پیاده سازی الگوریتم k-means

- **الگوریتم K-means :**

این الگوریتم یک نوع Unspervised خوشه بندی و شاید بتوان گفت یکی از ساده ترین روش های خوشه بندی است که بعضا بسیار کارا عمل می کند.

این الگوریتم به این صورت عمل می کند که به عنوان مثال یک سری داده در یک فضای n بعدی داریم و می خواهیم داده هایی که احتمالا به یک خوشه تعلق دارند را تشخیص دهیم:

- به تعداد خوشه های مورد نظر داده رندوم که نمایانگر خوشه ها هستند در همان فضا با همان ابعاد ایجاد می کنیم و در هر مرحله از اجرای الگوریتم به این صورت عمل می کنیم که :
  - فاصله (همانند محاسبه ی دو بردار n بعدی)هر کدام از داده ها را با تک تک مرکز خوشه ها پیدا میکنیم و آن داده را متعلق به خوشه ای می نامیم که نزدیکتر بوده باشد
  - وقتی همه ی داده ها label گذاری شدند مراکز خوشه جدید را به این صورت تعییین می کنیم: میانگین تمام داده های یک خوشه مرکز خوشه اعلام می شود

این الگوریتم را آنقدر اجرا می کنیم تا هیچ داده ای طی دو مرحله متوالی تغییری در خوشه بندی نداشته باشد.

این الگوریتم عموما همگراست ولی حالاتی هم وجود دارد که الگوریتم هیچ وقت هم گرا نشود.

من در این تمرین الگوریتم K-means را برای تفکیک اجزای یک عکس یا عمل Segmentation انجام دادم و چون تعداد پیکسل های عکس زیاد بود من این الگوریتم تا همگرا شدن انجام ندادم و فقط برای تعداد دورهای کم اجرا کردم.

عکس مورد نظر در زیر نشان داده شده است:

ابتدا از پکیج PIL , Image را فراخوانده ام تا فرآیند خواندن عکس و ساختن مجدد عکس را انجام دهم و پکیج های دیگری که در این کد به کار رفته را وارد کردم.
سپس عکس مورد نظر را در pix ذخیره کردم.

```
from PIL import Image
import numpy as np
import math
import random
import sys

im = Image.open('segmentation.jpg')
pix = im.load()
```

در این قسمت کد پارامتر ها و آایه های مورد نظر را تعریف می کنیم.

Epoch که تعداد دور های اجرای برنامه را مشخص می کند.

summ مجموع مقادیر داخل یک خوشه را مشخص می کند.

num که تعداد پیکسل های درون یک خوشه را مشخص می کند.

Pic مشخص می کند هر پیکسل در کدام خوشه قرار دارد.

Subtract و distance که برای تعیین فاصله ی هر پیکسل یا خوشه ها به کار می رود.

```
epoch = 0
pic = np.zeros((300,450), dtype=int)
summ = np.zeros((7,3), dtype=int)
num = np.zeros((7,), dtype=int)
photo = np.zeros((300,450,3), dtype=np.uint8)
cluster = []
cluster1 = np.random.random_integers(0, 255, (1, 3))
cluster2 = np.random.random_integers(0, 255, (1, 3))
cluster3 = np.random.random_integers(0, 255, (1, 3))
cluster4 = np.random.random_integers(0, 255, (1, 3))
cluster5 = np.random.random_integers(0, 255, (1, 3))
cluster6 = np.random.random_integers(0, 255, (1, 3))
cluster7 = np.random.random_integers(0, 255, (1, 3))
cluster.append(cluster1)
cluster.append(cluster2)
cluster.append(cluster3)
cluster.append(cluster4)
cluster.append(cluster5)
cluster.append(cluster6)
cluster.append(cluster7)
subtract = np.zeros((3,1), dtype=int)
distance = np.zeros((7,))
```

در این قسمت کد به ازای تمام پیکسل ها فاصله ی هر پیکسل با مرکز خوشه ها را به دست می آوریم و خوشه ای که کمترین فاصله را داشته باشد به عنوان خوشه ی آن پیکشل انتخاب می شود و در آرایه ی pic ذخیره می شود.

```
for i in range (300):
    for j in range (450):
        for k in range (7):
            subtract =  np.subtract(pix[i,j], cluster[k])
            distance[k] = np.inner(subtract,subtract)
        d = distance.argmin()
        pic[i,j] = d
```

در این قسمت به ازای تمام درایه ی های آرایه ی pic مخصات رنگی آن پیکسل را summ آن مرکز خوشه اضافه می کنیم. (هر پیکسل چون در فضای RGB است به صورت یک بردار ۳تایی از میزان رنگ قرمز و سبز و آبی آن پیکسل مشخص می شود)

```
for i in range (300):
    for j in range (450):
        if pic [i,j] == 0 :
            summ[0] += pix[i,j]
            num[0] += 1
        if pic [i,j] == 1 :
            summ[1] += pix[i,j]
            num[1] += 1
```

مقدار جدید هر خوشه از تقسیم مقدار summ(summ به همان مفهموم sum است اما چون sum یک تابع در پایتون است از این نام استفاده کردم) آن خوشه به تعداد پیکسل های مرتبط با آن خوشه به دست می آید.(جز صحیح این مقدار)

```
for i in range (7):
    if num[i] != 0 :
        summ[i] = (summ[i]/num[i])
        summ[i] = np.floor(summ[i])
        cluster[i] = summ[i]
epoch += 1
```

در این قسمت آرایه photo به ازای تمام پیکسل ها شکل می گیرد و مقدار هر پیکس برابر با مقدار خوشه ی متناظر با آن تعیین می شود سپس این آرایه به صورت jpeg ذخیره می شود.

```
for i in range (300):
    for j in range (450):
        for k in range (7):
            if pic[i,j] == k:
                photo [i,j] = cluster[k]
img = Image.fromarray(photo)
img.save('myimg.jpeg')
```

نتیجه اجرای این الگوریتم با ۷ خوشه و به ترتیب از چپ به راست با ۱ بار ، ۳ بار و ۵ بار اجرا نشان داده شده:

Epoch = 1

Clusters = 7

Epoch = 3

Clusters = 7

Epoch = 5

Clusters = 7

حال این خوشه بندی را به ازای تعداد خوشه های کمتر اجرا می کنیم:

Epoch = 2

Clusters = 4

Epoch = 2

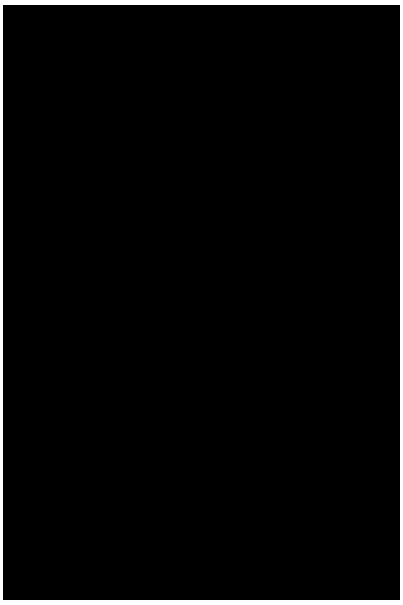
Clusters = 5

Epoch = 2

Clusters = 6

Epoch = 2

Clusters = 2



Epoch = 2

Clusters = 3

همانطور که ملاحظه می کنید بالا بردن تعداد خوشه ها لزوما جواب بهتری به ما نمی دهد (۵ خوشه و ۶ خوشه) و همچنین در هر عکس لزوما از تمامی خوشه ها استفاده نمی شود (عکس روبرو) زیرا خوشه ها در ابتدا به صورت رندوم انتخاب شده اند. ممکن است مقادیر این خوشه خیلی دور از داده ها باشد و اصلا استفاده نشود.