Họ tên : Nguyễn Quang Anh

Mã sinh viên : B21DCDT036

CHƯƠNG 9 : **Học máy Cổ điển trong Xử lý Ảnh**

Trong chương này, chúng ta sẽ thảo luận về việc áp dụng các kỹ thuật học máy trong xử lý ảnh. Chúng ta sẽ định nghĩa học máy và tìm hiểu về hai thuật toán của nó, gồm học có giám sát (supervised) và học không giám sát (unsupervised). Sau đó, chúng ta sẽ tiếp tục thảo luận về việc áp dụng một số kỹ thuật học máy không giám sát phổ biến, chẳng hạn như phân cụm (clustering), và các vấn đề như phân đoạn ảnh (image segmentation).

Chúng ta cũng sẽ tìm hiểu cách áp dụng các kỹ thuật học máy có giám sát cho các vấn đề như phân loại ảnh (image classification) và phát hiện đối tượng (object detection). Chương này sẽ sử dụng các thư viện phổ biến như scikit-learn, scikit-image và Python-OpenCV (cv2) để triển khai các thuật toán học máy trong xử lý ảnh. Nội dung sẽ giúp bạn hiểu sâu hơn về các thuật toán học máy và các vấn đề mà chúng giải quyết.

THUẬT TOÁN 1 : K-means clustering for image segmentation with color quantization

### 1, **Giới thiệu thuật toán**

**Color Quantization**:

* + Quá trình giảm số lượng màu trong ảnh bằng cách gom nhóm các pixel có màu sắc gần giống nhau.
  + Lợi ích: Giảm kích thước lưu trữ ảnh, tạo hiệu ứng nghệ thuật, hoặc đơn giản hóa dữ liệu cho các ứng dụng khác.
  + Ảnh đầu vào thường có hàng nghìn màu khác nhau (mỗi pixel là một giá trị RGB). Thuật toán giảm xuống một số lượng nhỏ các màu (như 4, 16, 64) mà vẫn giữ được chất lượng tổng thể.

**K-Means Clustering**:

* + Thuật toán gom nhóm các điểm dữ liệu vào **k cụm**.
  + Với ảnh, mỗi pixel là một điểm trong không gian 3D (RGB).
  + Mục tiêu: Tìm **k cụm màu đại diện** (màu trung tâm của các cụm - "codebook").

**So sánh với Random Codebook**:

* + Chọn ngẫu nhiên các màu làm đại diện cụm.
  + Đơn giản nhưng thường kém chính xác hơn K-Means.

### 2,**Các bước thực hiện thuật toán**

1. Đọc ảnh và chuyển đổi sang định dạng ma trận RGB.
2. Sử dụng K-Means để phân cụm màu của các pixel.
3. Tái tạo ảnh bằng cách gán mỗi pixel thành màu của cụm mà nó thuộc về.
4. So sánh với kết quả khi dùng Random Codebook.

**3,Giải thích đoạn mã nguồn**

**. Import các thư viện**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.metrics import pairwise\_distances\_argmin

from skimage.io import imread

from sklearn.utils import shuffle

from skimage import img\_as\_float

from time import time

* **numpy**: Xử lý mảng và các phép tính ma trận.
* **matplotlib.pyplot**: Hiển thị ảnh.
* **KMeans**: Thuật toán K-Means từ sklearn.
* **pairwise\_distances\_argmin**: Gán nhãn cho các điểm dữ liệu với codebook gần nhất (dùng cho Random Codebook).
* **imread**: Đọc ảnh.
* **shuffle**: Lấy mẫu ngẫu nhiên để tăng tốc độ xử lý.
* **img\_as\_float**: Chuyển đổi ảnh sang số thực.
* **time**: Đo thời gian thực thi.

**Đọc và hiển thị ảnh gốc**

pepper = imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\pepper.jpg")

plt.figure(1)

plt.axis('off')

plt.title('Original image (%d colors)' % len(np.unique(pepper)))

plt.imshow(pepper)

* Đọc ảnh **pepper** từ file.
* **np.unique(pepper)**: Đếm số màu gốc của ảnh.
* Hiển thị ảnh với tiêu đề số lượng màu gốc.

**. Chuẩn bị dữ liệu**pepper = np.array(pepper, dtype=np.float64) / 255

w, h, d = original\_shape = tuple(pepper.shape)

image\_array = np.reshape(pepper, (w \* h, d))

* **Chuyển đổi dữ liệu ảnh**:
  + Chuyển ảnh sang kiểu float64 và chuẩn hóa về khoảng [0, 1].
* **Kích thước ảnh**:
  + w, h, d: Chiều rộng, chiều cao và số kênh (3 kênh: R, G, B).
  + **image\_array**: Chuyển ảnh thành ma trận dạng (số pixel, 3) để xử lý K-Means.

**Hàm tái tạo ảnh**

def recreate\_image(codebook, labels, w, h):

image = np.zeros((w, h, codebook.shape[1]))

label\_idx = 0

for i in range(w):

for j in range(h):

image[i][j] = codebook[labels[label\_idx]]

label\_idx += 1

return image

* **Input**:
  + **codebook**: Trung tâm các cụm (k màu).
  + **labels**: Nhãn của mỗi pixel (gán vào một trong k cụm).
  + **w, h**: Kích thước ảnh.
* **Output**:
  + Ảnh tái tạo với mỗi pixel được gán vào màu trung tâm của cụm.

**. Áp dụng K-Means**

for k in [64, 32, 16, 4]:

image\_array\_sample = shuffle(image\_array, random\_state=0)[:1000]

kmeans = KMeans(n\_clusters=k, random\_state=0).fit(image\_array\_sample)

labels = kmeans.predict(image\_array)

plt.imshow(recreate\_image(kmeans.cluster\_centers\_, labels, w, h))

* **Số cụm màu**: Giảm số màu từ 64, 32, 16, đến 4.
* **Lấy mẫu ngẫu nhiên**:
  + **shuffle(image\_array, random\_state=0)[:1000]**: Lấy 1000 pixel để tăng tốc độ huấn luyện.
* **Huấn luyện K-Means**:
  + **fit()**: Tìm trung tâm cụm.
  + **predict()**: Gán nhãn cụm cho toàn bộ ảnh.
* **Tái tạo ảnh**:
  + Sử dụng các trung tâm cụm để tái tạo ảnh giảm màu.

**. Áp dụng Random Codebook**

for k in [64, 32, 16, 4]:

codebook\_random = shuffle(image\_array, random\_state=0)[:k + 1]

labels\_random = pairwise\_distances\_argmin(codebook\_random, image\_array, axis=0)

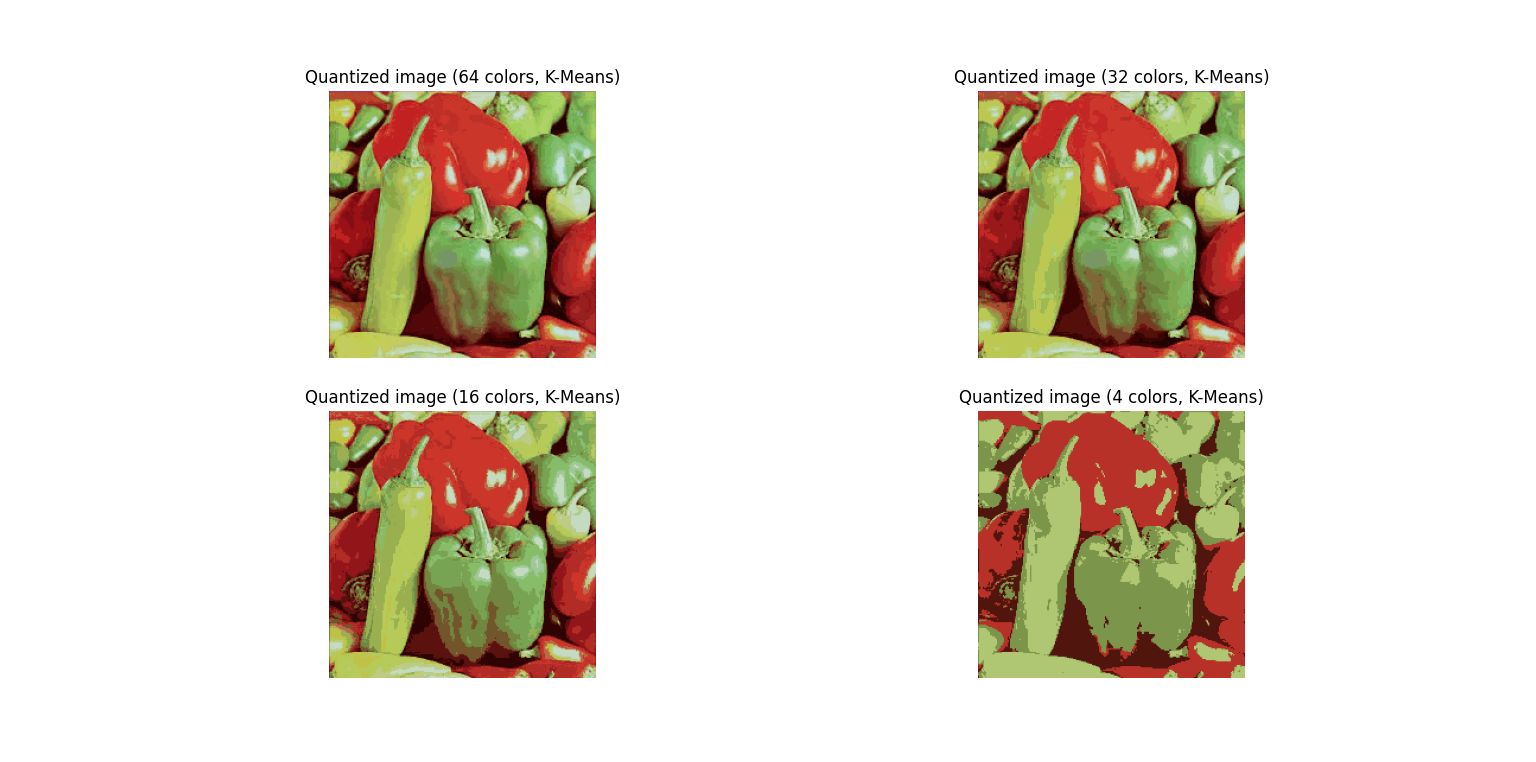
plt.imshow(recreate\_image(codebook\_random, labels\_random, w, h))

* **Codebook ngẫu nhiên**:
  + Chọn ngẫu nhiên **k** màu từ các pixel.
* **Gán nhãn**:
  + **pairwise\_distances\_argmin()**: Tính khoảng cách từ mỗi pixel đến các màu trong codebook và chọn màu gần nhất.

4,Kết quả thực hiện :

Ví dụ 1 : Ảnh gốc :

Ảnh sau khi thực hiện :



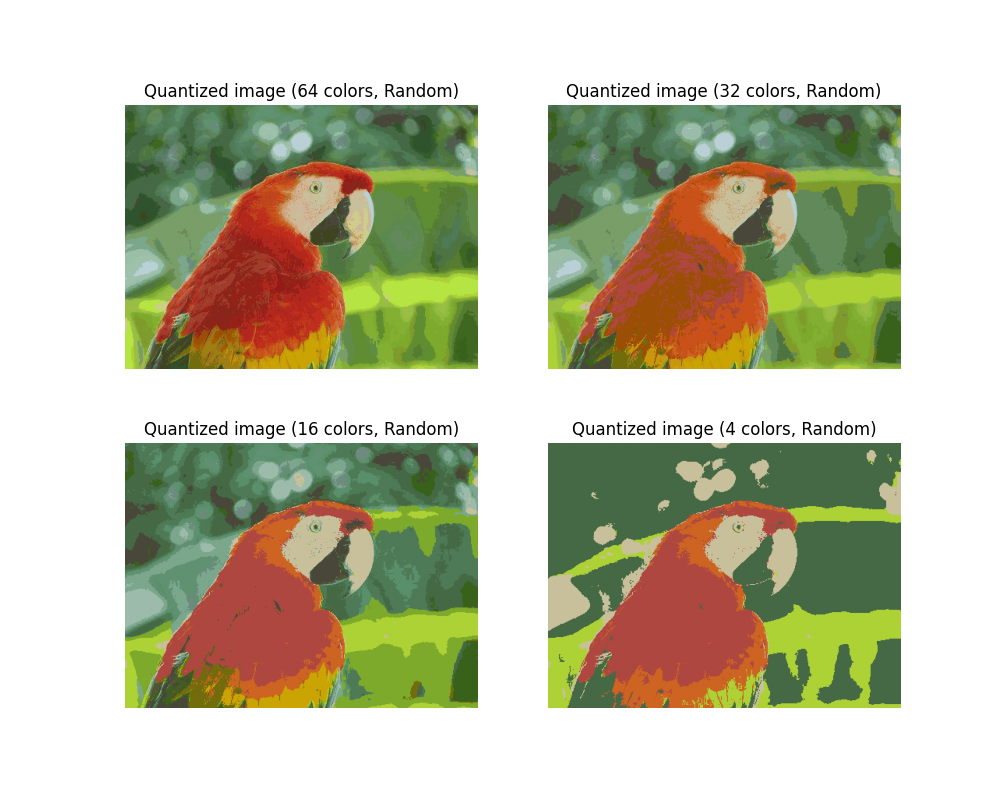


Ví dụ 2 : Ảnh gốc

: 

Ảnh sau khi thực hiện :





THUẬT TOÁN 2 : Spectral clustering for image segmentation

**1,Giới thiệu thuật toán**

**Spectral Clustering**

* **Ý tưởng chính**:
  + Coi ảnh là một đồ thị, mỗi pixel là một nút và trọng số cạnh giữa các nút thể hiện sự tương đồng giữa pixel.
  + Phân đoạn ảnh là tìm các nhóm nút có mối quan hệ mật thiết (các cụm pixel tương tự nhau).
  + Sử dụng phương pháp **Normalized Cuts** để chọn đường cắt đồ thị tối ưu, giảm thiểu độ tương phản giữa các cụm và cân bằng kích thước các cụm.
* **Lợi ích**:
  + Hoạt động tốt trên dữ liệu không tuyến tính hoặc có hình dạng phức tạp.
  + Không yêu cầu các cụm có dạng cầu.

**K-Means Clustering**

* Phân đoạn dựa trên khoảng cách Euclidean giữa pixel trong không gian màu.
* Mỗi pixel được gán vào cụm gần nhất dựa trên tâm cụm.
* Hiệu quả cao nhưng không tốt với dữ liệu phức tạp hoặc phi tuyến tính.

**So sánh**

* **K-Means**: Nhanh nhưng bị hạn chế bởi giả định cụm có dạng cầu.
* **Spectral Clustering**: Chậm hơn nhưng phù hợp với cấu trúc cụm phức tạp.

### **2.Các bước thực hiện thuật toán :**

**B1: Chuẩn bị dữ liệu**

* + Đọc và thay đổi kích thước ảnh.
  + Chuyển ảnh thành dạng ma trận để áp dụng thuật toán.

**B2: Phân đoạn với K-Means**

* + Gom nhóm pixel vào 2 cụm.
  + Hiển thị ảnh phân đoạn và các đường viền cụm.

**B3: Phân đoạn với Spectral Clustering**

* + Dùng đồ thị pixel và các giá trị riêng để chia ảnh thành 2 cụm.
  + Hiển thị ảnh phân đoạn và các đường viền cụm.

**B4: So sánh kết quả**

**3,Giải thích đoạn mã nguồn**

**. Import thư viện**

from sklearn import cluster

from skimage.io import imread

from skimage.color import rgb2gray

from skimage.transform import resize

import numpy as np

import matplotlib.pylab as pylab

* **cluster**: Cung cấp các thuật toán phân cụm như K-Means và Spectral Clustering.
* **imread**: Đọc ảnh từ file.
* **rgb2gray**: Chuyển đổi ảnh màu thành ảnh xám.
* **resize**: Thay đổi kích thước ảnh.
* **numpy**: Xử lý dữ liệu ma trận.
* **matplotlib.pylab**: Hiển thị ảnh.

**. Chuẩn bị dữ liệu**

im = resize(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\me14.jpg"),

(100, 100, 3), anti\_aliasing=True)

img = rgb2gray(im)

* **imread**: Đọc ảnh từ đường dẫn.
* **resize**: Giảm kích thước ảnh xuống 100×100×3 (3 kênh màu RGB) để giảm độ phức tạp tính toán.
* **rgb2gray**: Chuyển ảnh màu thành ảnh xám (nếu cần sử dụng ảnh grayscale).

**Phân đoạn bằng K-Means**

X = np.reshape(im, (-1, im.shape[-1]))

two\_means = cluster.MiniBatchKMeans(n\_clusters=k, random\_state=10)

two\_means.fit(X)

y\_pred = two\_means.predict(X)

labels = np.reshape(y\_pred, im.shape[:2])

* **Chuyển đổi ảnh**:
  + **np.reshape(im, (-1, im.shape[-1]))**: Chuyển ảnh thành ma trận 2D với mỗi hàng là một pixel (R, G, B).
* **K-Means**:
  + **MiniBatchKMeans(n\_clusters=k)**: Áp dụng K-Means với 2 cụm.
  + **fit()**: Huấn luyện mô hình trên dữ liệu ảnh.
  + **predict()**: Gán mỗi pixel vào một cụm (0 hoặc 1).
* **Tái tạo nhãn cụm**:
  + **np.reshape(y\_pred, im.shape[:2])**: Chuyển nhãn cụm về dạng ma trận 2D khớp với kích thước ảnh gốc.

**Phân đoạn bằng Spectral Clustering**

spectral = cluster.SpectralClustering(n\_clusters=k, eigen\_solver='arpack',

affinity="nearest\_neighbors", n\_neighbors=10, random\_state=10)

spectral.fit(X)

y\_pred = spectral.labels\_.astype(int)

labels = np.reshape(y\_pred, im.shape[:2])

* **Spectral Clustering**:
  + **n\_clusters=k**: Số cụm là 2.
  + **affinity="nearest\_neighbors"**: Sử dụng đồ thị kết nối láng giềng gần nhất.
  + **n\_neighbors=10**: Mỗi pixel kết nối với 10 pixel gần nhất.
  + **fit()**: Huấn luyện mô hình trên dữ liệu ảnh.
* **Tái tạo nhãn cụm**:
  + **spectral.labels\_**: Nhãn cụm sau khi phân đoạn.
  + **np.reshape(y\_pred, im.shape[:2])**: Chuyển nhãn cụm về dạng ma trận.

**. Hiển thị kết quả**

pylab.subplot(221)

pylab.imshow(np.reshape(y\_pred, im.shape[:2]))

pylab.title('k-means segmentation (k=2)', size=30)

pylab.subplot(222)

pylab.imshow(im)

pylab.contour(labels == 0, levels=[0], colors='red')

* **Hiển thị kết quả K-Means**:
  + Ảnh phân đoạn và đường viền cụm.

pylab.subplot(223)

pylab.imshow(np.reshape(y\_pred, im.shape[:2]))

pylab.title('spectral segmentation (k=2)', size=30)

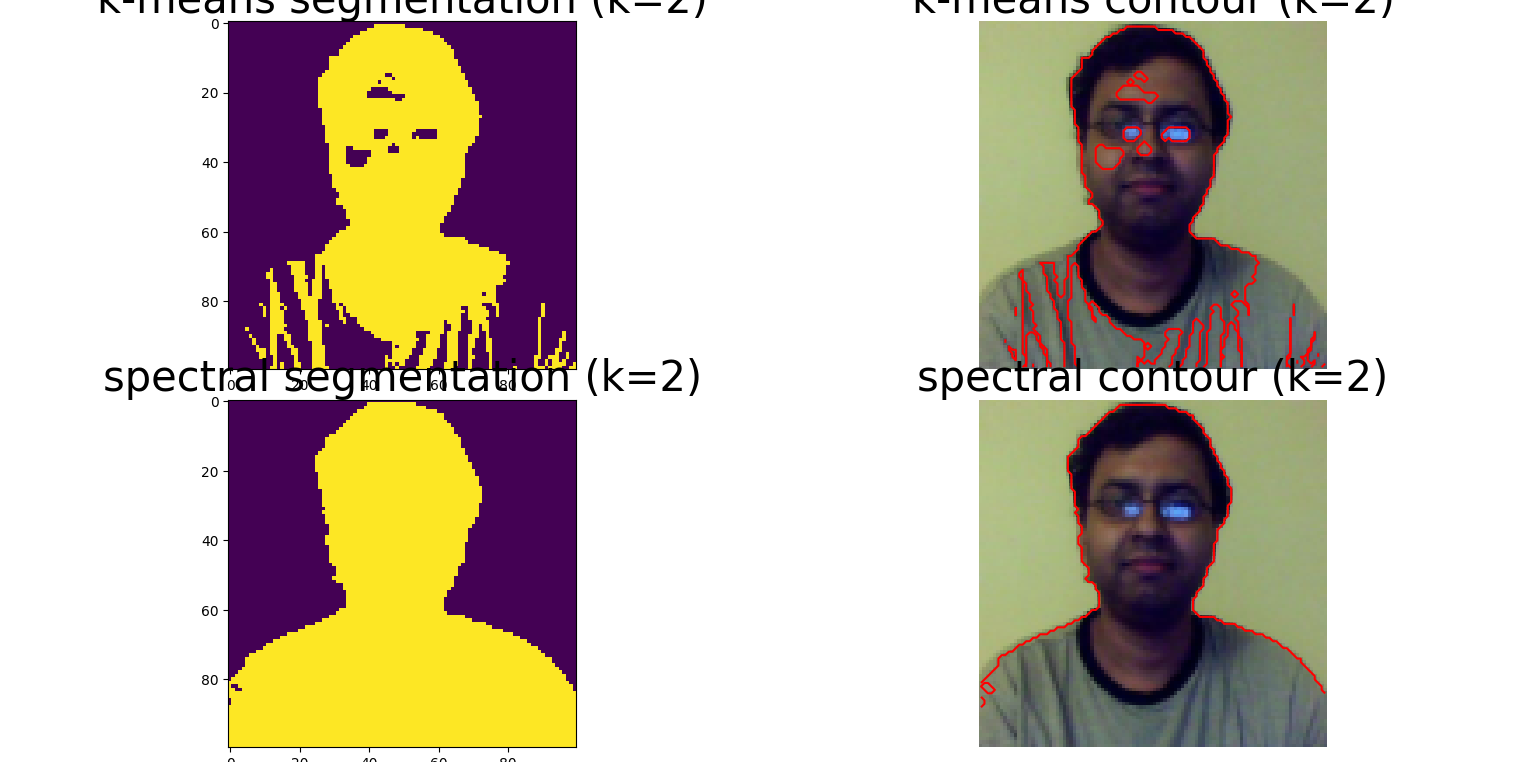
pylab.subplot(224)

pylab.imshow(im)

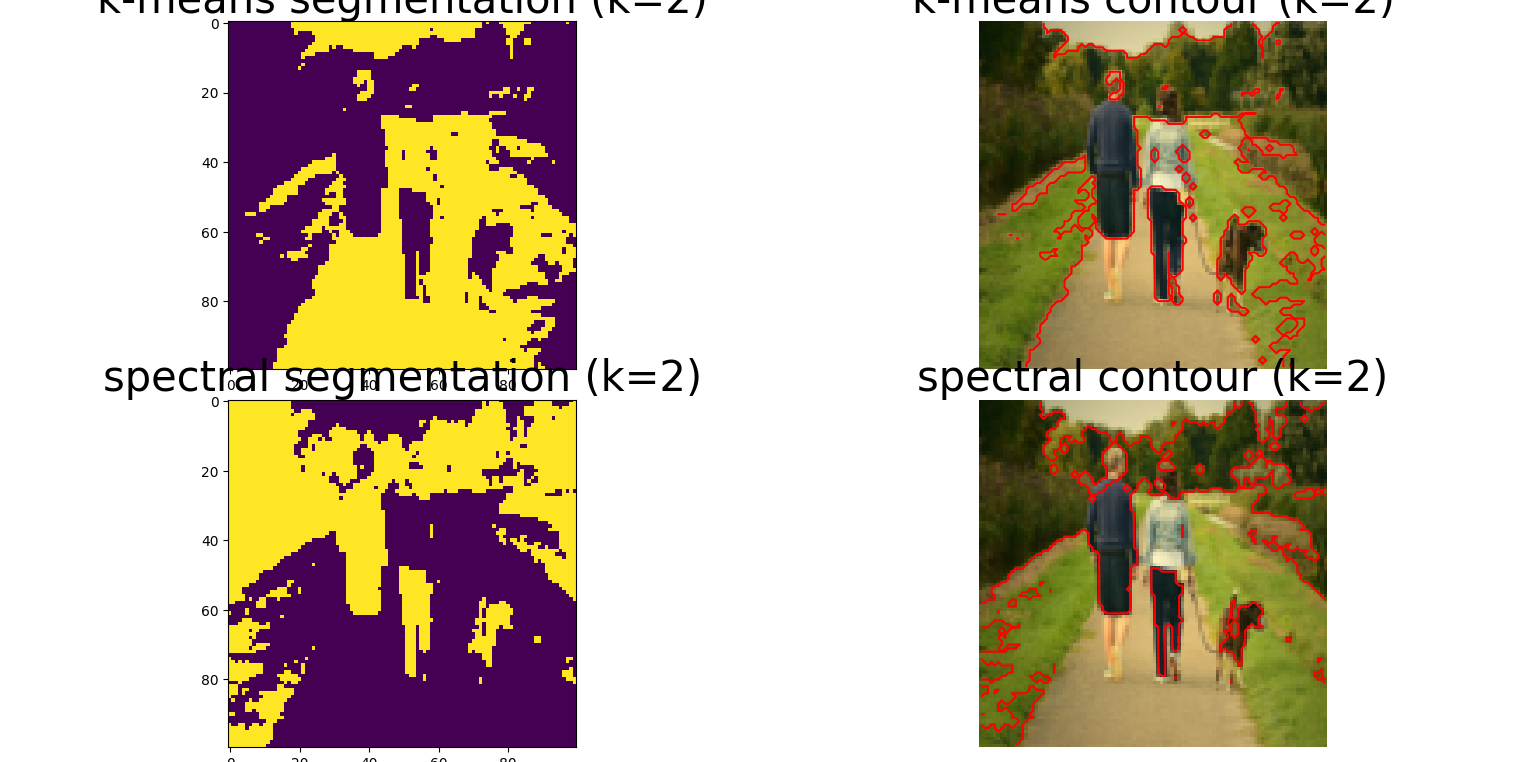
pylab.contour(labels == 0, levels=[0], colors='red')

* **Hiển thị kết quả Spectral Clustering**:
  + Ảnh phân đoạn và đường viền cụm.

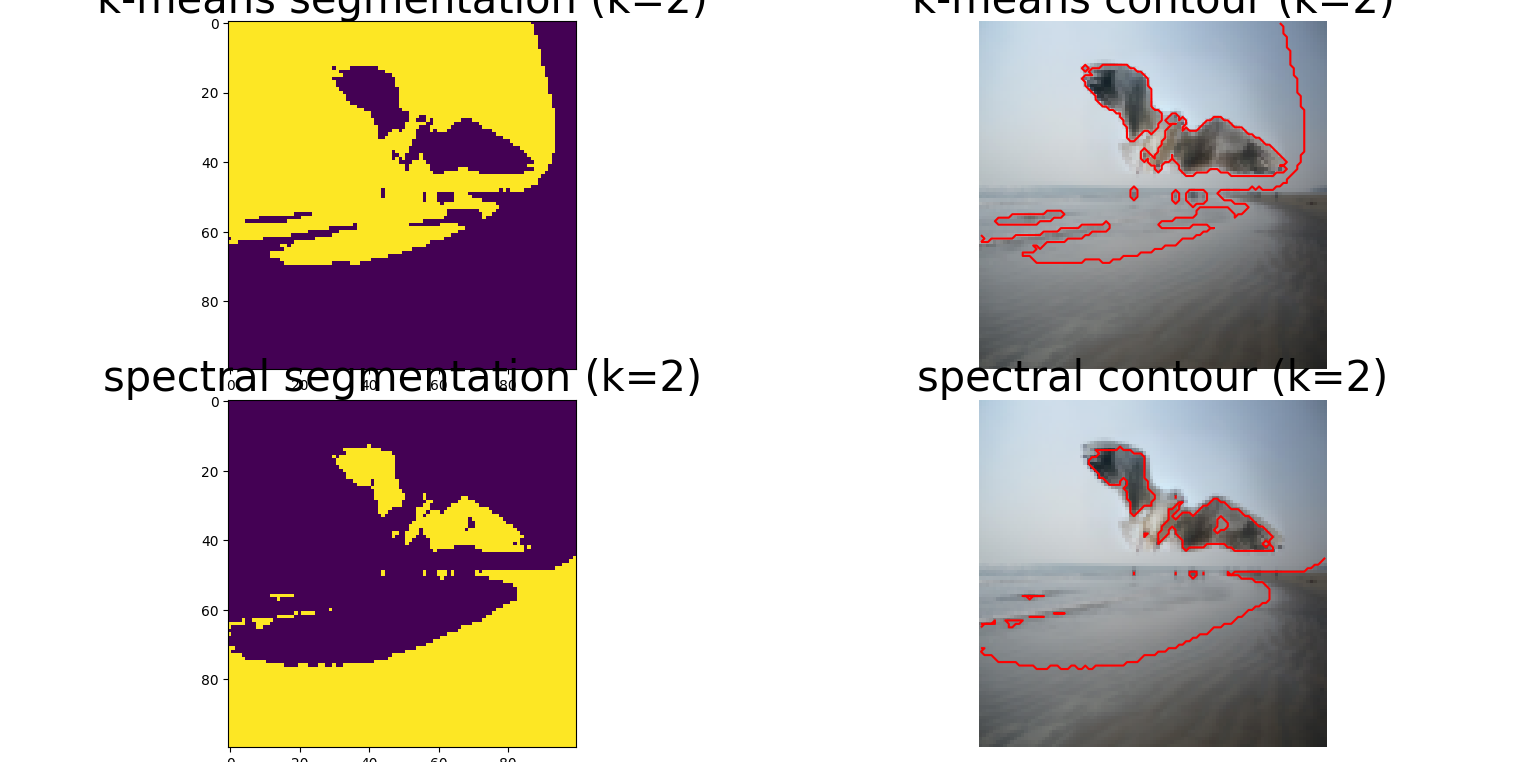
**4,Kết quả thực hiện :**

Ví dụ 1 :

Ví dụ 2 :



Ví dụ 3 :



THUẬT TOÁN 3 : Face classification using the Haar-like feature descriptor

1,Giới thiệu thuật toán :

**Haar-like Features**

* **Ý tưởng chính**:
  + **Haar-like features** là các đặc trưng dựa trên sự tương phản ánh sáng giữa các vùng của ảnh, ví dụ:
    - Một vùng sáng cạnh một vùng tối.
    - Một vùng sáng bao quanh bởi vùng tối.
  + Những đặc trưng này giúp nhận diện khuôn mặt vì các bộ phận trên mặt (mắt, mũi, miệng) thường có sự tương phản đặc trưng.
* **Tính toán nhanh**:
  + Sử dụng **Integral Image** để tính toán tổng độ sáng của một vùng trong thời gian hằng số.

**Viola-Jones Framework**

* **Các bước chính**:
  1. Trích xuất **Haar-like features** từ ảnh.
  2. Chọn ra các đặc trưng quan trọng nhất thông qua các thuật toán học máy.
  3. Dùng thuật toán phân loại (như AdaBoost hoặc Random Forest) để xác định khuôn mặt.

**Random Forest Classifier**

* **Lý do sử dụng**:
  + Random Forest chọn ra các đặc trưng quan trọng nhất từ một tập lớn các đặc trưng Haar-like.
  + Đây là một thuật toán học máy mạnh mẽ, ổn định với dữ liệu không đồng nhất.

2,Câc bước thực hiện thuật toán :

B1: **Tải tập dữ liệu**:

* Sử dụng tập LFW (Labeled Faces in the Wild), gồm các ảnh khuôn mặt và không khuôn mặt.
* Mỗi ảnh được chuyển đổi kích thước về 19×19 pixel.

B2: **Trích xuất Haar-like features**:

* Xác định vùng ROI từ mỗi ảnh.
* Tính **Integral Image** của ROI để tăng tốc tính toán.
* Trích xuất các đặc trưng Haar-like từ Integral Image.

B3: **Chia tập dữ liệu**:

* Chia dữ liệu thành 75% huấn luyện và 25% kiểm tra.

B4: **Phân loại**:

* Huấn luyện mô hình Random Forest trên tập huấn luyện.
* Sử dụng tập kiểm tra để đánh giá hiệu năng mô hình (ROC AUC score).

4, Giải thích mã nguồn

from time import time  
import numpy as np  
import matplotlib.pyplot as pylab  
from dask import delayed  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import roc\_auc\_score  
from skimage.data import lfw\_subset  
from skimage.transform import integral\_image  
from skimage.feature import haar\_like\_feature  
from skimage.feature import haar\_like\_feature\_coord  
from skimage.feature import draw\_haar\_like\_feature  
  
  
@delayed  
def extract\_feature\_image(img, feature\_type, feature\_coord=None):  
 *"""Extract the haar feature for the current image"""* ii = integral\_image(img)  
 return haar\_like\_feature(ii, 0, 0, ii.shape[0], ii.shape[1],  
 feature\_type=feature\_type,  
 feature\_coord=feature\_coord)  
  
  
# Load dataset  
images = lfw\_subset()  
print(images.shape)  
# (200, 25, 25)  
  
# Display faces in a grid  
fig = pylab.figure(figsize=(5, 5))  
fig.subplots\_adjust(left=0, right=0.9, bottom=0, top=0.9, hspace=0.05,  
 wspace=0.05)  
for i in range(25):  
 pylab.subplot(5, 5, i + 1), pylab.imshow(images[75+i, :, :], cmap='bone'),  
pylab.axis('off')  
pylab.suptitle('Faces')  
pylab.show()

**Import thư viện**

from time import time

import numpy as np

import matplotlib.pyplot as pylab

from dask import delayed

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import roc\_auc\_score

from skimage.data import lfw\_subset

from skimage.transform import integral\_image

from skimage.feature import haar\_like\_feature

from skimage.feature import haar\_like\_feature\_coord

from skimage.feature import draw\_haar\_like\_feature

* **dask.delayed**: Dùng để tối ưu hóa quá trình tính toán song song.
* **RandomForestClassifier**: Thuật toán phân loại ngẫu nhiên sử dụng cây quyết định.
* **lfw\_subset**: Lấy tập con của tập dữ liệu LFW (gồm ảnh khuôn mặt và không khuôn mặt).
* **integral\_image**: Tính Integral Image để tăng tốc tính toán Haar-like features.
* **haar\_like\_feature**: Trích xuất các đặc trưng Haar-like.

**Hàm trích xuất đặc trưng Haar-like**

@delayed

def extract\_feature\_image(img, feature\_type, feature\_coord=None):

"""Extract the haar feature for the current image"""

ii = integral\_image(img)

return haar\_like\_feature(ii, 0, 0, ii.shape[0], ii.shape[1],

feature\_type=feature\_type,

feature\_coord=feature\_coord)

* **Hàm extract\_feature\_image**:
  + Input:
    - **img**: Ảnh đầu vào.
    - **feature\_type**: Loại đặc trưng Haar-like cần trích xuất (ví dụ: cạnh, đường chéo, ...)
  + Output: Các đặc trưng Haar-like.
* **Integral Image**:
  + Tăng tốc tính toán tổng độ sáng của bất kỳ vùng hình chữ nhật nào trong ảnh.

**Tải và hiển thị dữ liệu**

images = lfw\_subset()

print(images.shape) # (200, 25, 25)

# Display faces in a grid

fig = pylab.figure(figsize=(5, 5))

fig.subplots\_adjust(left=0, right=0.9, bottom=0, top=0.9, hspace=0.05, wspace=0.05)

for i in range(25):

pylab.subplot(5, 5, i + 1), pylab.imshow(images[75+i, :, :], cmap='bone'),

pylab.axis('off')

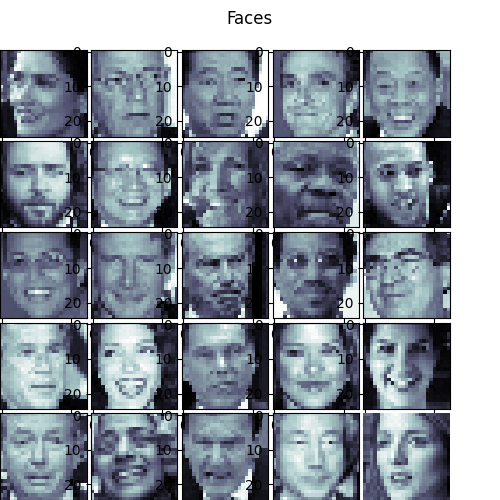
pylab.suptitle('Faces')

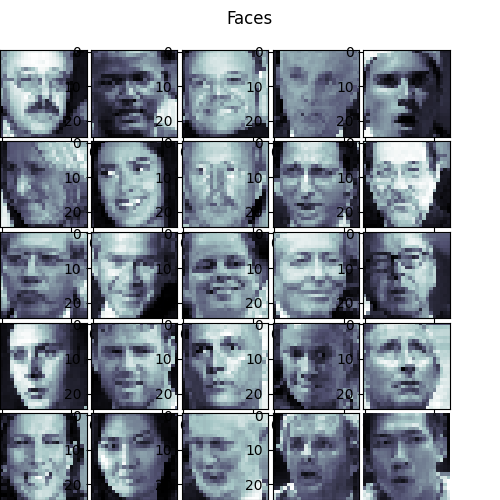
pylab.show()

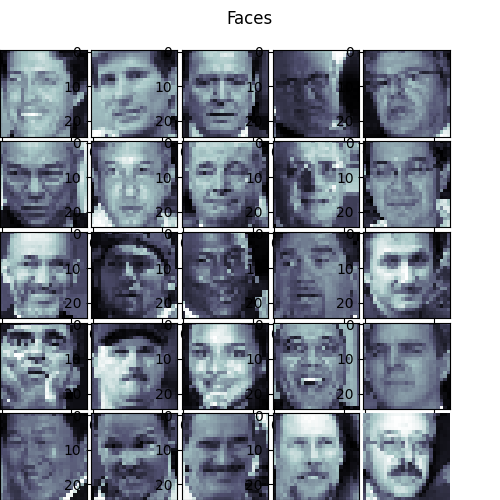
* **lfw\_subset**: Lấy tập dữ liệu gồm 200 ảnh (100 khuôn mặt, 100 không phải khuôn mặt).
* **pylab.figure**:
  + Hiển thị 25 ảnh từ tập dữ liệu, sắp xếp trong lưới 5×5.
* **images[75+i]**:
  + Lấy ảnh từ 75 đến 99, đây là các ảnh khuôn mặt.

4 , Kết quả thực hiện :

Ví dụ 1 :



Ví dụ 2 : 

Ví dụ 3 : 

**Nhận xét ảnh đầu ra**

* **Hình ảnh hiển thị**: Lưới gồm 25 ảnh khuôn mặt từ tập LFW.
* **Quan sát**:
  + Các khuôn mặt được hiển thị rõ ràng, có kích thước nhỏ (25×25 pixel).
  + Kích thước nhỏ giúp giảm độ phức tạp tính toán nhưng vẫn giữ lại đặc trưng quan trọng.

Tóm tắt

Ưu điểm:

Sử dụng Haar-like features và Integral Image giúp tăng tốc đáng kể so với cách tính tổng pixel thông thường.

Random Forest dễ triển khai và hiệu quả trong việc chọn đặc trưng quan trọng.

Hạn chế:

Kích thước nhỏ của ảnh có thể làm giảm độ chính xác trên tập dữ liệu thực tế phức tạp hơn.

Haar-like features hiệu quả với khuôn mặt cơ bản nhưng có thể gặp khó khăn với biến dạng hoặc chi tiết phức tạp.