Trí tuệ nhân tạo

Ứng dụng Gaussian Mixture Model phát hiện chuyển động trong video

Nhóm 3

Ngày 22 tháng 4 năm 2024

Sinh viên thực hiện
106200284 - Hồ Đức Vũ - 20KTMT2
106200241 - Nguyễn Minh Phương - 20KTMT1
106200240 - Huỳnh Vũ Đình Phước - 20KTMT1

Giáo viên hướng dẫn
TS. Hoàng Lê Uyên Thục

Môn học
Trí tuệ nhân tạo

Đề tài
Ứng dụng Gaussian Mixture Model phát hiện chuyển động trong video

Xuất bản
Đà Nẵng, Ngày 22 tháng 4 năm 2024

Số trang
12

Mục lục

Nội dung báo cáo

1 Giới thiệu Gaussian Mixture Model

Gaussian Mixture Model (viết tắt GMM) là một mô hình phân cụm thuộc lớp bài toán học không giám sát mà phân phối xác suất của mỗi một cụm được giả định là phân phối Gassian đa chiều. Sở dĩ mô hình được gọi là Mixture là vì xác suất của mỗi điểm dữ liệu không chỉ phụ thuộc vào một phân phối Gaussian duy nhất mà là kết hợp từ nhiều phân phối Gaussian khác nhau từ mỗi cụm.

GMM được ứng dụng rộng rãi trong nhiều lĩnh vực của machine learning, đặc biệt là trong các tác vụ liên quan đến việc hiểu và phân tích cấu trúc của dữ liệu không gian nhiều chiều. Trong phân loại dữ liệu, GMM được sử dụng để xác định ranh giới giữa các lớp khác nhau trong dữ liệu, giúp cải thiện độ chính xác của việc phân loại. Trong phân cụm dữ liệu, GMM cho phép xác định số lượng và hình dạng của các cụm một cách linh hoạt, không giống như một số phương pháp phân cụm khác như K-means, vốn giả định rằng tất cả các cụm đều có hình dạng cầu và kích thước tương tự nhau. GMM cũng được sử dụng trong giảm chiều dữ liệu, giúp lọc nhiễu và trích xuất các đặc trưng quan trọng từ dữ liệu, qua đó cung cấp cái nhìn sâu sắc và giúp đơn giản hóa các mô hình học máy phức tạp hơn.

Một GMM bao gồm nhiều thành phần cơ bản:

- Các phân phối thành phần (Component Distributions): Mỗi phân phối Gaussian trong GMM được gọi là một thành phần. Mỗi thành phần mô hình hóa một cụm hoặc nhóm dữ liệu trong tổng thể dữ liệu.
- Trung bình (Means): Trung bình μ của mỗi phân phối thành phần cho biết vị trí trung tâm của cụm dữ liệu đó trong không gian nhiều chiều. Trung bình là một yếu tố quan trọng giúp xác định vị trí của từng cụm.
- Phương sai (Variances): Phương sai σ^2 của mỗi thành phần thể hiện mức độ phân tán của dữ liệu trong cụm đó. Phương sai có thể khác nhau giữa các thành phần, cho phép mỗi cụm có hình dạng và kích thước riêng biệt.
- Hệ số hỗn hợp (Mixture Coefficients): Hệ số hỗn hợp π đại diện cho trọng số của mỗi phân phối Gaussian trong tổng thể GMM, cho biết tầm quan trọng tương đối của từng phân phối thành phần. Tổng các hệ số hỗn hợp bằng 1, thể hiện việc chia tỉ lệ đóng góp của mỗi thành phần vào mô hình tổng thể.

GMM hoạt động dựa trên nguyên tắc rằng mỗi điểm dữ liệu có thể được xem như một mẫu từ một trong các phân phối Gaussian, với xác suất thuộc về mỗi phân phối được xác định bởi hệ số hỗn hợp. Sử dụng thuật toán Expectation-Maximization (EM), GMM tìm cách tối ưu hóa các tham số này để phù hợp nhất với dữ liệu quan sát, cung cấp một cách mạnh mẽ để mô hình hóa và phân tích dữ liệu phức tạp.

2 Quá trình xây dựng Model

2.1 Khai báo các thư viện cần thiết

Khai báo một vài thư viện để phục vụ cho quá trình xây dựng chương trình.

- cv2: Đây là một phần của OpenCV (Open Source Computer Vision Library), một thư viện phổ biến được sử dụng để xử lý hình ảnh và thị giác máy tính. OpenCV cung cấp các công cụ và chức năng để đọc và ghi hình ảnh, xử lý hình ảnh
- os: Thư viện này cung cấp các chức năng để tương tác với hệ điều hành.Có thể thực hiện các thao tác như tạo, xóa, đổi tên và di chuyển các tệp và thư mục, xác định đường dẫn tệp và thư mục, kiểm tra sự tồn tại của tệp và thư mục
- glob: Thư viện này cung cấp một phương pháp để tìm kiếm các tệp trên hệ thống. Bằng cách sử dụng glob, bạn có thể tìm kiếm các tệp theo mẫu định sẵn hoặc mẫu mà bạn chỉ định.
- numpy: Cung cấp hỗ trợ cho mảng và ma trận đa chiều, cùng với một loạt các hàm toán học để làm việc với chúng.
- natsort: Thư viện này cung cấp các chức năng để sắp xếp các chuỗi số theo thứ tự tự nhiên.

2.2 Xây dựng các hàm tính toán

Xây dựng các hàm tính toán như pdf_value(), Gaussian_Mixture() với các mục đích cụ thể dưới đây:
pdf_value(x1, u1, sigma1), với đầu vào là điểm ảnh x1, giá trị trung bình u1 và độ lệch chuẩn sigma1.

- Hàm này tính toán giá trị của hàm mật độ xác suất (PDF) tại một điểm x1 dựa trên các tham số của phân phối Gaussian có trung bình u1 và độ lệch chuẩn sigma1.
- Giá trị PDF này được sử dụng trong thuật toán EM (Expectation-Maximization) để tính toán xác suất điểm dữ liệu thuộc vào mỗi phân phối Gaussian trong mô hình GMM (Gaussian Mixture Model).

Gaussian_Mixture(x, u1, sigma1, u2, sigma2, w1, w2)

- Hàm này triển khai thuật toán EM cho mô hình GMM với hai phân phối Gaussian.Trong mỗi vòng lặp của thuật toán EM, các bước E (Expectation) và M (Maximization) được thực hiện để cập nhật các tham số của hai phân phối Gaussian: trung bình (mean), độ lệch chuẩn (standard deviation), và trọng số (weight).
- Thuật toán tiếp tục lặp lại cho đến khi đạt được điều kiện dừng hoặc số lần lặp vượt quá một ngưỡng nhất định. Kết quả của thuật toán EM là trung bình cuối cùng của phân phối Gaussian nào đó, được xác định bởi trọng số lớn hơn giữa hai phân phối. Nếu thuật toán không hội tụ, nó sẽ trả về trung bình của tất cả các điểm dữ liệu.

2.3 Xử lý video đầu vào

Đoạn video sẽ được xử lý bằng cách sử dụng thư viện cv2 để lấy ra tất cả các khung hình có trong video. Các khung hình này sẽ được chuyển sang ảnh xám và được lưu vào một thư mục data/frame để phục vụ cho các tính toán sau này.

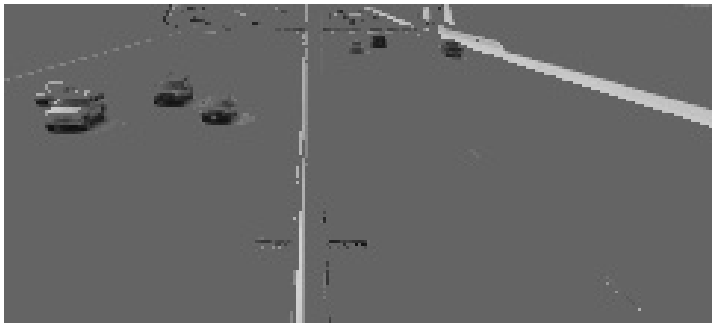
2.4 Áp dụng GMM cho từng khung hình

Sau khi đã thu thập được tất cả khung hình trong video đầu vào, ta sẽ áp dụng GMM đã được xây dựng trước đó cho từng điểm ảnh trên từng khung hình để thu về

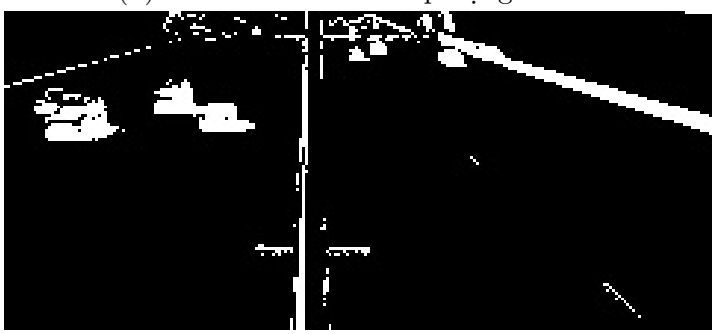
3 Kết quả

Dưới đây là một vài hình ảnh được trích xuất từ video đầu vào và được xử lý bởi GMM

(a) Frame 210 - Video gốc



(b) Frame 210 sau khi áp dụng GMM



(c) Frame 210 lọc nền và đối tượng từ GMM

Code

Link group's code on colab.

```
"""GMM.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/122Ac84juP3fuRvd7gfSCzr9LH9wztrqm
"""
import cv2
import os
import glob
import time

vid = cv2.VideoCapture("/GMM/video/Original-Video.mp4")
currentframe=1
count = 0
image = 1

if not os.path.exists('data2'):
    os.makedirs('data2')

while(True):
    success, frame = vid.read()
    if not success:
        print("Error reading frame. Exiting loop.")
        break
    count += 1

    if count%10 == 0: # to limit the number of frames.
      gray_image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
      gray_image = gray_image[123:940, 239:2155]
      cv2.imwrite('./data2/frame' + str(image) + '.jpg', gray_image)
      image += 1

    if count > 7200: # 720
        break
    currentframe += 1

cv2.destroyAllWindows()
print(count)

```

```
41 import numpy as np
42 def pdf_value(x1,u1,sigma1):
43   p=((x1-u1)*(x1-u1))/(2*sigma1*sigma1)
44   t = (1/np.sqrt((2*3.14*sigma1*sigma1)))*(np.exp(-p))
45   return t
46
47 def Gaussian_Mixture(x, u1, sigma1, u2, sigma2, w1, w2):
48   mean1 = u1
49   mean2 = u2
50   weight1 = w1
51   weight2 = w2
52   SD1 = sigma1
53   SD2 = sigma2
54   count = 0
55   while True:
56     m1 = mean1
57     m2 = mean2
58     we1 = weight1
59     we2 = weight2
60     si1 = SD1
61     si2 = SD2
62     count += 1
63
64     print(mean1, SD1, mean2, SD2, weight1, weight2, count)
65
66     prob_array_cluster1 = []
67     prob_array_cluster2 = []
68     for i in range(len(x)):
69       # calculating the pdf values coorsponding to each entry in
           x
70       prob_array_cluster1.append(pdf_value(x[i], mean1, SD1))
71       prob_array_cluster2.append(pdf_value(x[i], mean2, SD2))
72
73     prob_element_in_cluster1=[] #  b_k
74     prob_element_in_cluster2=[]
75
76     for i in range(len(x)):
77       # calculating bayers probability of x[i] belonging to both
           the clusters
78       prob_element_in_cluster1.append((prob_array_cluster1[i]*
           weight1)/(prob_array_cluster1[i]*weight1+
           prob_array_cluster2[i]*weight2))
79       prob_element_in_cluster2.append((prob_array_cluster2[i]*
           weight2)/(prob_array_cluster1[i]*weight1+
           prob_array_cluster2[i]*weight2))
80     weight1=sum(prob_element_in_cluster1)/2  # updating the
```

```
        weights
    weight2=sum(prob_element_in_cluster2)/2

    d1=0  # sigma
    d2=0
    for i in range(len(x)):
      d1 = d1+prob_element_in_cluster1[i]*(x[i]-mean1)*(x[i]-
         mean1)
      d2 = d2+prob_element_in_cluster2[i]*(x[i]-mean2)*(x[i]-
         mean2)
    SD1=np.sqrt(d1/(2*weight1)) # upadting the varience/standard
        deviation
    SD2=np.sqrt(d2/(2*weight2))

    t1=0   # u_k
    t2=0
    for i in range(len(x)):
      t1 = t1+prob_element_in_cluster1[i]*x[i]
      t2 = t2+prob_element_in_cluster2[i]*x[i]
    mean1=t1/(2*weight1)
    mean2=t2/(2*weight2) # updating the mean values

    # convergence condition
    if (abs(mean1-m1)<0.1 or abs(mean2-m2)<0.1 or mean1>10000000
        or mean2>10000000 or weight1>10000000 or weight1>10000000
        or SD1>1000000 or SD2>10000000 or count>15):
      if count>15:
        return (sum(x)/len(x)) # if the algorithm doesnt converge
           , mean is returned as the output
        # return 255

      elif weight1>weight2:
        # print(mean1)
        #print(mean1, mean2)
        return mean1

      else:
        # print(mean2)
        #print(mean1, mean2)
        return mean2

import fnmatch
import numpy as np
array = []
IMAGE_DIR = '/GMM/data2'

```

```
image_names = []
image_dictionary = []

image_1D = []
# Walk through the directory and collect image names
for root, dirnames, filenames in os.walk(IMAGE_DIR):
    for filename in fnmatch.filter(filenames, "*.*"):
        image_names.append(os.path.join(root, filename))

# Read and preprocess images
for idx, image_name in enumerate(image_names):
    img = cv2.imread(image_name, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"Error reading image: {image_name}")
        continue
    img = img.astype(np.float64)
    img = cv2.resize(img, (240, 108), interpolation=cv2.
       INTER_AREA)
    array.append(img)

count = 0
# final_array = np.zeros((240,108))
final_array = np.zeros((108,240))
for i in range(len(array[0])): #rows
  for j in range(len(array[0][0])): #cols
    x = []
    for e in array: #frames
      x.append(e[i][j])
    summ = sum(x)
    avg = summ/len(x)
    xx = (Gaussian_Mixture(x,(avg),3,(avg*2),4,1/2,1/2)) #
       appling gaussian mixture to each pixel location values
       across all the frames to generate the background
    final_array[i][j] = xx; # Extracted background
print(final_array)

import matplotlib.pyplot as plt
fig,axarr = plt.subplots()
axarr.set_title(" plot_mean_vector")
# avg_image = np.reshape(final_array, (240,108))

axarr.imshow(final_array, cmap=plt.cm.gray)

from google.colab.patches import cv2_imshow
image=cv2.imread('/GMM/data2/frame6.jpg')
image2 = cv2.resize(image, (240,108),
```

```
                interpolation = cv2.INTER_NEAREST)
gray_image = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
dim1 = gray_image.shape
# print(dim1)
dim = final_array.shape

print(dim)
cv2_imshow(image2)
cv2_imshow(final_array)

gray_image = np.asarray(gray_image, np.float64)
final_array = np.asarray(final_array, np.float64)
image_final = cv2.absdiff(gray_image, final_array)

if not os.path.exists('answer2'):
    os.makedirs('answer2')

img_array = []
for p in range(1,len(array)+1):
  img1 = cv2.imread('/GMM/data2/frame' + str(p) + '.jpg')
  gray_image = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
  # print((np.shape(gray_image)))
  gray_image = cv2.resize(gray_image, (240,108), interpolation =
     cv2.INTER_AREA)

  # subtracting the background from the original frames
  m = np.zeros((108,240))
  for i in range(len(final_array)):
    for j in range(len(final_array[0])):
      diff = final_array[i][j] - gray_image[i][j]
      if (abs(diff)>30): #threshold to check similarity betwwen
         two pixels for pixel classification as either foreground
          or background
        m[i][j]= gray_image[i][j]
      else:
        m[i][j] = 100
  # print(m)
  cv2.imwrite('./answer2/frame' + str(p) + '.jpg', m)
  height, width = m.shape
  size = (width,height)
  img_array.append(m)
  # cv2_imshow(m)
"""# Save video"""
import natsort
import cv2
import numpy as np
```

```
206 import glob
207 # sorting foreground frames for video generation
208 img_array = []
209 for filename in glob.glob('/GMM/answer2/*.jpg'):
210     img_array.append(filename)
211 sorted_arr = natsort.natsorted(img_array)
212
213 out = cv2.VideoWriter('/GMM/video/GMM-Video-2.avi',cv2.
    VideoWriter_fourcc(*'DIVX'), 15, size)
214 for i in range(len(sorted_arr)):
215   out.write(cv2.imread(sorted_arr[i]))
216 out.release()
217 cv2_imshow(cv2.imread('/GMM/answer2/frame27.jpg')) # Output
218 img = cv2.imread('/GMM/data2/frame27.jpg')  # Raw Frame
219 img = cv2.resize(img, (240,108))
220 cv2_imshow(img)
221
222 """# Filter frames"""
223
224 if not os.path.exists('frames_for_comparison-2'):
225     os.makedirs('frames_for_comparison-2')
226 img_array = []
227 for p in range(1,len(array)+1):
228   img1 = cv2.imread('/GMM/data2/frame' + str(p) + '.jpg')
229   gray_image = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
230   # print((np.shape(gray_image)))
231   gray_image = cv2.resize(gray_image, (240,108), interpolation =
      cv2.INTER_AREA)
232   m = np.zeros((108,240))
233   for i in range(len(final_array)):
234     for j in range(len(final_array[0])):
235       diff = final_array[i][j] - gray_image[i][j] # # subtracting
          the background from the original frames to generate
          only black and white foreground for comparison with
          built-in function
236       if (abs(diff)>30):
237         m[i][j]= 255
238       else:
239         m[i][j] = 0
240   # print(m)
241   cv2.imwrite('./frames_for_comparison-2/frame' + str(p) + '.jpg'
      , m)
242   height, width = m.shape
243   size = (width,height)
244   img_array.append(m)
245
```

```
import cv2
import os
import math
import glob
import time
from skimage.metrics import mean_squared_error
import csv
vid = cv2.VideoCapture("/GMM/video/car-on-highway.mp4")
fgbg = cv2.createBackgroundSubtractorMOG2()
currentframe=1
count = 0
image = 1
from PIL import Image
RMSE=[]
while(True):
    success, frame = vid.read()
    count += 1

    if count%10 == 0:
      gray_image = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
      gray_image = gray_image[123:940, 239:2155]
      fgmask = fgbg.apply(gray_image) # in-built function for
          foreground generation

      gray_image = cv2.resize(gray_image, (240,108))
      fgmask = cv2.resize(fgmask, (240,108))

      our_image = cv2.imread('/GMM/frames_for_comparison-2/frame'
          + str(image) + '.jpg')
      our_image = cv2.cvtColor(our_image, cv2.COLOR_BGR2GRAY)
      print('our foreground')
      cv2_imshow(our_image)
      m = np.zeros((108,240))
      for i in range(len(final_array)):
        for j in range(len(final_array[0])):
          diff = fgmask[i][j]
          if (diff>10):
            m[i][j]= 255
          else:
            m[i][j] = 0
      print('in-built function foreground')
      cv2_imshow(m)
      dim1 = our_image.shape
      dim2 = gray_image.shape
      RMSE.append(math.sqrt(mean_squared_error(our_image, m))) #
          comparing both the generated foregrounds
```

```
289         image += 1
290       if count > 720:
291           break
292       currentframe += 1
293 
294 """## Save filter"""
295 import natsort
296 import cv2
297 import numpy as np
298 import glob
299 
300 # sorting foreground frames for video generation
301 img_array = []
302 for filename in glob.glob('/GMM/frames_for_comparison-2/*.jpg'):
303       img_array.append(filename)
304 sorted_arr = natsort.natsorted(img_array)
305 
306 out = cv2.VideoWriter('/GMM/video/GMM-Video-filter-v2.avi',cv2.
      VideoWriter_fourcc(*'DIVX'), 15, size)
307 for i in range(len(sorted_arr)):
308   out.write(cv2.imread(sorted_arr[i]))
309 out.release()
310 
311 a=RMSE[21:len(RMSE)-3] # storing the foregrounds in an array
312 cv2.destroyAllWindows()
```