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 №9

Керівник: Виконав:

Прогонов Дмитро Олександрович студент 5 курсу групи ФЕ-91мп

Захищено з оцінкою Проценко Дмитро Олександрович

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

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

дата, підпис

Київ – 2020 р.Завдання на лабораторну роботу:

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:
4. Математичних очікувань значень яскравості для кожного каналу кольору;
5. Математичних очікувань і дисперсії значень яскравості для кожного каналу кольору;
6. Математичних очікувань, дисперсії і коефіцієнта асиметрії значень яскравості для кожного каналу кольору;
7. Математичних очікувань, дисперсії, коефіцієнтів асиметрії та ексцесу значень яскравості для кожного каналу кольору;
8. Побудувати гаусові моделі зображень з використанням розрахованих раніше параметрів.
9. 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.
10. Провести моделювання окремих каналів кольору зображень з використанням марковських ланцюгів:
11. 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;
12. 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
* Os
* Numpy
* Pandas
* Sklearn
* Networkx

Тестова вибірка сформована із псевдовипадкового вибору 250 кольорових зображень із пакету mirflick.

Приклад коду для вибірки зображень:

*# load all images in a directory*

random.seed(9)

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

loaded\_images = list()

**for** i **in** range(250):

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

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

loaded\_images.append(img\_data)

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

Тепер отриманий масив зображень буде знаходитись в loaded\_images в виді двомірного масиву з трьома значеннями яскравості в кожрній комірці.

Далі сформуємо матрицю для збору статистичних даних про зобьраження. Для цього створимо двомірний Numpy масив на три рядки для кожного каналу кольору та на 256 стовпчиків, що відповідатиме кількості пікселів відповідної яскравості.

values = np.zeros((3, 256))

index = 0

**for** image **in** loaded\_images:

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

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

values[0][image[i][j][0]] += 1

values[1][image[i][j][1]] += 1

values[2][image[i][j][2]] += 1

index += 1

**if** (index % 10) == 0:

print('> processed **%i** files' % (index))

**2. Обчислення характеристик для кожного каналу зображень**

1. Математичне сподівання і дисперсія

Розрахунки будуть проводитись за наступними формулами:



(1),



(2),

Де (1) - математичне очікування, а (2) – дисперсія, xi - значення яскравості, pi – ймовірність її появи. pi можна знайти як кількість пікселів даної яскравості поділену на всю кількість пікселів

*#GREEN*

sum\_value = sum(values[GREEN])

M\_green = 0

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

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

M\_green += p \* index

D\_green = 0

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

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

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

print("Green: expected value (math. ochikuvanya) : **{0:.2f}**, dispersion : **{1:.2f}**"\

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

Таким чином отримуэмо значення Мат. очікування та дисперсії:

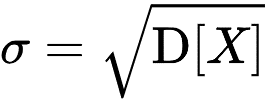
Red: expected value (math. ochikuvanya) : 109.15, dispersion : 6344.77

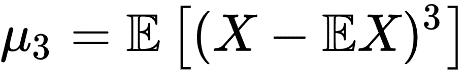
Green: expected value (math. ochikuvanya) : 101.70, dispersion : 5744.94

Blue: expected value (math. ochikuvanya) : 93.15, dispersion : 6225.48

1. Коефіцієнти асиметрії та ексцесу

Використаємо відповідні формули коефіцієнту асиметрії:





**def** E\_operator(arr\_values, M, power):

sum\_val = sum(arr\_values)

ans = 0

**for** index **in** range(len(arr\_values)):

p = (arr\_values[index] / sum\_val)

ans += p \* ((index - M) \*\* power)

**return** ans

*#asymmetry, ekscess*

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))

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

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

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

print('Red: asymmetry - **{0:.3f}**, ekscess - **{1:.3f}**'\

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

print('Green: asymmetry - **{0:.3f}**, ekscess - **{1:.3f}**'\

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

print('Blue: asymmetry - **{0:.3f}**, ekscess - **{1:.3f}**'\

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

Отримаємо:

Red: asymmetry - 0.283, ekscess - -1.155

Green: asymmetry - 0.366, ekscess - -1.028

Blue: asymmetry - 0.544, ekscess - -0.956

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

Сформуємо вектор всіх потрібних нам значень,та використаемо окремі частини для розрахунку:

Vector\_A = np.array([np.array([M\_red, D\_red, Asym\_red, Ekscess\_red]),

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

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

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

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

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)

Отримаємо масив Vector\_All\_DATA, що міститиме всі потрібні нам дані всіх трьох каналів.

* + 1. **Побудувати гаусові моделі зображень з використанням розрахованих раніше параметрів.**

Сформуємо гаусові моделі. Отримаємо вектори параметрів зображень, що складаються з:

1. Математичних очікувань значень яскравості для кожного каналу кольору;

expected value + colors:

[11712.77091714]

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

expected value + dispersion + colors:

[[6345.63611094 5264.20122769]

[5264.20122769 5745.6493194 ]]

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

expected value + dispersion + asymmetry + colors:

[[6345.63611094 5264.20122769 4409.70467577]

[5264.20122769 5745.6493194 5340.86506565]

[4409.70467577 5340.86506565 6226.31387514]]

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

expected value + dispersion + asymmetry + ekscess + colors:

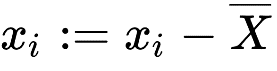
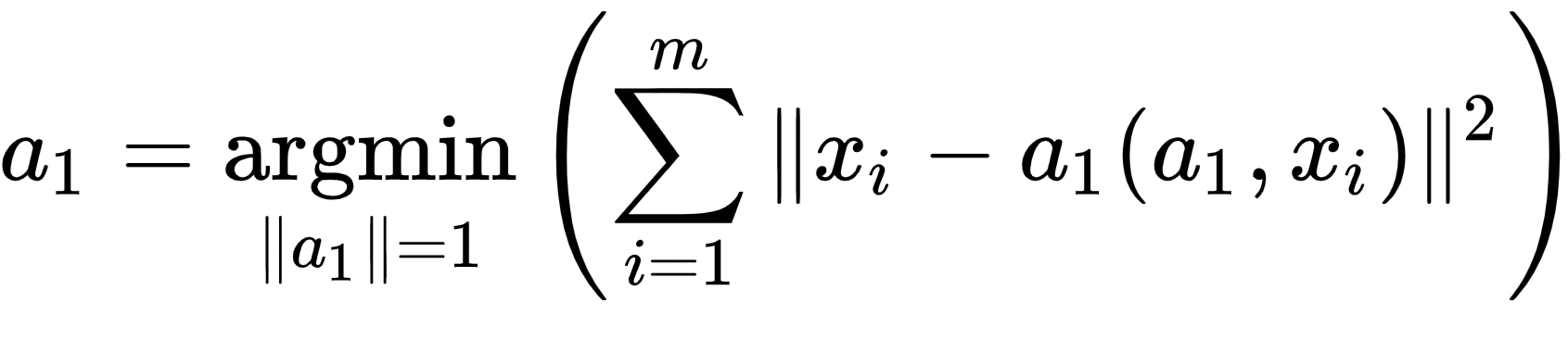
[[6345.63611094 5264.20122769 4409.70467577]

[5264.20122769 5745.6493194 5340.86506565]

[4409.70467577 5340.86506565 6226.31387514]]

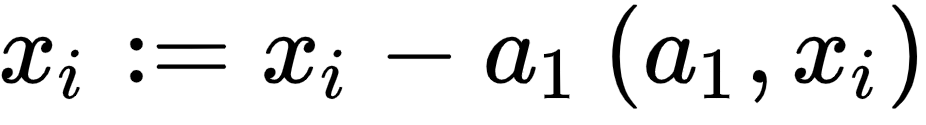
* + 1. **Провести декомпозицію кожного каналу кольору кожного зображення з застосуванням методу головних компонент (PCA)**

Для проведення декомпозиції каналів кольору сформуємо функцію що працюватиме по методу головних компонентів (PCA). Складемо наступний алгоритм:

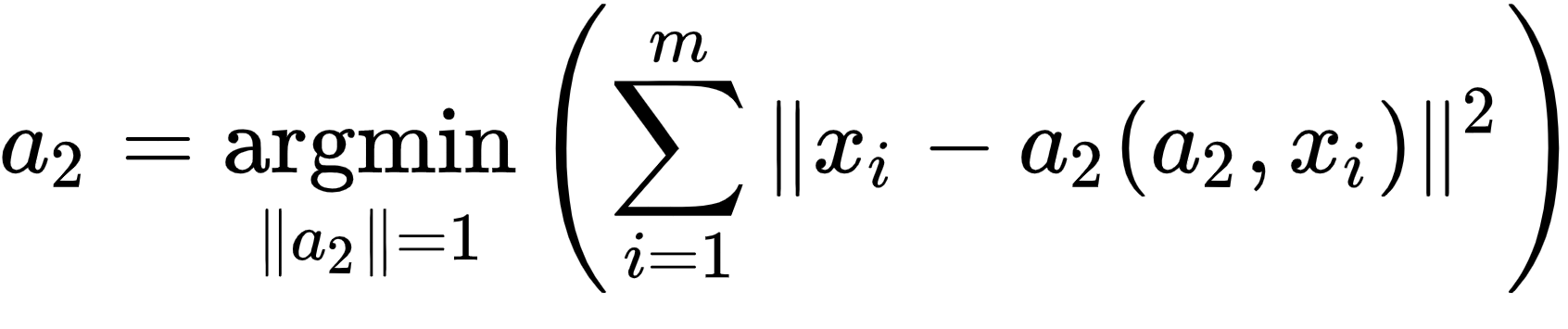
1. Централізувати дані (відніманням середнього):
2. Знайти першу головну компоненту як рішення задачі: 

якщо рішення не одне, то здійснюється вибір одного з них.

1. З даних відняти проекцію на першу головну компоненту:



1. Відшукати другу головну компоненту як рішення задачі:



якщо рішення не одне, то здійснюється вибір одного з них.

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

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[2])

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, 30), PCA\_2d(a\_g, 30), PCA\_2d(a\_b, 30)

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

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

plt.title('30 components')

plt.imshow(recon\_color\_img)

a\_r\_recon, a\_g\_recon, a\_b\_recon = PCA\_2d(a\_r, 80), PCA\_2d(a\_g, 80), PCA\_2d(a\_b, 80)

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

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

plt.title('80 components')

plt.imshow(recon\_color\_img)

plt.show()

Отримаємо наступний результат:

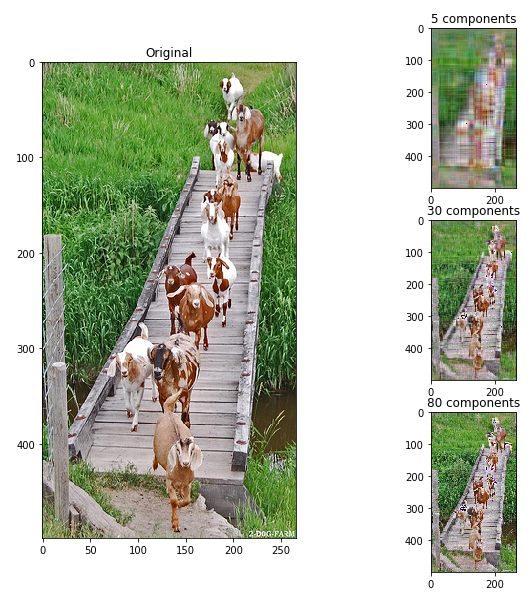


Рисунок 1 **–** відновлені фото з різною кількістю компонентів

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.

def mse(imageA, imageB):

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()

В результаті отримуеємо графік:

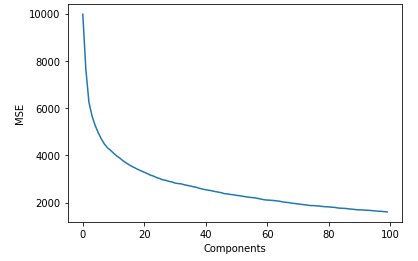


Рисунок 2 – залежність MSE відновлених фото від кількості компонент

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

* + 1. **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;

Для побудови марківського ланцюга сформуємо матрицю 256 на 256 та пройшови по всіх пікселях картинки запишемо кількість переходів між ними. При цьому представимо два типи обходу матриць “C-type” та “Fortran-type”

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))

Отримаємо наступні матриці:

Red matrix 1st oder:

[[0.00294985 0.00589971 0. ... 0. 0. 0. ]

[0.01282051 0. 0. ... 0. 0. 0. ]

[0.01041667 0. 0. ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.11961722 0.16746411 0.215311 ]

[0. 0. 0. ... 0.09570957 0.1980198 0.17821782]

[0. 0. 0. ... 0.08163265 0.09105181 0.22291994]]

Red matrix 2nd order:

[[5.09338420e-03 8.97927655e-04 1.51707202e-03 ... 5.23097312e-05

1.61268123e-04 4.14854386e-04]

[5.24400288e-03 1.34305199e-03 1.56642213e-03 ... 3.83645833e-05

2.70519507e-04 7.93485752e-04]

[5.79240434e-03 9.64267146e-04 1.34309489e-03 ... 2.95065792e-05

2.15990898e-04 4.00926463e-04]

...

[1.38859205e-04 2.60274372e-05 3.25454597e-05 ... 6.38340351e-02

9.72716035e-02 1.65832111e-01]

[9.02749673e-05 2.22686400e-05 4.49283385e-05 ... 6.41013847e-02

1.00274964e-01 1.69159215e-01]

[2.91141810e-04 5.34812378e-05 7.04672152e-05 ... 5.42491944e-02

7.50325788e-02 1.47500026e-01]]

Red matrix 2-nd type 1st oder:

[[0.04424779 0.01769912 0.02359882 ... 0. 0. 0. ]

[0.07692308 0.01282051 0. ... 0. 0. 0. ]

[0.0625 0. 0.03125 ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.13397129 0.11961722 0.32535885]

[0. 0. 0. ... 0.11551155 0.2310231 0.23432343]

[0. 0. 0. ... 0.08791209 0.14285714 0.28728414]]

Red matrix 2-nd type 2nd order:

[[2.07095975e-02 4.87635147e-03 5.42644297e-03 ... 2.49044384e-05

1.55330638e-05 1.18453270e-04]

[2.23105247e-02 5.57449683e-03 6.31245523e-03 ... 0.00000000e+00

3.88256353e-05 1.77039001e-04]

[1.99057683e-02 5.76324114e-03 8.16107518e-03 ... 1.31357713e-05

1.26877791e-05 3.01010265e-05]

...

[5.64232193e-06 0.00000000e+00 0.00000000e+00 ... 8.05171238e-02

1.14387935e-01 2.21455687e-01]

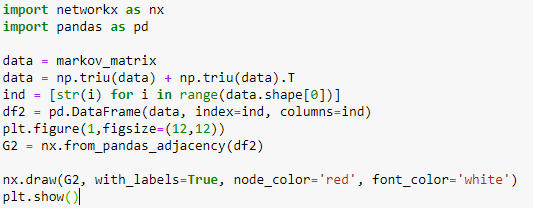
[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 8.31951218e-02

1.25622484e-01 2.11623785e-01]

[1.53729633e-05 0.00000000e+00 0.00000000e+00 ... 7.43198297e-02

1.11006195e-01 2.03312079e-01]]

Для графічного представлення ланцюга використаємо бібліотеку Networkx. Пікселі будуть вузлами, а переходи між ними – з’єднаннями.



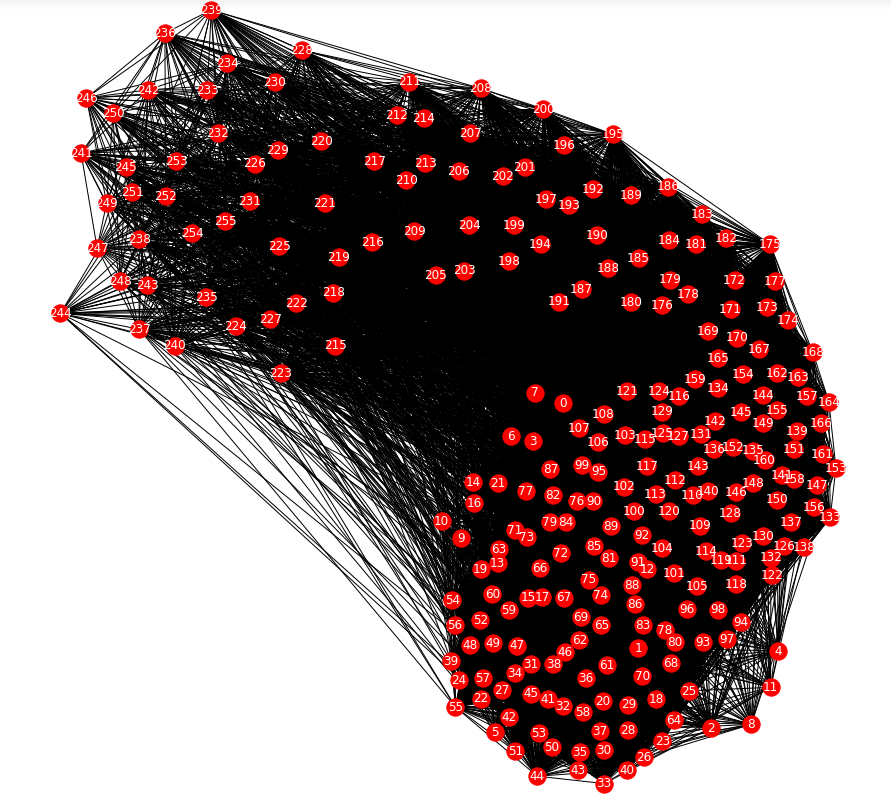


Рисунок 3 - Вигляд марківського ланцюга

**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 – 109.15 для чевоного каналу, 101.70 для зеленого і 93.15 для синьогоканалу відповідно і дисперсію 6344.77, 5744.94 та 6225.48. Також знайдено коефіцієнти асиметрії та ексцесу:

Червоний: асиметрія - 0.283, ексцес -1.155

Зелений: асиметрія - 0.366, ексцес -1.028

Синій: асиметрія - 0.544, ексцес - 0.956

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