ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Project 01: Color Compression

Trần Quỳnh Như

18127266

Môn: Toán Ứng dụng và Thống kê

Mục lục

Ý tưởng thực hiện và Mô tả các hàm

Ý tưởng thực hiện:

- Biến ảnh ban đầu thành mảng 1 chiều gồm các pixels có 3 kênh màu RGB.
- Thuật toán K-Means:
  - Khởi tạo các means (trung tâm/trung bình cộng) của các clusters bằng cách chọn ngẫu nhiên k pixels bất kì.
  - Cho max_iter vòng lặp:
    - Tính khoảng cách từ tất cả các pixels đến tất cả các means.
    - Đối với mỗi pixel, chọn ra khoảng cách từ pixel đó đến mean gần nó nhất, sau đó gán nhãn cho pixel đó thuộc về cluster tương ứng.
    - Tính lại các means bằng cách tính trung bình cộng các pixels thuộc về các clusters của means đó.
- Thay các màu ban đầu của các pixels thành màu của các means tương ứng.
- Biến mảng 1 chiều các pixels trở lại thành ảnh có kích thước ban đầu.

Mô tả các hàm:

1. Đoạn code xử lí hình ảnh đầu vào

- Đọc ảnh bằng thư viện PIL/PIL.Image.
- Chuyển ảnh sang numpy array để thuận tiện cho việc xử lí và tính toán.
- Chia ảnh cho 255 (chia các kênh màu cho 255) để giảm độ lớn của các số liệu, giúp việc tính toán (tính khoảng cách, trung bình cộng) dễ dàng hơn.
- Dùng np.reshape() để biến bức ảnh thành array 1 chiều, mỗi phần tử trong array là 1 pixel có 3 kênh màu RGB.

```
[ ]  1 im = PIL.Image.open('image.png')
```

```
[ ]  1 im = np.array(im)
     2 im = im / 255
     3 pixels = np.reshape(im, (im.shape[0]*im.shape[1], 3))
```

2. Tính khoảng cách

- Tính khoảng cách giữa 2 pixels bằng cách sử dụng khoảng cách Manhattan.

```
[ ]  1 def distance(p1, p2):
     2
     3     # Manhattan distance
     4     dist = abs(p1[0] - p2[0]) + abs(p1[1] - p2[1]) + abs(p1[2] - p2[2])
     5     return dist
```

3. Khởi tạo means

- Dùng np.zeros() khởi tạo mảng means có độ dài bằng số lượng cluster, mỗi phần tử trong mảng là 1 pixel có 3 kênh màu RGB.
- Dùng np.random.random() để chọn ra k pixels làm means.

```
[ ]  1 def init_means(pixels, clusters):
     2
     3     # Initialize the list means
     4     means = np.zeros((clusters, 3))
     5
     6     # Randomly select k distinct pixels to become the 'means'
     7     # Each pixel has 3 color channels\n",
     8     for k in range(clusters):
     9
    10         rand_pixel = int(np.random.random(1)*pixels.shape[0])
    11         means[k] = pixels[rand_pixel]
    12
    13     return means
```

4. Thuật toán K-Means

- Khởi tạo list lables để dùng cho việc gán nhãn
- Cho max_iter vòng lặp
    - Khởi tạo nested list classes có độ dài bằng số lượng cluster.
        - Ý nghĩa của classes: classes[i] là một list gồm các pixels thuộc về cluster thứ i đó.

- Đối với mỗi pixel:
  - Tính khoảng cách giữa pixel đó và tất cả các clusters rồi lưu vào list distances.
  - Tìm min distance, sau đó lấy k = index của distance đó (chính là cluster tương ứng).
  - Nếu chưa phải vòng lặp cuối (max_iter – 1) thì thêm pixel đó vào classes[k], nếu là vòng lặp cuối cùng thì gán nhãn cho pixel đó luôn.
- Sau khi xét hết tất cả pixel, nếu chưa phải vòng lặp cuối thì tính lại giá trị các means bằng cách lấy trung bình cộng các pixels thuộc về cluster tương ứng với mean đó.

- Cuối cùng return list means nhân cho 255 (do ta đã chia 255 ở bước xử lí ảnh đầu vào) và np.array(labels) để tiện cho bước xử lí cuối cùng.

```python
def kmeans(pixels, means, clusters, max_iter):

    labels = []
    # Just an iterator
    for i in range(max_iter):

        classes = []
        for k in range(clusters):
            classes.append([])

        # Visit each pixel
        for pixel in pixels:

            # A list of distances btw the current pixel to the means
            distances = [distance(pixel, mean) for mean in means]
            # The cluster that has the min distance
            k = distances.index(min(distances))
            # The current pixel is now added
            # to the class number k
            if i < max_iter - 1:
                classes[k].append(pixel)
            else:
                # Label the pixel
                labels.append(k) # labels[i] = k <=> pixel i belongs to cluster k

        if i < max_iter - 1:
            for k in range(clusters):
                means[k] = np.average(classes[k], axis=0)

    return means*255, np.array(labels)
```

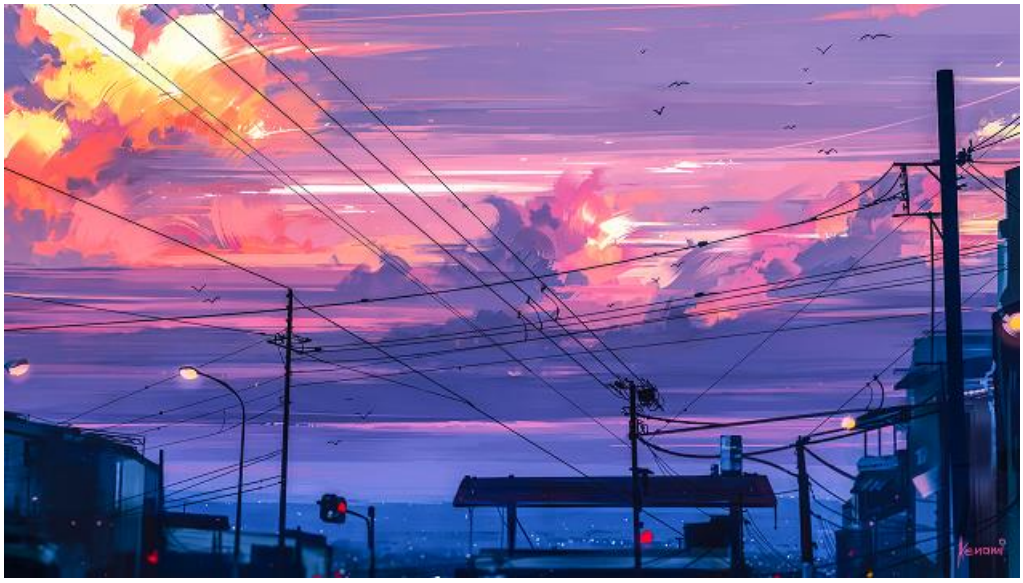
5. Nén màu ảnh

- Cho mảng centroid bằng np array của list means với các phần tử đều là số nguyên (int).
- Cho mảng recovered bằng centroid[labels].
    - Ý nghĩa: Gán pixel thứ i bằng giá trị centroid (màu) của labels thứ i tương ứng, rồi assign vào mảng recovered cũng là np.array().
    - Ví dụ: labels[0] là cluster mà pixel 0 thuộc về, nên recovered[0] = centroid[labels[0]] = màu tương ứng của pixel 0. Lặp lại các thao tác này cho đến pixel cuối cùng.
- Dùng np.reshape() để biến mảng 1 chiều thành mảng có kích thước của ảnh ban đầu.
- Dùng plt.imshow() để hiển thị ảnh.

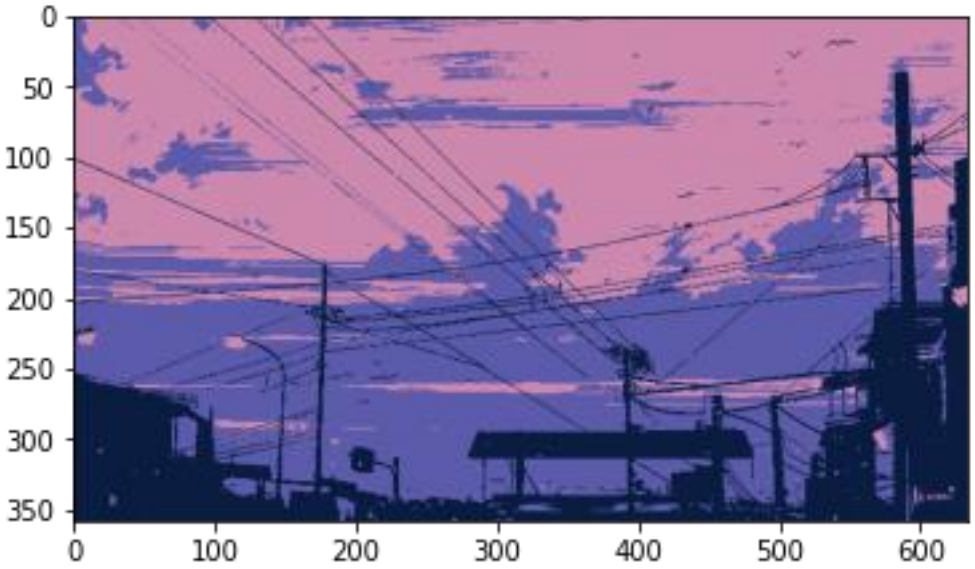
```
[16]  1 def compress_image(im, means, labels): # labels: list
      2
      3     # recovering the compressed image by
      4     # assigning each pixel to its corresponding centroid.
      5     centroid = np.array(means, dtype=int)
      6     recovered = centroid[labels]
      7
      8     # getting back the 3d matrix (row, col, rgb(3))
      9     recovered = np.reshape(recovered, (im.shape[0], im.shape[1], 3))
     10
     11     plt.imshow(np.array(recovered))
```

Hình ảnh kết quả

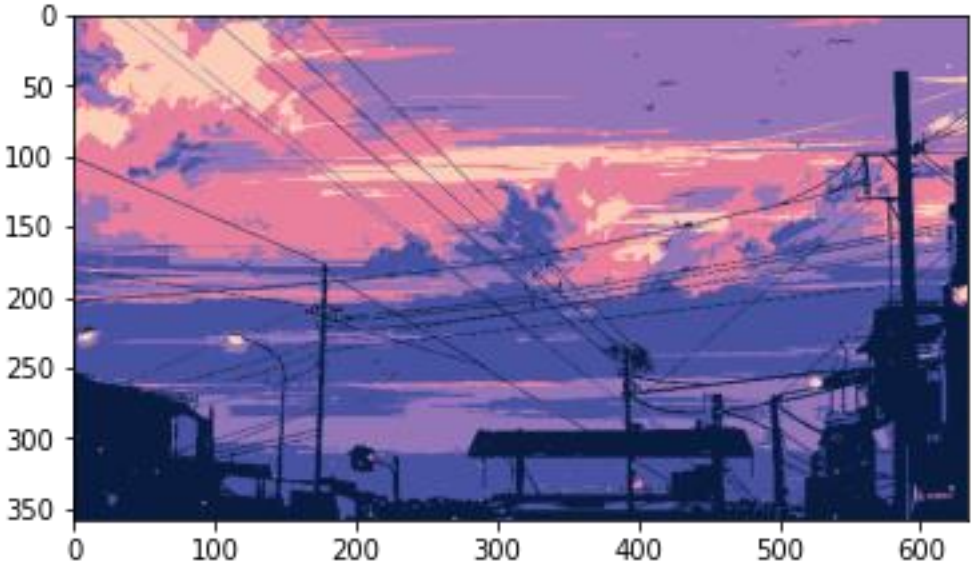
Ảnh gốc:



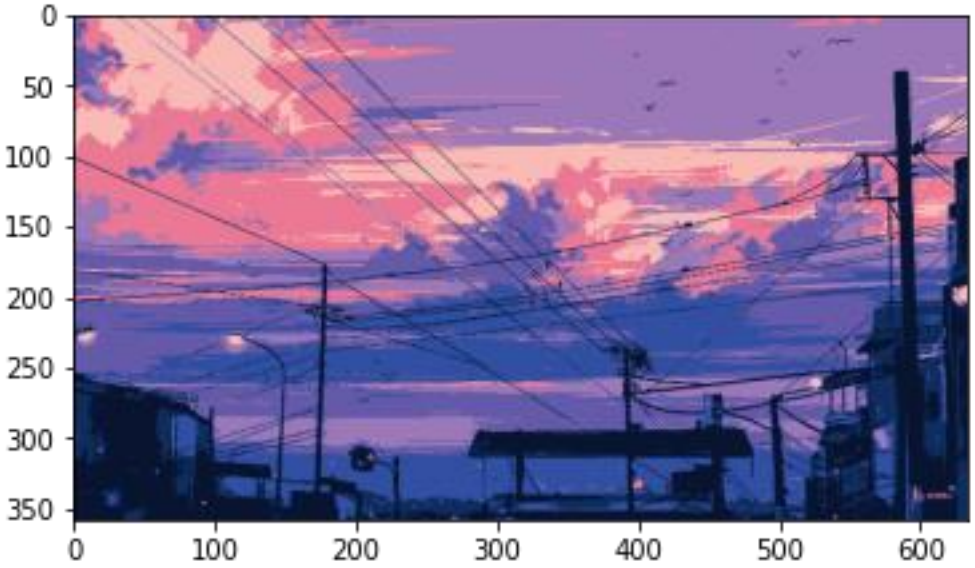
K = 3



K = 5


0
50
100
150
200
250
300
350
0
100
200
300
400
500
600

K = 7


0
50
100
150
200
250
300
350
0
100
200
300
400
500
600

Nhận xét kết quả trên

Với k = 3:

Ta vẫn có thể nhận diện, phân biệt được bầu trời, mây, nhà cửa, cột điện và dây điện. Bầu trời và mây thì mang màu hồng và tím, còn lại nhà cửa, cột điện, dây điện thì mang màu xanh đen.

Với k = 5:

Bầu trời và mây đã có thêm nhiều màu sắc hơn (vàng, cam hồng), những cột đèn thì ánh đèn giờ đã có màu khác nhưng nhà cửa, cột điện dây điện vẫn mang màu xanh đen.

K = 7:

Bầu trời có vẻ vẫn tương tự như như k = 5, nhưng điểm khác biệt lớn nhất là nhà cửa đã có thêm những mảng màu sáng tối khiến bức ảnh trở nên sắc nét hơn và ta thấy được nhiều chi tiết hơn.

References

Image compression using K-means clustering:
https://www.geeksforgeeks.org/image-compression-using-k-means-clustering/

Implementing K-Means clustering from Scratch-in Python:

http://madhugnadig.com/articles/machine-learning/2017/03/04/implementing-k-means-clustering-from-scratch-in-python.html