Міністерство освіти і науки України

НТУУ «Київський політехнічний інститут ім. Ігоря Сікорського»

Фізико-технічний інститут

Лабораторна робота №2

З дисципліни

«Автоматизація обробки ІзОД»

Варіант 5

**Виконав:**

Студент 5 курсу ФТІ

групи ФЕ-91мп

Мазурок В. О.

**Перевірив:**

Прогонов Д. О.

Київ-2020

**І. Підготовка**

**Вхідні 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.

**Завдання**

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.

**ІI. Хід роботи**

Роботу виконуватимемо мовою Python за допомогою блокового інтерпритатора Jupyter. Також в роботі будкть використані такі бібліотеки як:

* Os
* Matplotlib
* Numpy
* Scipy
* Pandas
* Sklearn
* Networkx

1. **Формування тестової вибірки зображень з вихідного пакета**

Для цього скористаємося функцією numpy.random.choices() що обирає випадкові числа з переданого масива за допомогою генератора Мерсена. Також задамо початкове значення варіату за допомогою функції numpy.random.RandomState()

np.random.RandomState(5)

random\_indexes = np.random.choice(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))

Після цього ориманий масив зображень буде знаходитись в loaded\_images в виді двомірного масиву з трьома значеннями яскравості в кожій комірці.

Тепер сформуємо матрицю для збору статистичних даних. 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 images' % (index))

1. **Знаходження статистичних даних**
2. **Математичне сподівання і дисперсія**

Для знаходження скористаємось відповідними формулами:



- Математичне очікування

- Дисперсія

Де xi наше значення яскравості, а pi – ймовірність її появи. pi можна знайти як кількість пікселів даної яскравості поділену на всю кількуість пікселів

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:\tmat ochikuvannya - {0:.2f},\tdispersiya - {1:.2f}".format(M\_red, D\_red))

Отримуємо:

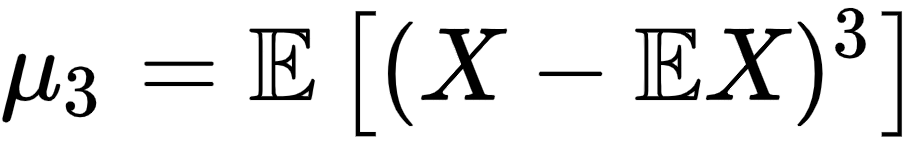
**Red: mat ochikuvannya - 108.70, dispersiya - 6163.60**

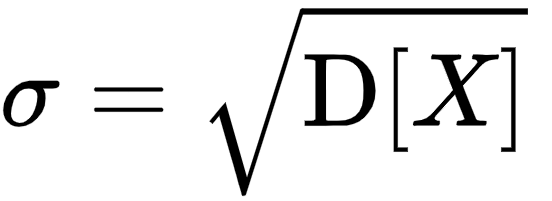
**Green: mat ochikuvannya - 99.80, dispersiya - 5495.78**

**Blue: mat ochikuvannya - 89.87, dispersiya - 5847.97**

**b. Коефіцієнти асиметрії та ексцесу**

Для цього також використаємо відповідні формули коефіцієнту асиметрії:





Для спрощення напишемо загальну формулу для оператора Е, що приймає степінь та мат очікування.

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)

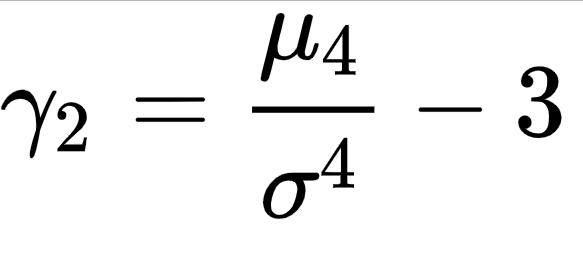
ans += p \* ((index - M) \*\* power)

return ans

Тепер формула отримає вигляд:

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

Та формули коефіцієнту ексцесу:



Ekscess\_red = E\_operator(values[RED], M\_red, 4) / (D\_red \*\* 2) - 3

Отримуємо:

**Red: Asimmetriya - 0.252, Ekscess - -1.174**

**Green: Asimmetriya - 0.404, Ekscess - -0.958**

**Blue: Asimmetriya - 0.613, Ekscess - -0.826**

1. **Побудувати вектори параметрів зображень:**

Для цього сформуємо вектор всіх потрибних нам значень, та для розрахунків будемо брати окремі чатини готових даних:

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)

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

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

Тепер сформуємо гаусові моделі. Для першого випадку маємо одномірний варіант лише з мат очікуванням:

#a Mat ochikuvannya

P\_x1 = np.random.normal(Vector\_All\_DATA[0, 4:])

print("Mat ochikuvannya + colors:\n" + str(P\_x1))

**Mat ochikuvannya + colors:**

**6163.5977630649895**

Наступні вектори є дво- трьох- та чотирьохвимірними варіантами матриці коваріації, представимо їх:

#b Mat ochikuvannya and dispersion

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

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

**Mat ochikuvannya + dispersion + colors:**

**[[6164.41341013 5127.366324 ]**

**[5127.366324 5496.42737747]]**

**Mat ochikuvannya + dispersion + asymetry + colors:**

**[[6164.41341013 5127.366324 4525.58191176]**

**[5127.366324 5496.42737747 5160.35020174]**

**[4525.58191176 5160.35020174 5848.70350154]]**

**Mat ochikuvannya + dispersion + asymetry + ekscess + colors:**

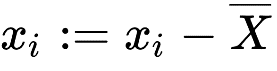
**[[6164.41341013 5127.366324 4525.58191176 6164.41341013]**

**[5127.366324 5496.42737747 5160.35020174 5127.366324 ]**

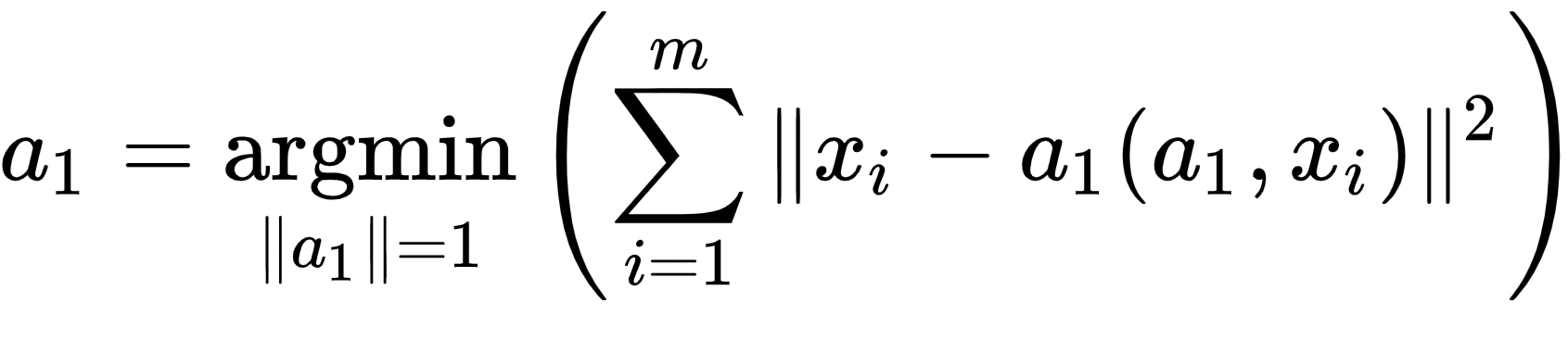
**[4525.58191176 5160.35020174 5848.70350154 4525.58191176]**

**[6164.41341013 5127.366324 4525.58191176 6164.41341013]]**

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

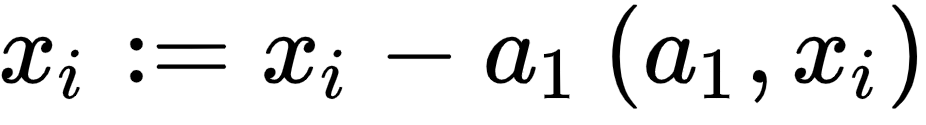
Для проведення декомпозиції каналів кольору сформуємо функцію що працюваниме по методу головних компонент (PCA). Алгоритм даної функції буде наступним:

1. Централізувати дані (відніманням середнього)

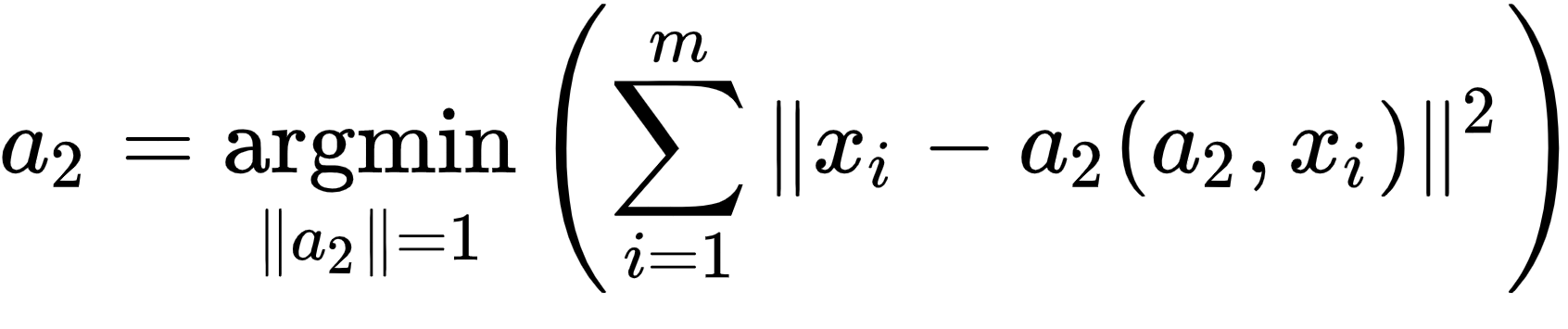
2. Знайти першу головну компоненту як рішення задачі: 

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

3. З даних відняти проекція на першу головну компоненту:



4. Відшукати другу головну компоненту як рішення задачі:



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

Даний процес проходть для визначеної кількості компонент 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

Передавши переметри тестової фотографії отримаємо наступний результат:

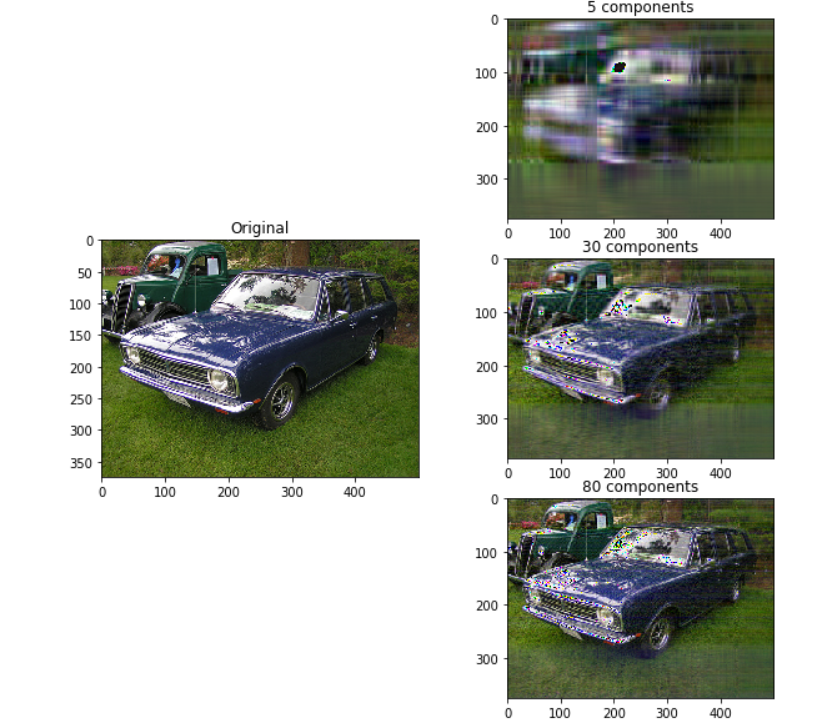


Рисунок 1.1. **–** відновлені фото з різної кількості компонент

Як бачимо, зі збільшенням кількості компонентів росте якість відновлення зображення. Також можемо спостерігати «артефакти» відновлення на частинах з різким кольоровим зсувом та контрастом. Це спричинено високою кількістю інформації, що потрбна для відновлення таких частин.

Після цього фиконаємо 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()

Отримаємо наступний графік:

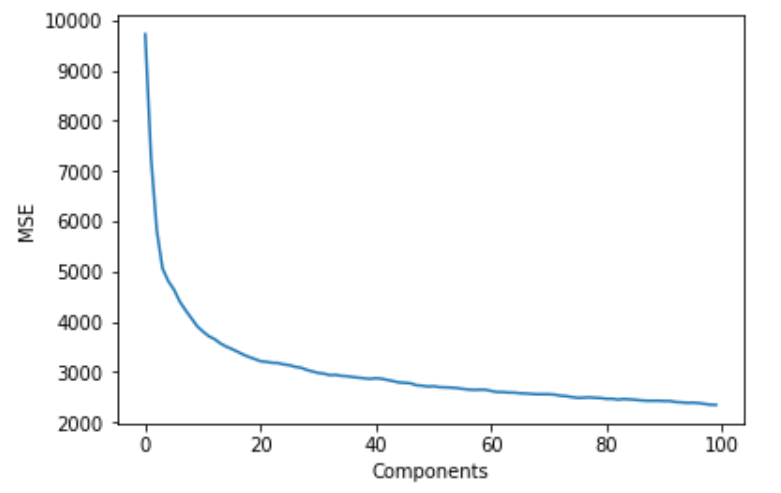


Рисунок 1.2. – залежність MSE відновлених фото від кількості компонент

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**
2. 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:**

**[[0.17527675 0.07933579 0.04059041 ... 0. 0. 0. ]**

**[0.18604651 0.04186047 0.05581395 ... 0. 0. 0. ]**

**[0.11848341 0.02843602 0.06635071 ... 0. 0. 0. ]**

**...**

**[0. 0. 0. ... 0.08095238 0.12619048 0.27380952]**

**[0. 0. 0. ... 0.06970509 0.16621984 0.26809651]**

**[0. 0. 0. ... 0.07055394 0.11137026 0.33411079]]**

**Red matrix 2nd order:**

**[[6.17423992e-02 2.44156378e-02 2.07109672e-02 ... 2.41688769e-04**

**5.23919999e-04 1.46420316e-03]**

**[6.25854970e-02 2.66901558e-02 2.24826477e-02 ... 1.53634526e-04**

**5.14595213e-04 1.10242764e-03]**

**[5.09497341e-02 2.23700350e-02 2.04517715e-02 ... 3.71592027e-04**

**4.73881468e-04 1.50485782e-03]**

**...**

**[3.21301636e-04 2.99768628e-05 3.05413219e-05 ... 5.32813814e-02**

**9.07073346e-02 1.96470702e-01]**

**[1.20299765e-04 6.17242920e-05 3.30442652e-05 ... 5.62899469e-02**

**9.75271735e-02 2.03987630e-01]**

**[2.05912430e-04 1.10442939e-04 3.40639567e-05 ... 5.31138008e-02**

**8.97990839e-02 2.03656664e-01]]**

**Red matrix 2-nd type 1st oder:**

**[[0.05350554 0.03136531 0.01476015 ... 0. 0. 0. ]**

**[0.06511628 0.01860465 0.02325581 ... 0. 0. 0. ]**

**[0.04265403 0.02369668 0.04265403 ... 0. 0. 0. ]**

**...**

**[0. 0. 0. ... 0.07380952 0.11190476 0.22619048]**

**[0. 0. 0. ... 0.05898123 0.08579088 0.16487936]**

**[0. 0. 0. ... 0.05830904 0.07405248 0.1819242 ]]**

**Red matrix 2-nd type 2nd order:**

**[[0.01219736 0.00572942 0.00659111 ... 0.00066656 0.00156568 0.0039511 ]**

**[0.01307571 0.00691102 0.00797268 ... 0.00066142 0.00193495 0.00391466]**

**[0.01521409 0.00796804 0.0111424 ... 0.00045686 0.00094671 0.00221036]**

**...**

**[0.00083443 0.0003704 0.00029082 ... 0.03863306 0.06049128 0.12480949]**

**[0.0010203 0.0003571 0.00035109 ... 0.0328739 0.05553853 0.11191085]**

**[0.00104039 0.00046047 0.00037995 ... 0.03133446 0.0509181 0.10721584]]**

Для графічного представлення ланцюга використаємо бібліотеку networkx. Пікселі будуть вуздами а переходи між ними – з’єднаннями.

#Graph

import networkx as nx

import 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='skyblue', font\_color='red')

plt.show()

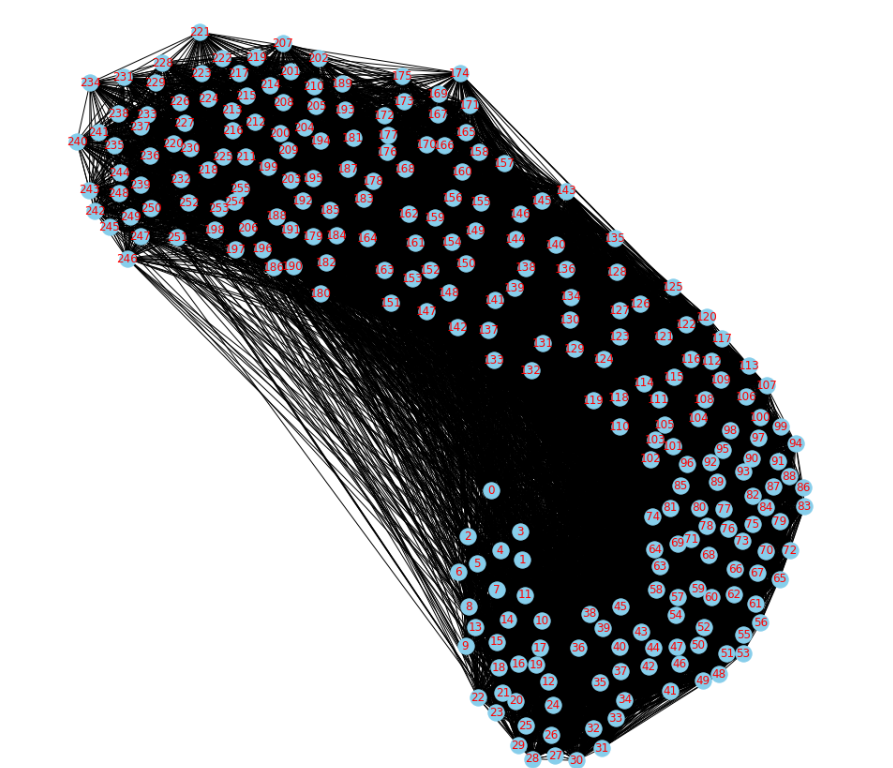


Рисунок 1.3 - Вигляд марківського ланцюга

**ІІІ. 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, 105 для зеленого і 96 для синьогоканалу відповідно і дисперсію 6024.15, 5724.37 та 5948.75.

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 побуловано графік залежності середньої квадратичної похибки відновлених дображень 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 проведуно моделювання окремих каналів кольору зображень з використанням марковських ланцюгів, та сформовано стохастичні матриці за різними типами обходів. Також за даними було побуловано графічну можель марківського ланцюга. З рисунку 1.3 видно скупчення схожих яскравостей та плавний перехід від великих значень до малих. Це говорить про відсутність різких зміщень в кольоровій гамі пікселів зображень.