Visual Computing

T4: Image Segmentation

Authors:

Archit YADAV
Nairit BANDYOPADHYAY

Table of Contents

1. Introduction

1.1 Problem Statement

The objective of this lab is to study and apply k-means algorithm for the application of image segmentation. The main principle behind this algorithm is that every pixel of an image is associated with a group, and the number of such groups can be pre-defined for the algorithm.

Till now, we have been dealing with PGM images. Now we will deal with coloured images, which are stored in PPM format. So now we'll have a 3rd dimension for accessing one of the 3 components R, G and B. So, in order to access each pixel, we will have to iterate 3 loops. This can be depicted in the following code snippet-

```
for(int i = 0; i < rows; i++)
    for(int j = 0; j < cols; j++)
        for(int k = 0; k < 3; k++)
            pixel[k] = bitmap[(i * cols + j) * 3 + k];
```

1.2 Input Images

We will consider the following 2 images while experimenting with k-means clustering.

Figure 1: Original input images frog.ppm and nightking.ppm

2. The Method

The k-means clustering works in the manner that every pixel of an input image is part of a group, where the total number of possible such groups are equal to k. The algorithm goes to every pixel, and tries to associate the pixel with one of the k groups. If C_k is the k^{th} cluster, then all the pixels in the vicinity of the C_k become part of the C_k group. Vicinity can be determined by 2 ways - either by RGB values, or by cartesian coordinates. In the first part, we will focus on clustering on the basis of closeness with RGB values.

The main steps to follow for performing k-means clustering are as follows-

2.1 Initialize the k clusters

The RGB values of all the k clusters are initialized to a value between 0 to 255. There are several methods to do it, and theories behind it. We will try with initialization with zeros, as well as with random values.

We initialize all the 3 components, R, G, and B with 0.

```
void AllocateRandomClusters(bit* clusters, short clustercount)
{
  // Initializing clusters
  for(int i = 0; i < clustercount; i++)
  {
    for(int j = 0; j < 3 ; j++)
    {
      clusters[i * 3 + j] = 0;
    }
  }
}
```

2.2 Calculate distance from clusters

Our next task is to go over each pixel of the image, and calculate its RGB distance from each of the k clusters. The cluster Ck from which the distance is minimal of all is the cluster associated with the current pixel. We will store the index of the cluster in another matrix called as label matrix.

The RGB distance between the current pixel and cluster is calculated as follows-

$$dist = \sqrt{(R_c - R)^2 + (G_c - G)^2 + (B_c - B)^2}$$

Where R_c, G_c, B_c are the RGB values of the cluster, and R, G, B are the values of the current pixel in consideration.

This whole process can be depicted in the following diagram-

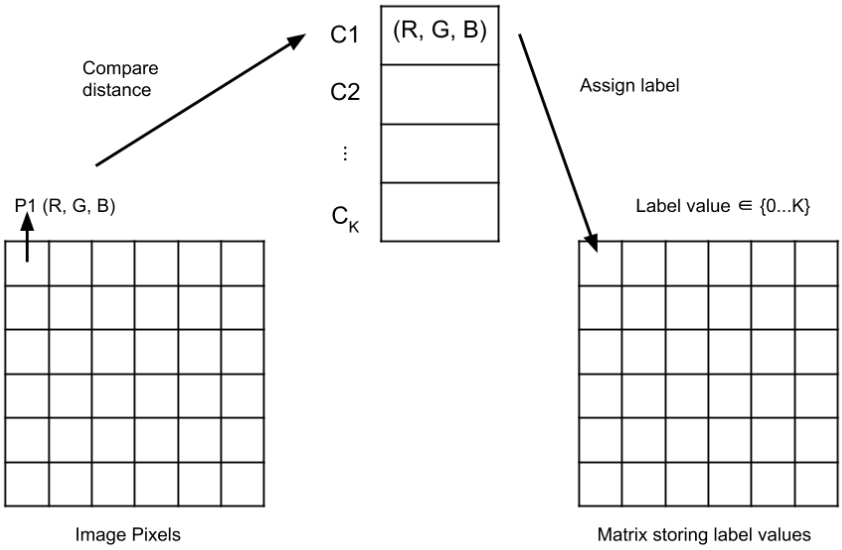

Figure 2: Distance calculation and label assignment

```
int GetClusterPixelIndex(bit *pixel, bit *clusters, short clustercount)
{
  int min_dist = -1, dist, clusterIndex;
  for(int i = 0; i < clustercount; i++)
  {
    dist = 0;
    for(int j = 0; j < 3; j++)
    {
      dist = dist + pow(pixel[j] - clusters[(i * clustercount + j)], 2);
    }
    dist = sqrt(dist);
    if(dist < min_dist || min_dist == -1)
    {
      min_dist = dist;
      clusterIndex = i;
    }
  }
  return clusterIndex;
}
```

2.3 Assign pixel to cluster

Now that we have got the index corresponding to the cluster the current pixel belongs to, we can extract the RGB values of the cluster and assign it to the current pixel. These RGB values are written to another variable clustermap.

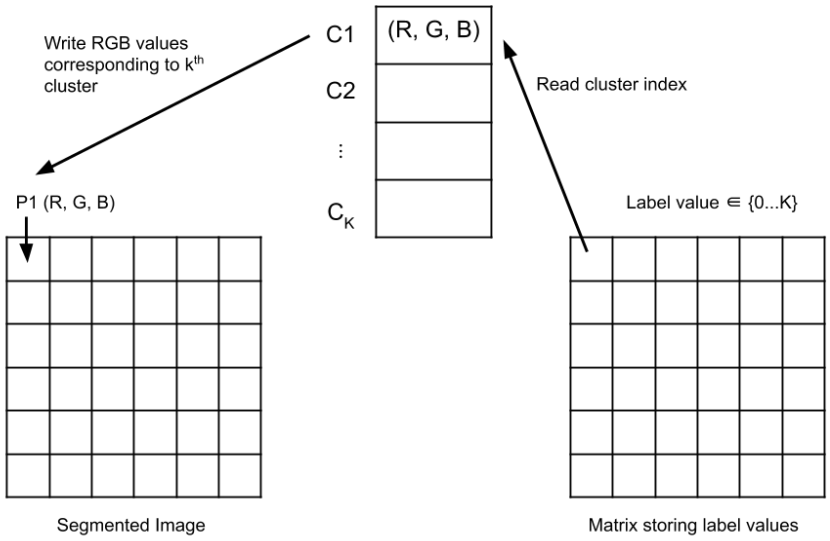

Figure 3: cluster index reading and RGB value extraction

```
void GetClusteredImage(bit *bitmap, bit *clustermap, bit *clusters,
bit *labels, short clustercount, int rows, int cols)
{
  bit pixel[3];
  int clusterIndex;
  for(int i = 0; i < rows; i++)
  {
    for(int j = 0; j < cols ; j++)
    {
      for(int k = 0; k < 3; k++)
        pixel[k] = bitmap[(i * cols + j) * 3 + k];

      clusterIndex = GetClusterPixelIndex(pixel, clusters, clustercount);

      for(int k = 0; k < 3; k++)
        pixel[k] = clusters[clusterIndex * 3 + k];
```

```
      for(int l = 0; l < 3; l++)
        clustermap[(i * cols + j) * 3 + l] = pixel[l];

      // Assign Label
      labels[i * cols + j] = clusterIndex;
    }
  }
}
```

2.4 Recalculate Clusters and Repeat

Now, in order to repeat the process, we need to update the RGB values of all the k clusters. This should take into account all the pixels associated with any particular clusters. For this, we will simply take the average of all the RGB values of all three pixels belonging to a cluster.

```
void    RecalculateClusters(bit    *bitmap,    bit    *clusters,    short
clustercount, bit *labels, int rows, int cols)
{
  for(int i = 0; i < clustercount; i++)
  {
    int count = 0;
    float pixelsum[3];
    for (int j = 0; j < 3; j++)
      { pixelsum[j] = 0; }
    for(int j = 0; j < rows; j++)
    {
      for(int k = 0; k < cols; k++)
      {
        if(labels[j * cols + k] == i)
        {
          for(int l = 0; l < 3; l++)
            { pixelsum[l] += bitmap[(j * cols + k) * 3 + l]; }
          count++;
        }
      }
    }
    for(int l = 0; l < 3; l++)
      clusters[i * 3 + l] = pixelsum[l]/count;
  }
}
```

3. Implementation

3.1 K-means with intensity values

So in order to finally implement K-means, we just put together all the modules we defined above. We will have to run it a certain number of times until the cluster values become stabilizing (to be precise, when the RGB values of the previous cluster and current cluster become the same). In order to better understand the effect of the number of iterations, we decided to have the number of iterations outputted.

```
void ApplyKmeans(bit *bitmap, bit *clustermap, bit *clusters, bit
*labels, short clustercount, int rows, int cols, int iterations)
{
  bit *prevClusters = (bit *) malloc(clustercount * 3 * sizeof(bit));

  AllocateRandomClusters(clusters, clustercount);

  for(int i = 0; i < iterations; i++)
  {
    //PrintFunc(clusters, clustercount, 3);

    if(ClusterCompare(prevClusters, clusters, clustercount, 3) && i !=
    0)
    {
      printf("No. of iterations: %d\n\n", i+1);
      break;
    }

    updateOldCluster(prevClusters, clusters, clustercount, 3);

    GetClusteredImage(bitmap, clustermap, clusters, labels,
    clustercount, rows, cols);

    RecalculateClusters(bitmap, clusters, clustercount, labels, rows,
    cols);
  }
}
```

Input Images:

Figure 4: Original input images frog.ppm and nightking.ppm

Output Images:

3.1.1 Centroids initialized to 0

Figure 5: **K=2** for frog (9 iterations) and nightking (13 iterations)


Figure 6: **K=3** for frog (11 iterations) and nightking (25 iterations)


Figure 7: **K=5** for frog (51 iterations) and nightking (50 iterations)


Figure 8: **K=10** for frog (78 iterations) and nightking (84 iterations)

Observations: We can see from Figure 6, 7, and 8 that increasing the number of clusters gives us more number of features in terms of colour regions. The number of iterations required for converging also increase with the increase of clusters.

3.1.2 Centroids initialized to MAX VAL (255)

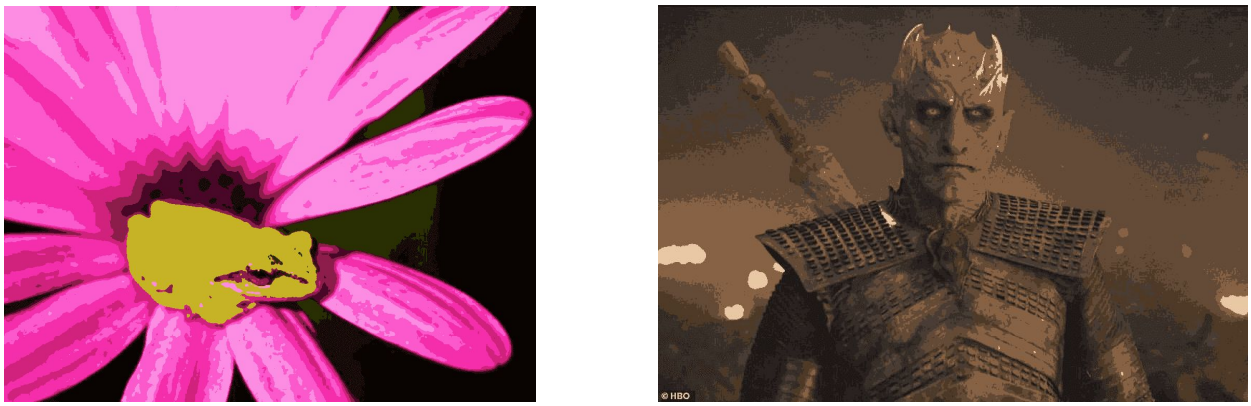

Figure 9: **K=10** for frog (78 iterations) and nightking (84 iterations)

Observation: We see that changing the initial value of centroids from (0, 0, 0) to (255, 255, 255) did NOT make any difference, by comparing Figure 8 and Figure 9. We got almost exact identical outputs, and the number of iterations taken to achieve this were also the same. This leads us to conclude that initialization to the same extreme values (0 and 255) does not make an impact on the choice of centroids and the resultant image.

3.1.3 Centroids initialized randomly

When we initialize cluster centres randomly, we observed some interesting results. The final output differed everytime we ran the program on the same input parameters. Let's consider the frog image only for the moment:

Figure 10: **K=3** for frog (7 iterations) and frog (23 iterations) **both using rand()**

Cluster #	R	G	B		Cluster #	R	G	B
1	249	224	26		1	51	188	200
2	14	200	224		2	49	154	203
3	144	69	76		3	76	209	106

Observation: We tried running the program several times in order to see the initial values of the centroid and try to establish a relation between the 2 observed images. One such set of values are shown in the table above. But it's really difficult to determine why the left set of cluster values are giving the more cleaner image (Figure 10 left) compared to the other image. What we can conclude though is that compared to fixed (0,0,0) values (Figure 6 left), randomized centres are giving better values (atleast maybe 50% of program executions), as seen in Figure 10 (left).

Now we consider The Night King image:

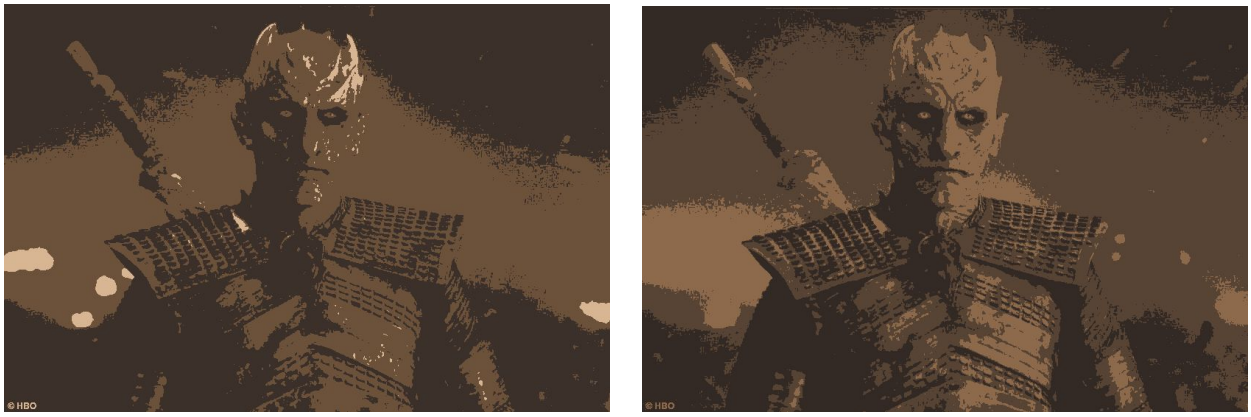

Figure 11: **K=3** for nightking (16 iterations) and nightking (20 iterations) **both using rand()**

Cluster #	R	G	B		Cluster #	R	G	B
1	173	64	57		1	63	223	123
2	117	236	230		2	138	130	150
3	45	103	188		3	27	77	16

Observations: Just like the frog image, the Night King image is also giving similar observations. The values initialized are inconclusive as to why we observe the way we

are observing. Before, we had concluded that random gives us better clustering. But now after applying the same randomized clustering on Night King, we observe that maybe it doesn't really make a difference. Figure 11 (right) is the same Figure 6 (right), which is possible because the values are random. In both these figures, there are NO white spots on the left part of the image. These white spots are present in Figure 11 (right), which is also a product of random clusters. Now these white spots are actually part of the original image (Figure 4 (right)), so we can't say for sure whether randomized cluster centres worked better for Night King image or not. It seems both the images in Figure 11 seem to be good enough. But it did seem to have worked better for the Frog image (Figure 10 left).

Bottomline conclusion - What initial values to use for segmentation using clustering really depends on the chosen input image, and we must experiment to arrive at the best possible result. Of course, the phrase 'best' is also subjective, as was seen in our Night King image comparisons.

Having answered the question ***"What is the influence of the initial values for region centers?"*** (in short, it produces different outputs), we now answer the next question -

What is the influence of the number of regions K?

Upon increasing the value of K, we get more segments (regions) in our image, and we also get to see more variety of colours, as can be seen by comparing Figure 5, 6, 7, & 8, for both Frog image as well as Night King image.