Lab 9 - K-means Clustering

I. Introduction

K-means clustering is one of the simplest and popular **unsupervised** machine learning algorithms. The objective of *K-means* is simple: group similar data points together and discover underlying patterns. To achieve this objective, *K-means* looks for a fixed number (*k*) of clusters in a dataset. A cluster refers to a collection of data points aggregated together because of certain similarities.

In this lab, we will write a program to segment different objects in a video using *K-means* clustering. There are several steps:

- [x] Load video & extract frames
- [x] Implement *K-means* clustering
- [x] Write back to video

II. Lab Practice

In this lab, you need following dependency:

```python
import numpy as np
import cv2
import tqdm
```

(1) Video Handling

We use `opencv` to handle videos.

For reading a video, we have:

```python
# load video, get fps and size of each frame
cap = cv2.VideoCapture(video_path)
fps = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))

# then, you can use read() to read a frame of a video
ret, frame = cap.read()
```

Pay attention that data type of `frame` is `uint8`, not `int`; in this lab, frame has 3 channels.

You can write a frame as following:

```
# intantiate a video write
video_writer = cv2.VideoWriter(filename,
                               cv2.VideoWriter_fourcc(*'mp4v'),
                               fps,
                               size,
                               isColor=True)
# create a new frame with dtype (data type) in uint8
new_frame = np.array((h, w, c)).astype("uint8")
video_writer.write(new_frame)
```

Pay attention: If you don't change `dtype` of frame into `unit8`, video you write will look strange which you can have a try.

(2) K-means

In this lab, you need to implement K-means, the rough procedure is:

1. **initialize centroids** of different classes

   In the simplest case, randomly choose centroids in original data

2. **calculate distances** between samples (pixels) and centroids

   Since one sample (pixel) has 3 channels, you can calculate square sum of differences in each channel between it and centroids.

$$dist(S,C) = \sum_{i=1}^{3}(C_i - S_i)^2$$

$$\begin{cases} dist(S,C) : \text{distance between a sample S and a centroid C} \\ C : \text{a centroid} \\ S : \text{a sample} \\ S_i : \text{the } i^{th} \text{ channel's value of S} \\ C_i : \text{the } i^{th} \text{ channel's value of C} \end{cases}$$

3. **classify** every samples

   A sample is belonging to the class whose centroid is closest to it among all centroids.

$$cls(S) = argmin(\sum_{i=1}^{3}(C_i^k - S_i)^2), k = 1, 2, \dots, K$$

$$\begin{cases} cls(S) : \text{class of a sample S} \\ K : \text{number of classes} \\ C^k : \text{centroid of } k^{th} \text{ class} \end{cases}$$

4. **update centroid**

   You can use mean of all samples in the same class to calculate new centroid.

$$C_i^k = \frac{1}{n^k}\sum_{n=1}^{n^k} S_{in}^k, \quad i = 1,2,3$$

$$\begin{cases} C_i^k : \text{the } i^{th} \text{channel's value of a centroid belonging to the } k^{th} \text{class} \\ n^k : \text{the number of samples in the } k^{th} \text{class} \\ S_{in}^k : \text{the } i^{th} \text{channel's value of a sample which is in the } k^{th} \text{class} \end{cases}$$

5. loop until classification result doesn't change

In addition, you may find there is code like this:

```
while ret:
    frame = np.float32(frame)
    h, w, c = frame.shape
    ...
```

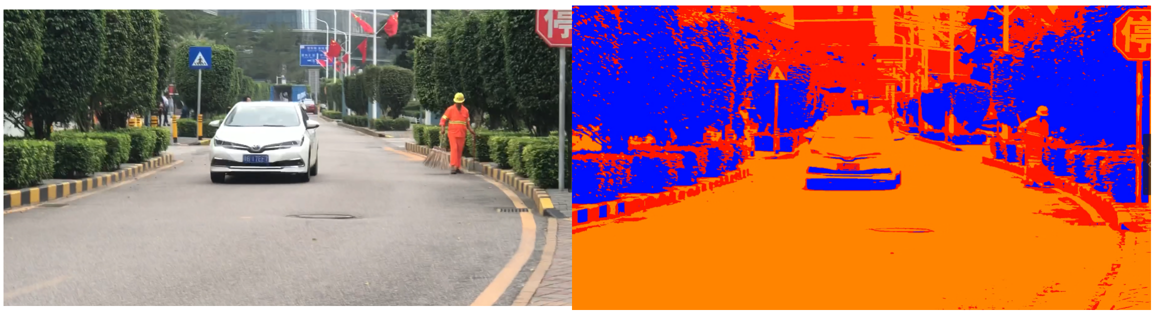
Since if you don't converse the `dtype`, K-means hardly converges which means it will stuck into dead loop easily.

After you finish K-means, you will find the written video is hard to watch because **color** between adjacent frames **changes almost all the time**. To alleviate this situation, you can use seed to initialize centroids which we offer you an interfere:

```
def update_seed(n_cl, label, distance)
```

It isn't compulsory, you can try if you want.

Sample Result



III. Lab Requirement

Please finish the **Exercise** and answer **Questions**.

Exercise

Complete the video segmentation algorithm with *K-means* and try different *k* to get better results.

Things you should submit:

- Files: Your code (*segmentation.py*)
- Video: your segmentation result (*segmentation_results_video.mp4*)
- Report: including the results and brief comments.

Questions

1. What are the strengths of K-means; when does it perform well?
2. What are the weaknesses of K-means; when does it perform poorly?
3. What makes K-means a good candidate for the clustering problem, if you have enough knowledge about the data?