МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. І.СІКОРСЬКОГО»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Кафедра фізико-технічних засобів захисту інформації

Лабораторна робота № 2

з дисципліни: «Автоматизація обробки ІзОД»

Варіант №1

Керівник: Виконав:

Прогонов Дмитро Олександрович студент 5 курсу

групи ФЕ-91мп

Захищено з оцінкою Баліцький Олег Анатолійович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, підпис

Київ – 2020 р.

**Хід роботи**

1. Сформувати тестову вибірку зображень з вихідного пакета;

Блок коду програми мовою високого рівня:

from os import listdir

from matplotlib import image

import random

import numpy as np

from dataclasses import dataclass

# load images in a directory

random.seed(1)

random\_indexes = random.sample(range(25000), 250)

loaded\_images = list()

for i in range(250):

# load image

filename = 'im' + str(random\_indexes[i]) + '.jpg'

img\_data = image.imread('Q:/mirflickr25k/mirflickr/' + filename)

# store loaded image

loaded\_images.append(img\_data)

print('> loaded %s %s' % (filename, img\_data.shape))

2. Для кожного каналу кольору кожного зображення з тестового пакета обчислити наступні характеристики:

a. Математичне сподівання і дисперсію;

# Expected value & Variance

#RED

sum\_val = sum(values[RED])

M\_red = 0

for index in range(len(values[RED])):

p = (values[RED][index] / sum\_val)

M\_red += p \* index

D\_red = 0

for index in range(len(values[RED])):

p = (values[RED][index] / sum\_val)

D\_red += p \* ((index - M\_red) \*\* 2)

print("Red: Expected value - {0:.2f}, Variance - {1:.2f}"\

.format(M\_red, D\_red))

#GREEN

sum\_val = sum(values[GREEN])

M\_green = 0

for index in range(len(values[GREEN])):

p = (values[GREEN][index] / sum\_val)

M\_green += p \* index

D\_green = 0

for index in range(len(values[GREEN])):

p = (values[GREEN][index] / sum\_val)

D\_green += p \* ((index - M\_green) \*\* 2)

print("Green: Expected value - {0:.2f}, Variance - {1:.2f}"\

.format(M\_green, D\_green))

#BLUE

sum\_val = sum(values[BLUE])

M\_blue = 0

for index in range(len(values[BLUE])):

p = (values[BLUE][index] / sum\_val)

M\_blue += p \* index

D\_blue = 0

for index in range(len(values[BLUE])):

p = (values[BLUE][index] / sum\_val)

D\_blue += p \* ((index - M\_blue) \*\* 2)

print("Blue: Expected value - {0:.2f}, Variance - {1:.2f}"\

.format(M\_blue, D\_blue))

Вихідні дані:

*Red: Expected value - 113.95, Variance - 5852.65*

*Green: Expected value - 107.31, Variance - 5392.78*

*Blue: Expected value - 99.49, Variance - 5774.33*

b. Коефіцієнти асиметрії та ексцесу (нормалізований);

#Asymmetry and excess

Asym\_red = E\_operator(values[RED], M\_red, 3) / (D\_red \*\* (3 / 2))

Asym\_green = E\_operator(values[GREEN], M\_green, 3) / (D\_green \*\* (3 / 2))

Asym\_blue = E\_operator(values[BLUE], M\_blue, 3) / (D\_blue \*\* (3 / 2))

Excess\_red = E\_operator(values[RED], M\_red, 4) / (D\_red \*\* 2)

Excess\_green = E\_operator(values[GREEN], M\_green, 4) / (D\_green \*\* 2)

Excess\_blue = E\_operator(values[BLUE], M\_blue, 4) / (D\_blue \*\* 2)

print('Red: Asymmetry - {0:.3f}, Excess - {1:.3f}'\

.format(Asym\_red, Excess\_red))

print('Green: Asymmetry - {0:.3f}, Excess - {1:.3f}'\

.format(Asym\_green, Excess\_green))

print('Blue: Asymmetry - {0:.3f}, Excess - {1:.3f}'\

.format(Asym\_blue, Excess\_blue))

Вихідні дані:

*Red: Asymmetry - 0.174, Excess - 1.853*

*Green: Asymmetry - 0.275, Excess - 1.994*

*Blue: Asymmetry - 0.425, Excess - 2.024*

1. Побудувати вектори параметрів зображень, що складаються з:

a. Математичних очікувань значень яскравості для кожного каналу кольору;

b. Математичних очікувань і дисперсії значень яскравості для кожного каналу кольору;

c. Математичних очікувань, дисперсії і коефіцієнта асиметрії значень яскравості для кожного каналу кольору;

d. Математичних очікувань, дисперсії, коефіцієнтів асиметрії та ексцесу значень яскравості для кожного каналу кольору;

Vector\_A =

np.array([np.array([M\_red, D\_red, Asym\_red, Excess\_red]),

np.array([M\_green, D\_green, Asym\_green, Excess\_green]),

np.array([M\_blue, D\_blue, Asym\_blue, Excess\_blue])])

print("Vector\_A:\n" + str(Vector\_A))

Vector\_All\_DATA = np.copy(Vector\_A)

itear = 0

for image in loaded\_images:

image = np.reshape(image, (-1, 3))

image = np.swapaxes(image, 0, 1)

Vector\_All\_DATA = np.concatenate((Vector\_All\_DATA,image),axis=1)

itear += 1

if (itear % 10 == 0):

print(str(itear) + " img")

Вихідні дані:

Vector\_A:

[[1.13946581e+02 5.85265242e+03 1.73982067e-01 1.85309506e+00]

[1.07313605e+02 5.39277757e+03 2.75405534e-01 1.99447289e+00]

[9.94871271e+01 5.77433321e+03 4.25110161e-01 2.02427163e+00]]

10 img

\*\*\*

250 img

4. Побудувати гаусові моделі зображень з використанням розрахованих раніше

параметрів.

#a Expected value

P\_x1 = np.random.normal(M\_red, D\_red, 1)

print("Expected value + colors:\n" + str(P\_x1))

#b Expected value and dispersion

P\_x2 = np.cov(Vector\_All\_DATA)

print("Expected value + dispersion + colors:\n" + str(P\_x2[:2, :2]))

#c Expected value, dispersion and asymetry

print("Expected value + dispersion + asymetry + colors:\n" + str(P\_x2[:3, :3]))

#d Expected value, dispersion, asymetry and excess

print("Expected value + dispersion + asymetry + excess + colors:\n" + str(P\_x2[:4, :4]))

Вихідні дані:

Expected value + colors:

[2077.79126664]

Expected value + dispersion + colors:

[[5853.39076062 5075.84012833]

[5075.84012833 5393.40387733]]

Expected value + dispersion + asymetry + colors:

[[5853.39076062 5075.84012833 4529.96354074]

[5075.84012833 5393.40387733 5112.40411108]

[4529.96354074 5112.40411108 5775.05507684]]

Expected value + dispersion + asymetry + excess + colors:

[[5853.39076062 5075.84012833 4529.96354074]

[5075.84012833 5393.40387733 5112.40411108]

[4529.96354074 5112.40411108 5775.05507684]

[5853.39076062 5075.84012833 4529.96354074]]

5. Провести декомпозицію кожного каналу кольору кожного зображення з застосуванням методу головних компонент (PCA):

a. Варіюючи кількість компонент, провести реконструкцію окремих каналів кольору зображень (від компонент з найбільшою енергією поступово переходячи до компонентів з мінімальною енергією).

import scipy

import scipy.ndimage

import matplotlib.pyplot as plt

import numpy as np

from PIL import Image

test\_img = loaded\_images[68]

a\_np = np.array(test\_img)

a\_r = a\_np[:,:,0]

a\_g = a\_np[:,:,1]

a\_b = a\_np[:,:,2]

def PCA\_2d(image\_2d, numpc):

cov\_mat = image\_2d - np.mean(image\_2d)

eig\_val, eig\_vec = np.linalg.eigh(np.cov(cov\_mat))

p = np.size(eig\_vec, axis =1)

idx = np.argsort(eig\_val)

idx = idx[::-1]

eig\_vec = eig\_vec[:,idx]

eig\_val = eig\_val[idx]

if numpc <p or numpc >0:

eig\_vec = eig\_vec[:, range(numpc)]

score = np.dot(eig\_vec.T, cov\_mat)

recon = np.dot(eig\_vec, score) + np.mean(image\_2d).T

recon\_img\_mat = np.uint8(np.absolute(recon))

return recon\_img\_mat

a\_r\_recon, a\_g\_recon, a\_b\_recon = PCA\_2d(a\_r, 5), PCA\_2d(a\_g, 5), PCA\_2d(a\_b, 5)

recon\_color\_img = np.dstack((a\_r\_recon, a\_g\_recon, a\_b\_recon))

fig=plt.figure(figsize=(10, 10))

fig.add\_subplot(1, 2, 1)

plt.title('Original')

plt.imshow(loaded\_images[68])

fig.add\_subplot(3, 2, 2)

plt.title('5 components')

plt.imshow(recon\_color\_img)

a\_r\_recon, a\_g\_recon, a\_b\_recon = PCA\_2d(a\_r, 25), PCA\_2d(a\_g, 25), PCA\_2d(a\_b, 25)

recon\_color\_img = np.dstack((a\_r\_recon, a\_g\_recon, a\_b\_recon))

fig.add\_subplot(3, 2, 4)

plt.title('25 components')

plt.imshow(recon\_color\_img)

a\_r\_recon, a\_g\_recon, a\_b\_recon = PCA\_2d(a\_r, 75), PCA\_2d(a\_g, 75), PCA\_2d(a\_b, 75)

recon\_color\_img = np.dstack((a\_r\_recon, a\_g\_recon, a\_b\_recon))

fig.add\_subplot(3, 2, 6)

plt.title('75 components')

plt.imshow(recon\_color\_img)

plt.show()

# recon\_color\_img = Image.fromarray(recon\_color\_img)

# recon\_color\_img.show()



b. Побудувати залежність помилки відновлення (середнє відхилення вихідного зображення відреконструйованого, MSE) від кількості використаних компонент.

def mse(imageA, imageB):

# the 'Mean Squared Error' between the two images is the

# sum of the squared difference between the two images;

err = np.sum((imageA.astype("float") - imageB.astype("float")) \*\* 2)

err /= float(imageA.shape[0] \* imageA.shape[1])

return err

mse\_list = list()

for i in range(100):

a\_r\_recon, a\_g\_recon, a\_b\_recon = PCA\_2d(a\_r, i), PCA\_2d(a\_g, i), PCA\_2d(a\_b, i)

recon\_color\_img = np.dstack((a\_r\_recon, a\_g\_recon, a\_b\_recon))

mse\_list.append(mse(test\_img, recon\_color\_img))

plt.plot(range(len(mse\_list)),mse\_list)

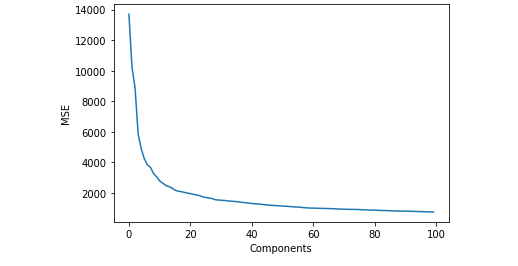
plt.xlabel("Components")

plt.ylabel("MSE")

plt.show()

Вихідні дані:

Гістограма залежності помилки відновлення від кількості використаних компонент:



6. Провести моделювання окремих каналів кольору зображень з використанням марковських ланцюгів:

a. Для кожного каналу кольору кожного зображення розрахувати стохастическую матрицю марковської ланцюга першого і другого порядків (обробка пікселів по горизонталі справа наліво і навпаки, а також по вертикалі зверху вниз і навпаки). У звіті привести явний вигляд одного марковського ланцюга для одного з каналів кольору тестового зображення;

b. Перевірити властивість регулярності, реккурентное і незворотності (irreducible) для отриманих марковских моделей для 5 ітерацій.

markov\_matrix1 = np.zeros(shape=(256, 256))

#c-type

arr = a\_r.flatten()

prev\_color = arr[0]

for i in range(len(arr) - 1):

markov\_matrix1[arr[i]][arr[i + 1]] += 1

markov\_matrix = markov\_matrix1[0] / sum(markov\_matrix1[0])

for i in range(1, 256):

markov\_matrix = np.vstack((markov\_matrix, markov\_matrix1[i] / sum(markov\_matrix1[i])))

print("Red matrix 1st oder:\n", markov\_matrix)

print("\nRed matrix 2nd order:\n", np.linalg.matrix\_power(markov\_matrix, 2))

markov\_matrix1 = np.zeros(shape=(256, 256))

#Fortran-type

arr = a\_r.flatten('F')

prev\_color = arr[0]

for i in range(len(arr) - 1):

markov\_matrix1[arr[i]][arr[i + 1]] += 1

markov\_matrix = markov\_matrix1[0] / sum(markov\_matrix1[0])

for i in range(1, 256):

markov\_matrix = np.vstack((markov\_matrix, markov\_matrix1[i] / sum(markov\_matrix1[i])))

print("Red matrix 2-nd type 1st oder:\n", markov\_matrix)

print("\nRed matrix 2-nd type 2nd order:\n", np.linalg.matrix\_power(markov\_matrix, 2))

markov\_matrix2 = np.zeros(shape=(256, 256))

arr = a\_g.flatten()

prev\_color = arr[0]

for i in range(len(arr) - 1):

markov\_matrix2[arr[i]][arr[i + 1]] += 1

markov\_matrix = markov\_matrix2[0] / sum(markov\_matrix2[0])

for i in range(1, 256):

markov\_matrix = np.vstack((markov\_matrix, markov\_matrix2[i] / sum(markov\_matrix2[i])))

print("\n\nGreen matrix 1st oder:\n", markov\_matrix)

print("\nGreen matrix 2nd order:\n", np.linalg.matrix\_power(markov\_matrix, 2))

markov\_matrix3 = np.zeros(shape=(256, 256))

arr = a\_g.flatten()

prev\_color = arr[0]

for i in range(len(arr) - 1):

markov\_matrix3[arr[i]][arr[i + 1]] += 1

markov\_matrix = markov\_matrix3[0] / sum(markov\_matrix3[0])

for i in range(1, 256):

markov\_matrix = np.vstack((markov\_matrix, markov\_matrix3[i] / sum(markov\_matrix3[i])))

print("\n\nBlue matrix 1st oder:\n", markov\_matrix)

print("\nBlue matrix 2nd order:\n", np.linalg.matrix\_power(markov\_matrix, 2))

Вихідні дані:

Red matrix 1st oder:

[[0.07692308 0.04682274 0.04013378 ... 0. 0. 0. ]

[0.0316092 0.07183908 0.08908046 ... 0. 0. 0. ]

[0.01810865 0.06237425 0.25150905 ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.05847953 0.11695906 0.1871345 ]

[0. 0. 0. ... 0.05594406 0.25874126 0.17482517]

[0. 0. 0. ... 0.0651341 0.08237548 0.11111111]]

Red matrix 2nd order:

[[1.11080021e-02 1.31638624e-02 2.36315591e-02 ... 0.00000000e+00

0.00000000e+00 1.20508689e-05]

[9.96073999e-03 1.74135083e-02 4.06389352e-02 ... 0.00000000e+00

0.00000000e+00 8.37773533e-06]

[1.19878467e-02 2.74696464e-02 8.47217280e-02 ... 0.00000000e+00

1.57439158e-06 3.16126788e-06]

...

[1.10854719e-04 8.03887479e-05 6.92407415e-05 ... 3.88979152e-02

7.18501256e-02 1.02970456e-01]

[9.49810110e-05 9.86965394e-05 3.79249341e-05 ... 4.18411819e-02

1.03734821e-01 1.16367166e-01]

[1.61171448e-04 1.82108303e-04 6.33520106e-05 ... 2.80996646e-02

5.39100264e-02 8.00357217e-02]]

Red matrix 2-nd type 1st oder:

[[0.20735786 0.11705686 0.0735786 ... 0. 0. 0. ]

[0.09770115 0.13793103 0.10632184 ... 0. 0. 0. ]

[0.04426559 0.09054326 0.27967807 ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.12865497 0.16374269 0.2748538 ]

[0. 0. 0. ... 0.0979021 0.30769231 0.3041958 ]

[0. 0. 0. ... 0.10344828 0.14942529 0.40038314]]

Red matrix 2-nd type 2nd order:

[[6.07341694e-02 5.15571482e-02 5.76762221e-02 ... 8.05899182e-06

3.66317810e-06 3.97972773e-05]

[4.35537006e-02 4.78795808e-02 7.00414438e-02 ... 0.00000000e+00

1.23838146e-05 1.64887323e-05]

[3.64793068e-02 5.26864753e-02 1.15071965e-01 ... 0.00000000e+00

2.20380332e-06 2.21106861e-06]

...

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 8.01131504e-02

1.39520571e-01 2.34494384e-01]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 8.59140597e-02

1.73461281e-01 2.68163009e-01]

[1.37523964e-06 5.52789293e-06 1.49898968e-06 ... 8.14261130e-02

1.39874774e-01 2.59719894e-01]]

Green matrix 1st oder:

[[0.07948718 0.04487179 0.04871795 ... 0. 0. 0. ]

[0.07079646 0.08141593 0.09026549 ... 0. 0. 0. ]

[0.03311966 0.03952991 0.16987179 ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.0872093 0.09302326 0.20930233]

[0. 0. 0. ... 0.06521739 0.25362319 0.10144928]

[0. 0. 0. ... 0.08970976 0.08443272 0.11345646]]

Green matrix 2nd order:

[[1.75798130e-02 1.52899239e-02 2.72334173e-02 ... 0.00000000e+00

4.65892785e-06 0.00000000e+00]

[2.07836619e-02 2.00922445e-02 4.05062252e-02 ... 0.00000000e+00

4.99634285e-06 0.00000000e+00]

[1.78703852e-02 1.98368593e-02 5.50294035e-02 ... 0.00000000e+00

2.78949365e-06 0.00000000e+00]

...

[4.43051867e-04 1.77507969e-04 8.94991830e-05 ... 4.54162898e-02

7.19320091e-02 8.66927630e-02]

[6.20723612e-04 1.88090005e-04 1.11351153e-04 ... 4.01459336e-02

9.38449409e-02 7.49416061e-02]

[4.84279018e-04 1.57925614e-04 6.87408783e-05 ... 3.47681737e-02

6.09946949e-02 7.12871463e-02]]

Blue matrix 1st oder:

[[0.07948718 0.04487179 0.04871795 ... 0. 0. 0. ]

[0.07079646 0.08141593 0.09026549 ... 0. 0. 0. ]

[0.03311966 0.03952991 0.16987179 ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.0872093 0.09302326 0.20930233]

[0. 0. 0. ... 0.06521739 0.25362319 0.10144928]

[0. 0. 0. ... 0.08970976 0.08443272 0.11345646]]

Blue matrix 2nd order:

[[1.75798130e-02 1.52899239e-02 2.72334173e-02 ... 0.00000000e+00

4.65892785e-06 0.00000000e+00]

[2.07836619e-02 2.00922445e-02 4.05062252e-02 ... 0.00000000e+00

4.99634285e-06 0.00000000e+00]

[1.78703852e-02 1.98368593e-02 5.50294035e-02 ... 0.00000000e+00

2.78949365e-06 0.00000000e+00]

...

[4.43051867e-04 1.77507969e-04 8.94991830e-05 ... 4.54162898e-02

7.19320091e-02 8.66927630e-02]

[6.20723612e-04 1.88090005e-04 1.11351153e-04 ... 4.01459336e-02

9.38449409e-02 7.49416061e-02]

[4.84279018e-04 1.57925614e-04 6.87408783e-05 ... 3.47681737e-02

6.09946949e-02 7.12871463e-02]]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Графічне представлення ланцюга:

#Граф

import networkx as nx

iport pandas as pd

data = markov\_matrix

data = np.triu(data) + np.triu(data).T

ind = [str(i) for i in range(data.shape[0])]

df2 = pd.DataFrame(data, index=ind, columns=ind)

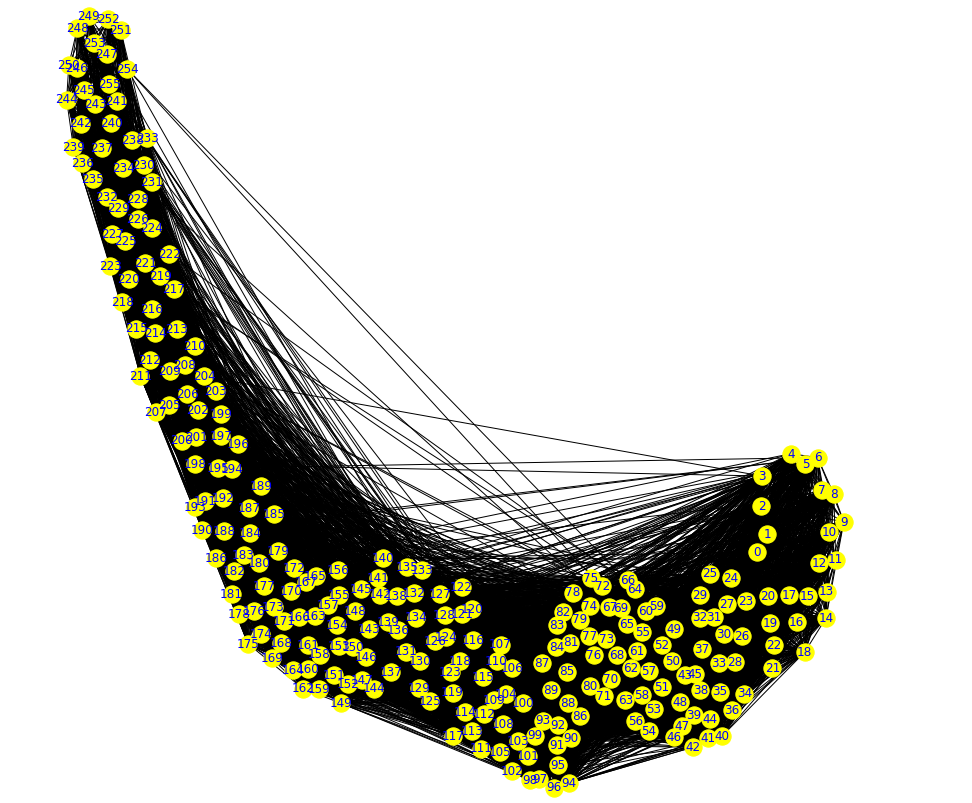
plt.figure(1,figsize=(12,12))

G2 = nx.from\_pandas\_adjacency(df2)

nx.draw(G2, with\_labels=True, node\_color='yellow', font\_color='blue')

plt.show()

Вихідні дані:



**Перевірка коректності гіпотез:**

import quantecon as qe

def regularity\_check(matrix):

counter = 0

for a in range(1,6):

matrix = np.linalg.matrix\_power(matrix,a)

for i in range(matrix.shape[0]):

for j in range(matrix.shape[1]):

if matrix[i,j] < 0:

counter += 1

print('iteration: ' , a, 'negative elements: ' ,counter)

regularity\_check(markov\_matrix)

print('Для 5 ітерації ,виконується умова регулярності.Модель регулярна ')

P = markov\_matrix

mc = qe.MarkovChain(P, [str(i) for i in range(0,256)])

print('Модель незворотня - ', mc.is\_irreducible)

print('Рекурентную називають модель M яка з ймовірністю в P = 1 при покиданні стану вернеться в цей стан за час t < inf,у майбутньму')

print("Кількість рекурентних станів " ,np.shape(mc.recurrent\_classes)[1])

if np.shape(mc.recurrent\_classes\_indices)[1] == 256:

print('Усі стани моделі рекурентні ,модель рекурентна ')

else :

print('Модель не рекурентна' )

Вихідні дані:

iteration: 1 negative elements: 0

iteration: 2 negative elements: 0

iteration: 3 negative elements: 0

iteration: 4 negative elements: 0

iteration: 5 negative elements: 0

Для 5 ітерації ,виконується умова регулярності.Модель регулярна

Модель незворотня - True

Рекурентную називають модель M яка з ймовірністю в P = 1 при покиданні стану вернеться в цей стан за час t < inf,у майбутньму

Кількість рекурентних станів 256

Усі стани моделі рекурентні ,модель рекурентна

**Висновки**

В даній лабораторній роботі було проаналізовано вибірку з 250 зображень датасету MIRFlickr-20k. Було знайдено що всі канали охоплюють увесь спектр значень. Було знайдено мат. очікування – 114 для чевоного каналу, 107 для зеленого і 99 для синьогоканалу відповідно і дисперсію 5852.65, 5392.78 та 5774.33.

Було побудовано вектори даних та знайдено Гаусовські моделі для одновимірного та багатовимірних варіантів в залежності від кількості даних.

За допомогою методу головних компонент було відновлено тестові зображення та показано, що при збільшенні кількості компонент зростає якість відновлення. Також було помічено та проаналізовано помилки відновлення. Зібравши дані, було побудовано графік залежності середньої квадратичної похибки відновлених зображень від кількості компонент. Було помічено експоненціальну залежність, що свідчить про значні зміни при невеликих кількостях компонент (до 20) та майже непомітні при великих значеннях (більше 100).

Було проведено моделювання окремих каналів кольору зображень з використанням марківських ланцюгів, та сформовано стохастичні матриці за різними типами обходів. Також за даними було побуловано графічну модель марківського ланцюга.