid based on Euclidean distance. Thereby, the RGB value of that respective pixel is replaced by the RGB value of the cluster it has been classified. This whole process could be implemented using Map and reduce techniques which will help to make the data processing more efficient. Ultimately, compression ratio is calculated by simply taking the ratio of original image and compressed image.

```
# Apply the function for all images and for each k value
for img_file in os.listdir(img_dir):
  if img_file.endswith(".jpg"):  # or ".png", ".jpeg", etc. as needed
      full_img_path = os.path.join(img_dir, img_file)
      for num_clusters in cluster_numbers:
        resultant_img_path = img_seg_with_kmeans(full_img_path, num_clusters)
        output_images[num_clusters] = resultant_img_path
        compression_val = calc_compression_ratio(full_img_path, resultant_img_path)
        compression_values[num_clusters] = compression_val
        print(f"Compression ratio for {img_file} with k = {num_clusters} is {compression_val}")
```

Fig. 1. Basic layout of the algorithm

```
for iteration in range(10):  # run 10 iterations
    # Assign each row in the DataFrame to the nearest centroid

    assigned_clusters = df.rdd.map(calculate_min_distance).toDF(["cluster_id", "coordinate"])

    # Recalculate the centroids
    new_centroids_list = assigned_clusters.rdd.map(lambda x: (x[0], (x[1], 1)))\
                                .reduceByKey(lambda x, y: ([x[0][i] + y[0][i] for i in range(len(x[0]))], x[1] + y[1]))\
                                .mapValues(lambda x: [x[0][i] / x[1] for i in range(len(x[0]))])\
                                .collect()
    new_centroids_list = sorted(new_centroids_list, key=lambda x: x[0])
    new_centroids_list = [x[1] for x in new_centroids_list]
    centroids_list = spark_session.sparkContext.broadcast(new_centroids_list)
```

Fig. 2. Implementing Map and reduce

```
# Replace the color of each coordinate with the color of its centroid
color_map = {i: color for i, color in enumerate(new_centroids_list)}
color_df = assigned_clusters.rdd.map(lambda x: (x[0], color_map[x[0]])).toDF(["cluster_id", "color"
modified_img_array = np.array(color_df.select('color').rdd.flatMap(list).collect())
```

Fig. 3. Replace the color of each color with the color of its centroid

```
def calc_compression_ratio(initial_path, final_path):
  original_file_size = os.path.getsize(initial_path)
  compressed_file_size = os.path.getsize(final_path)

  compression_ratio = original_file_size / compressed_file_size
  return compression_ratio
```

Fig. 4. Calculating compression ratio

III. Results and Analysis.

Seeing the result, we can analyze that the compression ratio is high for low values of K and low for high values of K. Also, the image quality increases with value of K but it stagnates after a point. In this scenario, compression ratio and the quality of image seems to stagnate around K = 12. Hence, it could be inferred that K = 12 is an appropriate value for compression of this image. Moreover, there are a few fluctuations in this trend. Firstly, compression ratio increases as value of K increases from 16 to 18. Secondly, compression ratio stays almost the same as K value increases from 24 to 28. The lowest compression value is reached at K = 30 at 2.47. Overall trend suggests that although higher value of K leads to better compression, but the gains seem to diminish after a certain point. Thereby, decision to choose K value should also be dependent on the computational resources available as processing of image with higher K value will require extensive computational resources and time.
Additionally, since quality of image decreases which implies the algorithm is lossy since it does not preserve the quality of image.

Table 1. K value vs compression ratio

K Value	Compression Ratio
2	5.14
4	3.69
6	3.32
8	2.89
10	2.72
12	2.71
14	2.68
16	2.54
18	2.56
20	2.48
22	2.47
24	2.49
26	2.49
28	2.49
30	2.47

Fig. 5. Original Image

Fig. 6. K = 2

Fig. 7. K = 4

Fig. 8. K = 6

Fig. 9. K = 8

Fig. 10. K = 10

Fig. 11. K = 12

Fig. 12. K = 14

Fig. 13. K = 16

Fig. 14. K = 18

Fig. 15. K = 20

Fig. 16. K = 22

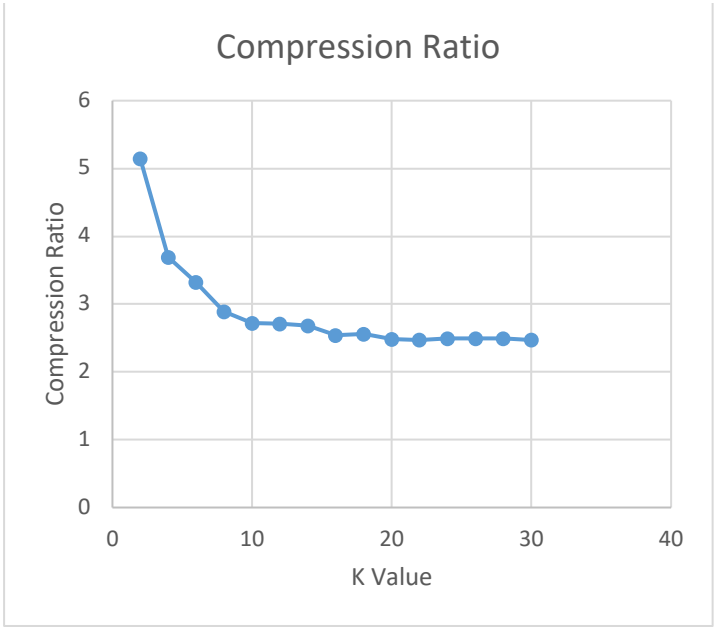
Fig. 17. K = 24

Fig. 18. K = 24

Fig. 19. K = 26

Fig. 20. K = 28

Fig. 21. K = 30



IV. Conclusion and Future Work

In conclusion from our study, we have noticed a particular pattern. As we increase the number of clusters, there's a corresponding reduction in the compression rate. Noticeably, this correlation reaches a stagnation point, beyond which additional clusters fail to decrease the compression ratio by a significant factor. Recognizing this limit could be an essential element in effectively managing computational resources when performing the compression process.

Moreover, it is observed that elevating the value of K, the number of clusters, has a positive effect on the image's quality. This means that the compressed image has a closer

resemblance to the original image as K is increased. This improvement, however, also hits a stagnation point, beyond which any sort of noticeable enhancement in the image quality is no longer significant. This gives us the idea that there is a peak K value beyond which there is not much significant improvement in image quality.

Discovering this optimal K value for each unique image emerges as an important task. This information could lead to a more judicious use of computational resources by eliminating excessive clustering that doesn't lead to significant gains in compression or image quality. It also helps strike the right balance between image quality and data volume needed to represent it which is basically the primary goal of image compression.

Further, future studies could focus on devising automated techniques to ascertain the best K value for any given image. It would be a good idea to investigate creating adaptive systems capable of guessing the color variance and intricacy in an image, hence estimating a suitable value of K. Also plans could be made to further investigate the various factors that may impact the value of compression ratio.

REFERENCES

[1] Tapas Kanungo, Nathan S. Netanyahu, Ruth Silverman, Angela Y. Wu, and Christine D. Piatko “An Efficient k-Means Clustering Algorithm: Analysis and Implementation” IEEE transactions on pattern analysis and machine intelligence, vol. 24, no. 7, july 2002