Кафедра фізико-технічних засобів захисту інформації

Лабораторна робота № 2

з дисципліни: «Автоматизація обробки ІзОД»

Варіант №7

Керівник: Виконав:

Прогонов Дмитро Олександрович студент 5 курсу групи ФЕ-91мп

Захищено з оцінкою Нікішин М.Ю.

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

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

дата, підпис

Київ – 2020 р.***Вихідні дані***

Тестовий пакет – MIRFlickr-20k (https://press.liacs.nl/mirflickr/#sec\_download)

Вибірка зображень – 250 зображень;

Формування вибірки зображень – псевдовипадкове, з використанням генератора Мерсена (стартове значення співпадає з номером студента в загальному списку групи) за модулем кількості зображень в тестовому пакеті.

***Лабораторна робота №2***

1. Сформувати тестову вибірку зображень з вихідного пакета;
2. Для кожного каналу кольору кожного зображення з тестового пакета обчислити наступні характеристики:
   1. Математичне сподівання і дисперсію;
   2. Коефіцієнти асиметрії та ексцесу (нормалізований);
3. Побудувати вектори параметрів зображень, що складаються з:
   1. Математичних очікувань значень яскравості для кожного каналу кольору;
   2. Математичних очікувань і дисперсії значень яскравості для кожного каналу кольору;
   3. Математичних очікувань, дисперсії і коефіцієнта асиметрії значень яскравості для кожного каналу кольору;
   4. Математичних очікувань, дисперсії, коефіцієнтів асиметрії та ексцесу значень яскравості для кожного каналу кольору;
4. Побудувати гаусові моделі зображень з використанням розрахованих раніше параметрів.
5. Провести декомпозицію кожного каналу кольору кожного зображення з застосуванням методу головних компонент (PCA):
   1. Варіюючи кількість компонент, провести реконструкцію окремих каналів кольору зображень (від компонент з найбільшою енергією поступово переходячи до компонентів з мінімальною енергією).
   2. Побудувати залежність помилки відновлення (середнє відхилення вихідного зображення відреконструйованого, MSE) від кількості використаних компонент.
6. Провести моделювання окремих каналів кольору зображень з використанням марковських ланцюгів:
   1. 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 однієї марковської ланцюга для одного з каналів кольору тестового зображення;

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

**Хід роботи**

**1. Формування тестової вибірки зображеннь**

Лабораторна робота буде виконана мовою Python за допомогою блокового інтерпритатора Jupyter. В данній лабораторній роботі використовувались наступні бібліотеки:

* Scipy
* Matplotlib.pyplot
* Random
* Os
* Numpy
* Pandas
* PIL
* Networkx
* Time
* Np
* Seaborn
* Fbpca

path = '/users/yuliachornenko/downloads/mirflickr/im'

folder = os.path.join(os.getcwd(), 'mirflickr/')

images\_sample = []

**for** i **in** random.sample(range(7,25001),250):

images\_sample.append(path + str(i) + '.jpg')

columns = ['num',

'channel',

'matspod',

'disp',

'skew',

'kurt',

'intens',

'data']

**def** do\_open(file):

im = Image.open(file)

**return** np.array(im)

**def** get\_sample():

t = time.time()

np\_arrays = map(do\_open, images\_sample)

array = [i **for** i **in** np\_arrays]

**return** array

**def** get\_df(image):

values = []

**for** i **in** range(len(image)):

temp = image[i]

intens = (temp.sum(axis=0).sum(axis=0)/(temp.shape[0]\*temp.shape[1]))/np.linalg.norm(temp.sum(axis=0).sum(axis=0)/(temp.shape[0]\*temp.shape[1]), ord=**None**)

**for** channel **in** range(3):

channel\_img = image[i][:,:,channel]

values.append(list((random.sample(range(6,25001),250)[i], channel,

np.mean(channel\_img),np.var(channel\_img),

skew(channel\_img, **None**), kurtosis(channel\_img, **None**),

intens[channel],

channel\_img)))

**return** pd.DataFrame(values, columns=columns)

* + 1. **Побудувати вектори параметрів зображень:**

**Побудувати гаусові моделі зображень з використанням розрахованих раніше параметрів.**

**def** get\_vectors():

ss = MinMaxScaler()

df1 = df[['matspod','disp','skew','kurt','intens']]

params = df1.columns

ss.fit(df1)

dfscal = pd.DataFrame(ss.transform(df1), columns=params)

std = np.array(dfscal.std())

cov = np.cov(dfscal.T)

diag = np.eye(5,5) \* np.array(dfscal.var())

gauss\_samples = np.random.multivariate\_normal(std, diag, 750)

plt.figure(5)

plt.subplot('521')

plt.plot(gauss\_samples[:, 0], dfscal['matspod'], '.')

plt.subplot('522')

plt.plot(gauss\_samples[:, 1], dfscal['disp'], '.')

plt.subplot('523')

plt.plot(gauss\_samples[:, 2], dfscal['skew'], '.')

plt.subplot('524')

plt.plot(gauss\_samples[:, 3], dfscal['kurt'], '.')

plt.subplot('525')

plt.plot(gauss\_samples[:, 4], dfscal['intens'], '.')

plt.show()

print('normal')

pgauss = normal.pdf(gauss\_samples, std, diag)

plt.figure(5)

plt.subplot('521')

plt.plot(gauss\_samples[:, 0], pgauss, '.')

plt.subplot('522')

plt.plot(gauss\_samples[:, 1], pgauss, '.')

plt.subplot('523')

plt.plot(gauss\_samples[:, 2], pgauss, '.')

plt.subplot('524')

plt.plot(gauss\_samples[:, 3], pgauss, '.')

plt.subplot('525')

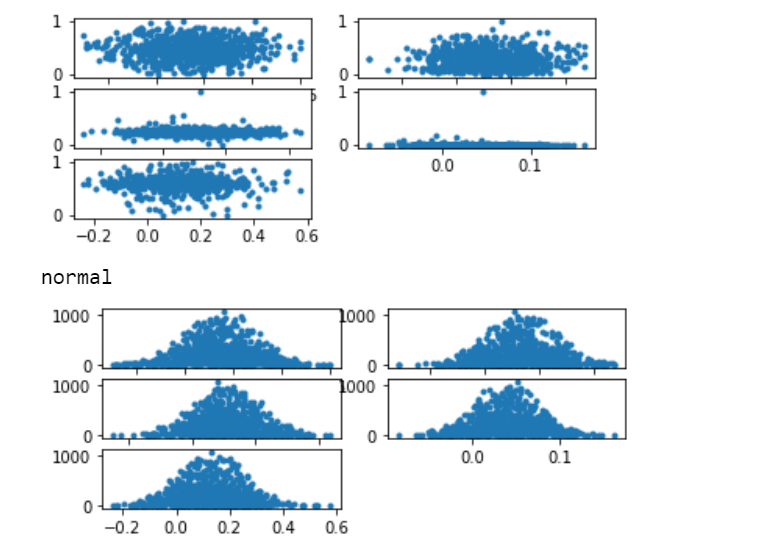
plt.plot(gauss\_samples[:, 4], pgauss, '.')

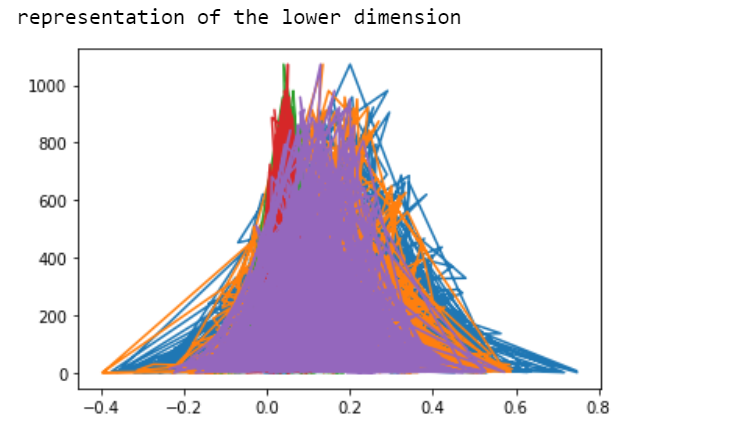
plt.show()

print('representation of the lower dimension')

plt.plot(gauss\_samples, pgauss)

plt.show()





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

Даний процес проходть для визначеної кількості компонентів n, яке ми також передаватимемо як параметр.

**def** get\_PCA(images, power):

model = PCA(power)

model.fit(images)

tr\_images = model.transform(images)

**return** model,tr\_images

**def** mse(image1, image2):

err = np.sum((image1.astype('float') - image2.astype('float')) \*\* 2)

err /= float(image2.shape[0] \* image2.shape[1])

**return** err

**def** my\_function(components, my\_list, fshape, sshape):

models = []

comperessed = []

mse\_logs = []

mse\_mean = []

**for** i **in** components:

model1, comp\_img = get\_PCA(my\_list,i)

models.append(model1)

comperessed.append(comp\_img)

reconstr = model1.inverse\_transform(comp\_img)

mse\_log = np.array(list(map(mse,reconstr.reshape((300, fshape, sshape)),

my\_list.reshape((300, fshape, sshape)))))

mse\_logs.append(mse\_log)

mse\_mean.append(np.mean(mse\_log))

**return** components, np.array(mse\_mean), np.array(mse\_log)

Отримаємо наступний результат:



Рисунок 1 **–** відновлені фото з різною кількістю компонентів

Опрацювавши результати можна зробити висновок, що зі збільшенням кількості компонентів росте якість відновлення зображення.

Далі виконаємо функцію для багатьох кроків та поріняємо початкове фото з відновленим за допомогою функції середньої квадратичної похибки.

print('MSE:' , np.mean(mse\_log))

cumsum = np.cumsum(model.explained\_variance\_ratio\_)\*100

plt.figure()

plt.plot([n **for** n **in** range(len(cumsum))], cumsum)

plt.ylabel('variance')

plt.xlabel('components')

plt.axhline(y = 90, linestyle='--')

index = np.random.choice(750, 300)

my\_list = standart[index]

my\_list.shape

components = np.arange(5, 150, 30)

array = my\_function(components, my\_list, fshape, sshape)

plt.figure(figsize=(5, 5))

plt.plot(array[0], array[1])

plt.ylabel('error')

plt.xlabel('components')

plt.show()

В результаті отримуеємо графік:

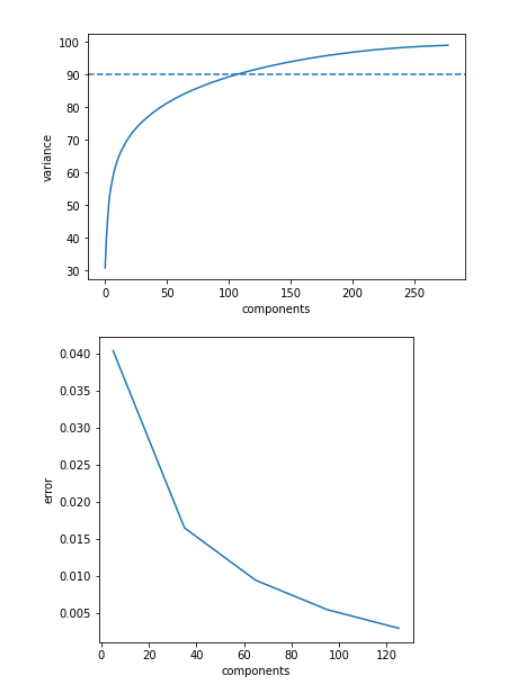


Рисунок 2 – залежність MSE відновлених фото від кількості компонент

З даного графіка видно що залежність має експоненціальний характер та похибка дуже значно зменшується при збільшенні компонент.

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

1. Для кожного каналу кольору кожного зображення розрахувати стохастичну матрицю марковського ланцюга першого і другого порядків;

**def** get\_transition(array, order):

\_shape = (max(array) + 1,) \* (order + 1)

M = np.zeros(\_shape)

**for** \_ind **in** zip(\*[array[\_x:] **for** \_x **in** range(order + 1)]):

M[\_ind] += 1

**return** np.array(M)

**def** get\_tran(images, order):

result = []

**for** i **in** images:

TM\_vert = []

TM\_hor = []

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

hor = np.array(list(reversed(i[x,:])))

hor\_tran = get\_transition(hor,order)

TM\_hor.append(hor\_tran)

**for** y **in** range(i.shape[1]):

vert = np.array(list(reversed(i[:,y])))

vert\_tran = get\_transition(vert,order)

TM\_vert.append(vert\_tran)

TM\_vert = np.array(TM\_vert)

TM\_hor = np.array(TM\_hor)

result.append(list((TM\_vert,TM\_hor)))

**return** np.array(result)

**def** get\_matrix\_and\_graph():

first\_order = get\_tran(df['data'][:10], 1)

second\_order = get\_tran(df['data'][:10], 2)

states = [tuple(i) **for** i **in** list(reversed(df['data'][:10][:,0]))]

labels= {}

edge\_labels= {}

**for** i, origin **in** enumerate(states):

**for** j, dest **in** enumerate(states):

rate = get\_tran(states,1)[i][j]

**if** rate > 0:

(nx.MultiDiGraph()).add\_edge(origin, dest,

weight=rate,

label="**{:.02f}**".format(rate))

edge\_labels[(origin\_state, destination\_state)] = label="**{:.02f}**".format(rate)

plt.figure(figsize=(14,7))

node\_size = 200

nx.draw\_networkx\_edges(nx.MultiDiGraph(), {state:list(state) **for** state **in** states}, width=1.0, alpha=0.5)

nx.draw\_networkx\_labels(nx.MultiDiGraph(), {state:list(state) **for** state **in** states}, font\_weight=2)

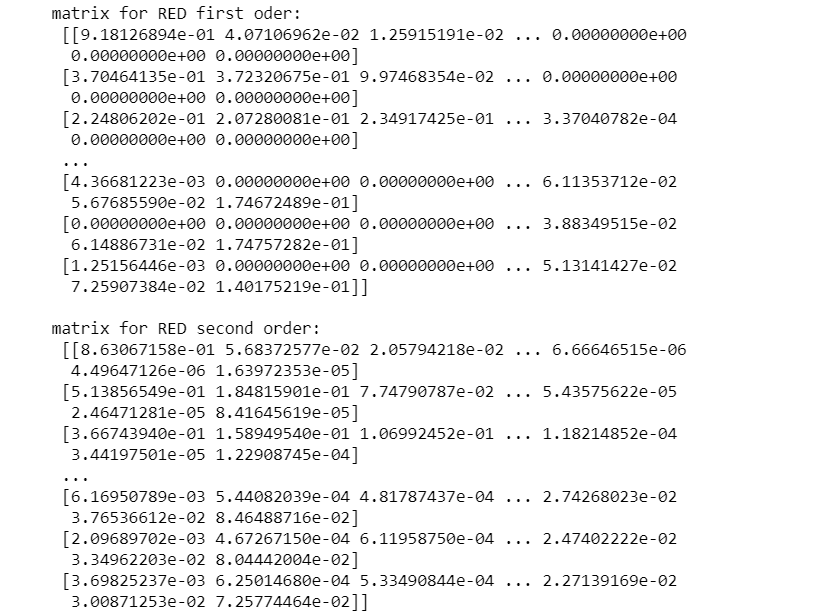
nx.draw\_networkx\_edge\_labels(nx.MultiDiGraph(), {state:list(state) **for** state **in** states}, edge\_labels)

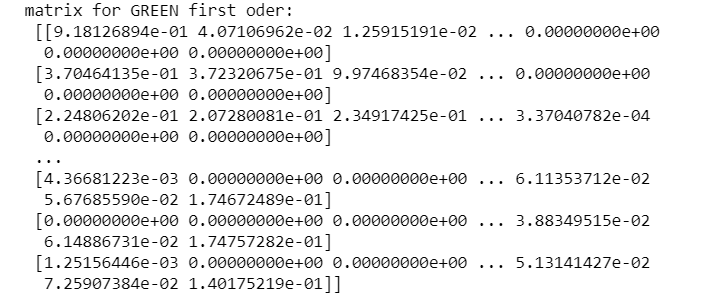
plt.axis('off')

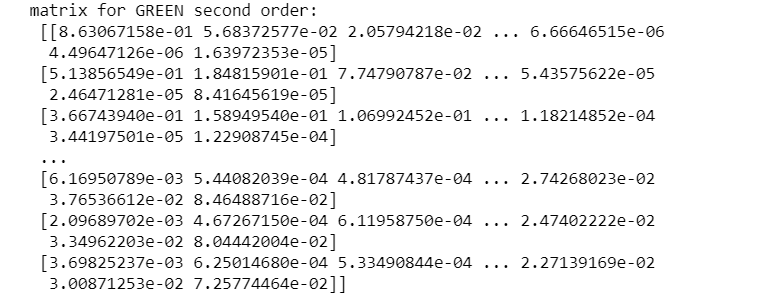
n = np.array(get\_transition(np.array(list(reversed(df['data'][34][:, 5]))),1))

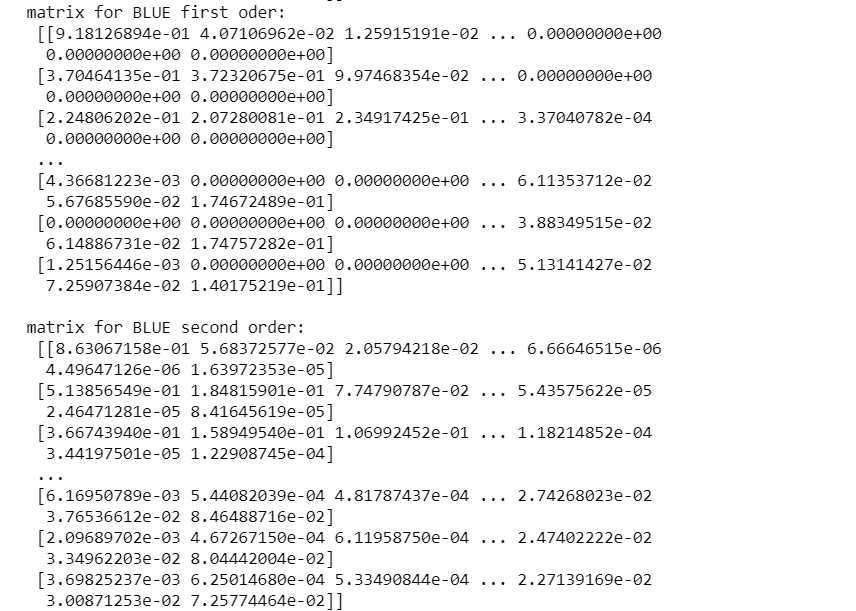
**for** row **in** n: print(' '.join('**{0:.2f}**'.format(x) **for** x **in** row))

Отримаємо наступні матриці:









Побудуємо графічне представлення ланцюга

**def** get\_final\_graph():

test\_list = [red, green, blue]

col = ['RED', 'GREEN', 'BLUE']

c = 0

**for** color **in** test\_list:

matrix1 = np.zeros(shape=(256, 256))

array = color.flatten('F')

prev\_color = array[0]

**for** i **in** range(len(array) - 1):

matrix1[array[i]][array[i + 1]] += 1

matrix = matrix1[0] / sum(matrix1[0])

**for** i **in** range(1, 256):

matrix = np.vstack((matrix, matrix1[i] / sum(matrix1[i])))

print(f'matrix for **{col[c]}** first oder:**\n**', matrix)

count = 0

**for** k **in** range(6):

reg=matrix\_power(matrix, k)

**for** i **in** range(256):

**if** reg[i][i] == 0:

count += 1

**if** count == 0:

print('regularity properties are executed and recurrence properties are executed')

**else**:

print('property of regularity and recurrence are not satisfied')

count=0

**for** i **in** range(256):

**for** j **in** range(256):

**if** (matrix[i][i] == 1) **or** (matrix[i][j] == 0):

count +=1

**if** count == 0:

print('irreversibility property is satisfied')

**else**:

print('irreversibility property is not satisfied')

print(f'**\n**matrix for **{col[c]}** second order:**\n**', np.linalg.matrix\_power(matrix, 2))

count = 0

**for** k **in** range(6):

reg= matrix\_power(np.linalg.matrix\_power(matrix, 2), k)

**for** i **in** range(256):

**if** reg[i][i] == 0:

count += 1

**if** count == 0:

print('regularity properties are executed and recurrence properties are executed')

**else**:

print('property of regularity and recurrence are not satisfied')

count = 0

**for** i **in** range(256):

**for** j **in** range(256):

**if** (np.linalg.matrix\_power(matrix, 2)[i][i] == 1) **or** (np.linalg.matrix\_power(matrix, 2)[i][j] == 0):

count +=1

**if** count == 0:

print('irreversibility property is satisfied')

**else**:

print('irreversibility property is not satisfied')

c +=1

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

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

dataframe = pd.DataFrame(data, index=index, columns=index)

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

g = nx.from\_pandas\_adjacency(dataframe)

nx.draw(g, with\_labels=**True**)

plt.show()

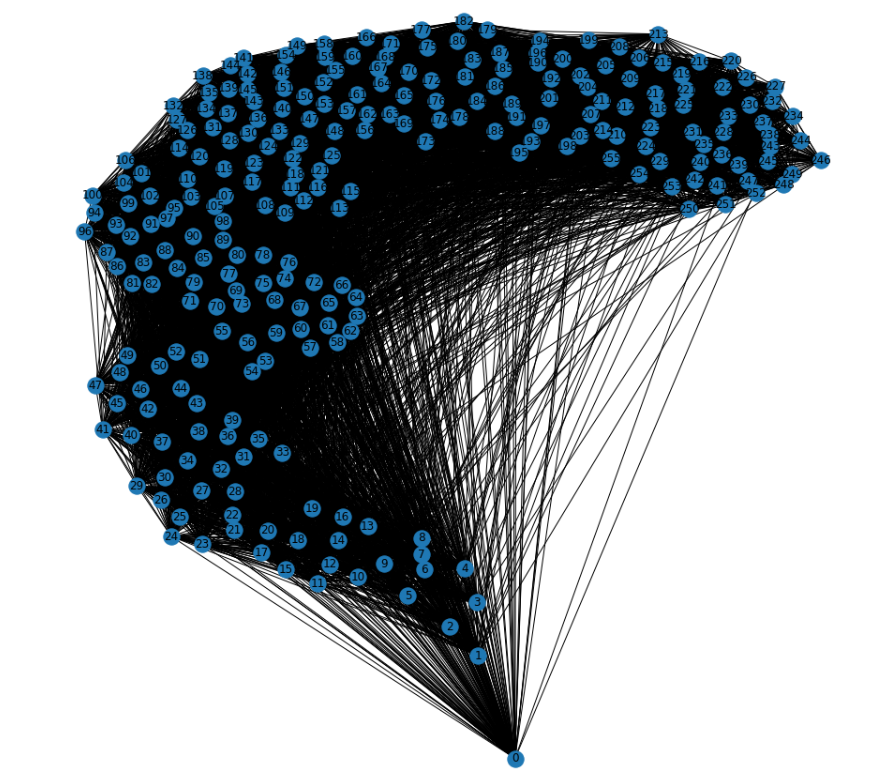


Рисунок 3 - Вигляд марківського ланцюга

**ВИСНОВКИ**

В даній лабораторній роботі було проаналізовано вибірку з 250 зображень датасету mirflickr-20k.

Побудувано вектори параметрів зображень 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 (рис. 1).

Зібравши дані, було побудовано графік залежності середньої квадратичної похибки відновлених дображень від кількості компонентів (рис. 2).

Було помічено експоненціальну залежність, що свідчить про значні зміни при невеликих кількостях компонентів (< 20) та майже непомітні при великих значеннях (> 100).

Проведуно моделювання окремих каналів кольору зображень з використанням марковських ланцюгів, та сформовано стохастичні матриці за різними типами обходів. За даними було побуловано графічну можель марківського ланцюга.

З (рис. 3) видно скупчення схожих яскравостей та плавний перехід від великих значень до малих. Це говорить про відсутність різких зміщень в кольоровій гамі пікселів зображень.