Project 1: Color Compression

20127479 – Lê Nhất Duy

MTH051 - FIT-HCMUS

1. Giới thiệu và yêu cầu:

Một bức ảnh có thể lưu trữ dưới ma trận của các điểm ảnh. Có nhiều loại ảnh được sử dụng trong thực tế, ví dụ: ảnh xám, ảnh màu,... Đối với ảnh xám, một điểm ảnh sẽ là được biểu diễn bằng giá trị không âm

Trong đồ án này, bạn được yêu cầu cài đặt chương trình giảm số lượng màu cho ảnh sử dụng thuật toán K-Means.

Các thư viện được phép sử dụng là: NumPy (tính toán ma trận), PIL (đọc, ghi ảnh), matplotlib (hiển thị ảnh).

2. Môi trường thực hiện:

Đồ án sử dụng Google Colab để chạy và test.

3. Mô tả các hàm và ý tưởng thực hiện:

Đồ án có khuôn hàm chính như sau:

```
def kmeans(img_1d, k_clusters, max_iter, init_centroids='random'):
  if check(k_clusters, max_iter, init_centroids) == False:
    return None

  # init 2 result
  labels = [None for _ in range(len(img_1d))]
  centroids, k_clusters = init_value_centroids(img_1d, init_centroids
, k_clusters)

  while max_iter:
    # Keep a previous labels version to compare
    labels_copy = labels

    clusters = [[] for _ in range(k_clusters)]

    # Travle all pixel
    for i in range(len(img_1d)):
      # get index has min value
     labels[i] = np.linalg.norm(centroids - img_1d[i], axis=1).argmin
()
      clusters[labels[i]].append(i)

    # Check labels = labels_copy ?
    if compare_label(labels, labels_copy) == True:
```

```
      break

    # Calculate mean and travel pixel again
    for i in range(k_clusters):
      if clusters[i]:
        centroids[i] = np.mean([img_1d[k] for k in clusters[i]])

    max_iter -= 1

    return centroids, labels
```

Đi kèm với hàm kmeans chính ta sẽ viết thêm một số hàm hỗ trợ để code dễ triển khai và dễ đọc hơn.

Ý tưởng chính: đầu tiên ta sẽ kiểm tra các tham số nhập vào có hợp lệ hay không (k_clusters, max_iter, init_centroids), nếu hợp lệ ta tiếp tục thực hiện.

```
# Init value of centroids
def init_value_centroids(img_1d, init_centroids, k_clusters):

  # Random case:
  if init_centroids == 'random':
    centroids = [np.random.randint(0, 255, 3) for _ in range(k_cluste
rs)]

    # Check reapeat element
    while len(np.unique(centroids, axis = 0)) != k_clusters:
      centroids = [np.random.randint(0, 255, 3) for _ in range(k_clus
ters)]
  # Pixel case:
  elif init_centroids == 'in_pixels':
      filter_pixel = np.unique(img_1d, axis=0)

      # Check k_cluster
      if k_clusters > len(filter_pixel):
        k_clusters = len(filter_pixel)

      # Random index in filter_pixel
      random_index = np.random.choice(len(filter_pixel), k_clusters,
replace = False)
      centroids = [filter_pixel[i] for i in random_index]

  return centroids, k_clusters
```

Tiếp đó ta khởi tạo giá trị cho centroids theo 2 trường hợp (random hoặc in_pixel), với mỗi trường hợp ta cần chú ý 2 điểm:

- Các giá trị của centroids phải khác nhau, nếu tồn tại 2 centroids giống nhau thì phải random lại từ đầu cho đến khi đạt mảng centroids hợp lí.
- Trong trường hợp in_pixel, cần phải kiểm tra xem k_clusters có lớn hơn số điểm ảnh không, nếu có thì phải gán lại k_clusters.

Ta khởi tạo mảng clusters có kích thước đúng bằng k_clusters, dùng để lưu index của các điểm ảnh, ví dụ điểm ảnh 0, 1, 2,.. của mảng ảnh img_1d thuộc centroids số 5 thì clusters[5] = [0, 1, 2,…].

```
# Compare |A - B| < 0
def compare_label(A, B):
  for i in range(len(A)):
    c = A[i] - B[i]
    if c != 0:
      return False
```

Bên cạnh đó ta copy label thành một mảng khác để thuận tiện cho việc kiểm tra điều kiện dừng của thuật toán (labels_copy != labels). Để kiểm tra labels ta viết một hàm ngoài như trên.

Sau khi kiểm tra labels xong, nếu hàm quyết định tiếp tục lặp thì ta cần tính toán lại giá trị trung bình của mỗi centroids và gán lại. Thuật toán tiếp tục thực hiện cho đến khi gặp điều kiện dừng hoặc max_iter về 0.

```
if __name__ == "__main__":
  k_clusters = [3, 5, 7]
  init_centroids = ['random', 'in_pixels']
  max_iter = 1000
  out_img = []

  # Enter path name
  img = input("Enter path name of picutre: ")

  # Add img
  raw_img = Image.open(img)
  raw_img = np.array(raw_img)
  out_img.append(raw_img)

  for i in range(len(init_centroids)):
    for j in range(len(k_clusters)):
      img_1d, img_height, img_width, num_channels = convert_image(img
)

      # Call kmean fucntion
      centroids, labels = kmeans(img_1d, k_clusters[j], max_iter, ini
t_centroids[i])
```

```
      # Compress
      img_1d = image_compress(img_1d, img_height, img_width, num_chan
nels, centroids, labels)

      # Add to arrray
      out_img.append(img_1d)

  # Show imgae
  show_img(out_img)

  type = input("Enter type of picture you want to save (png or pdf):
")

  if type != 'png' and type != 'pdf':
    print('Invalid type!')
  else:
    for i in range(len(out_img)):
      PIL.Image.fromarray(out_img[i], 'RGB').save(str(i) + '.' + type
)
```

Trên đây là hàm main: trước khi gọi hàm xử lý chính là kmeans, ta cần mở ảnh và chuyển ảnh thành mảng 1 chiều để dễ xử lý. Nên ta dùng một hàm riêng để chuyển đổi ảnh sang dạng mảng 1 chiều và trả lại kết quả.

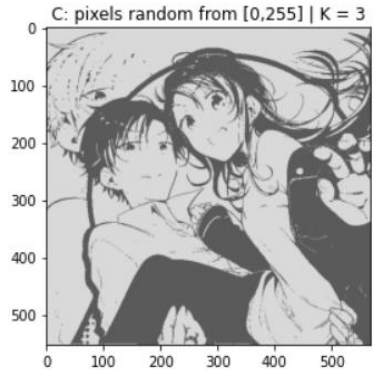
```
def convert_image(img):
  raw_img = Image.open(img)
  img = np.array(raw_img)
  img_height, img_width, num_channels = img.shape
  img = img.reshape(img_height * img_width, num_channels)

  return img, img_height, img_width, num_channels
```

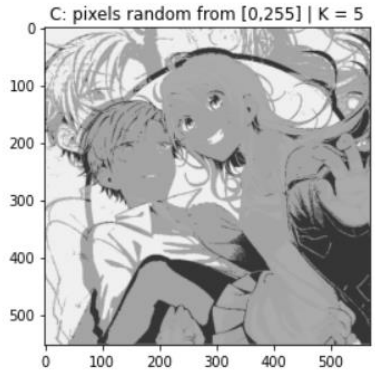
Và sau cùng khi chạy xong thuật toán thì ta cần gán lại các giá trị centroids cho điểm ảnh ban đầu. Đồng thời ta đưa mảng ảnh về mảng 3 chiều như ban đầu.

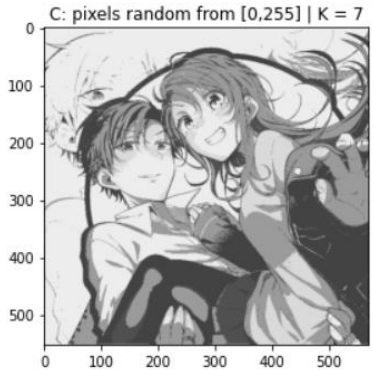
```
# Compress image
def image_compress(img_1d, img_height, img_width, num_channels, centr
oids, labels):
  for i in range(len(img_1d)):
    img_1d[i] = centroids[labels[i]]

  img_1d = img_1d.reshape(img_height, img_width, num_channels)
  return img_1d
```

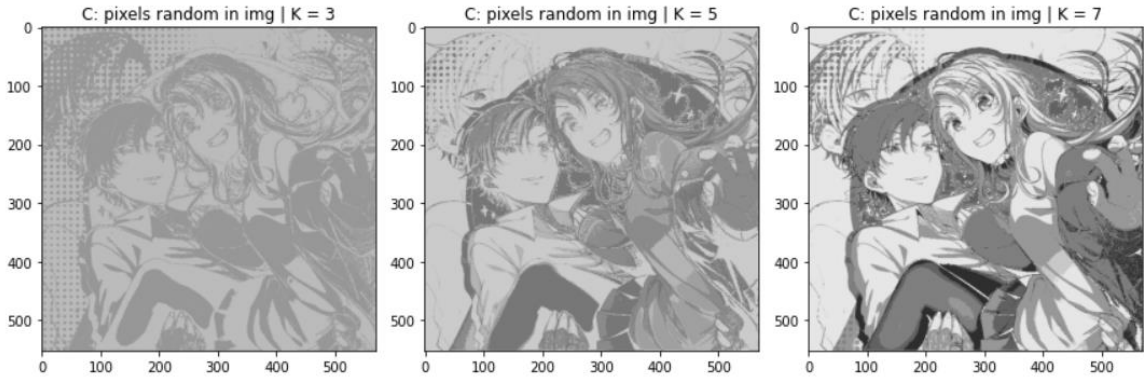
Cuối cùng người dùng có thể chọn kiểu đầu ra là jpg hay png hay pdf để xuất ảnh đã xử lý.

Trong file .ipynb tôi có viết một hàm để test mà không cần phải xuất file png hay pdf. Hàm test sẽ in ra ảnh đã xử lý ngay trên màn hình dể dễ kiểm tra và xem kết quả. Tôi đã test thử, và sau đây là kết quả: (cô có thể gọi hàm test và chạy nhanh để có thể nhìn nhanh kết quả mà không cần mở từng file đã xuất để kiểm tra)









Để in được hình như trên theo hàng đẹp nhất có thể, tôi có sử dụng một sô dòng code trên github (link tham khảo nằm ở cuối)

4. Tài liệu tham khảo:

- https://datahacker.rs/007-color-quantization-using-k-means-clustering/
- https://www.geeksforgeeks.org/image-compression-using-k-means-clustering/
- https://docs.python.org/3/library/random.html
- https://analyticsindiamag.com/beginners-guide-to-image-compression-using-k-means-clustering/
- https://shankarmsy.github.io/stories/clustering-sklearn.html#:~:text=K%20Means%20is%20an%20algorithm,alone%20(not%20the%20labels).
- https://numpy.org/doc/stable/reference/random/generated/numpy.random.rand.html
- https://numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html
- https://numpy.org/doc/stable/reference/generated/numpy.argmin.html?highlight=argmin#numpy.argmin
- https://github.com/t3bol90/ST-MA-Lab02