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.

**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
* Seaborn
* CV2
* networkx

1.Завантаження вибірки зображень відносно варіанту

**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(7)

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('E:/Labs/mirflickr/' + filename)

*# store loaded image*

loaded\_images.append(img\_data)

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

2.Створення датафрейму з данними математичних очікувань, дисперсії, коефіцієнтів асиметрії та ексцесу значень яскравості для кожного каналу кольору.

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

RED = 0

GREEN = 1

BLUE = 2

*# 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))

Таким чином отримуэмо значення Мат. очікування та дисперсії:

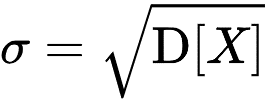
Red: Expected value - 111.54, Variance - 6052.51

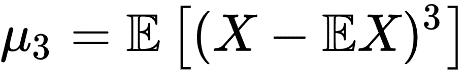
Green: Expected value - 102.63, Variance - 5560.91

Blue: Expected value - 92.00, Variance - 5701.13

1. Коефіцієнти асиметрії та ексцесу

Використаємо відповідні формули коефіцієнту асиметрії:





Реалізуємо даний функціонал за допомогою такого коду:

**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 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))

Отримаємо:

Red: Asymmetry - 0.199, Excess - 1.831

Green: Asymmetry - 0.335, Excess - 1.997

Blue: Asymmetry - 0.534, Excess - 2.114

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

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

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))

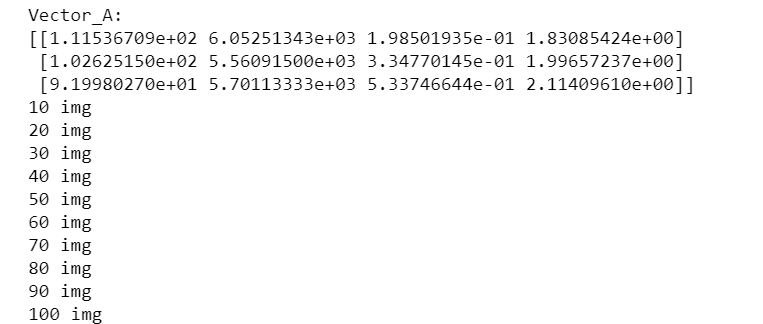
image = np.swapaxes(image, 0, 1)

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

itear += 1

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

print(str(itear) + " img")

****

4. Побудувати гаусові моделі зображень з використанням розрахованих раніше параметрів

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]))

Expected value + colors:

[784.06243683]

Expected value + dispersion + colors:

[[6053.30062782 5204.68775305]

[5204.68775305 5561.579443 ]]

Expected value + dispersion + asymetry + colors:

[[6053.30062782 5204.68775305 4502.85354859]

[5204.68775305 5561.579443 5028.78358684]

[4502.85354859 5028.78358684 5701.83490569]]

Expected value + dispersion + asymetry + excess + colors:

[[6053.30062782 5204.68775305 4502.85354859]

[5204.68775305 5561.579443 5028.78358684]

[4502.85354859 5028.78358684 5701.83490569]]

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.

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

**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()

Отримаємо наступний результат:

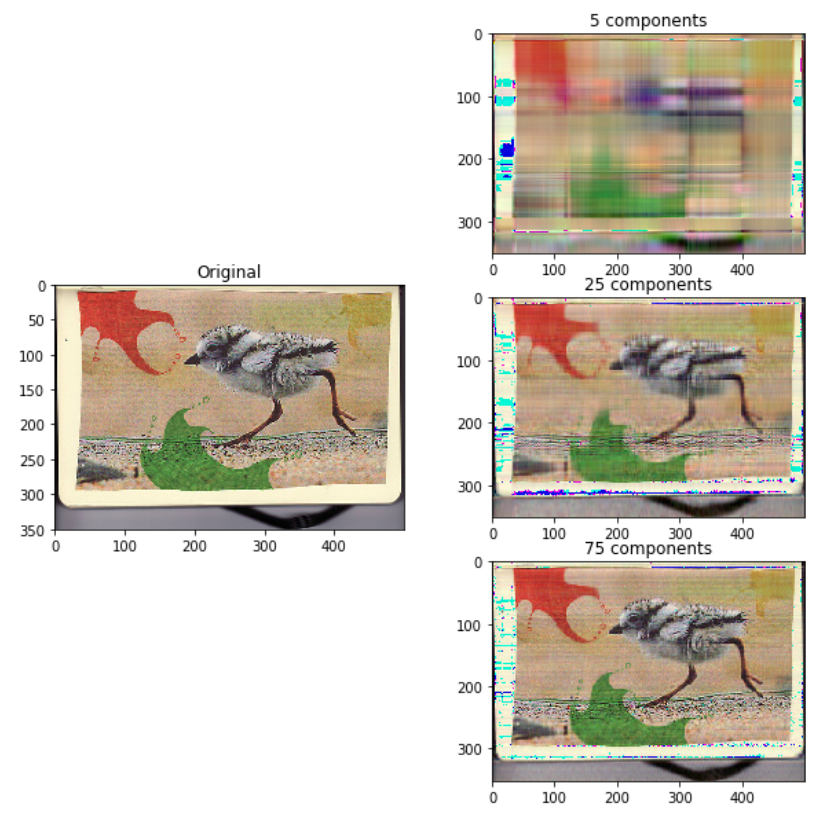


Рисунок 1 **–** відновлені фото з різною кількістю компонентів

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.

Побудова графіку залежності MSE від кількості 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()

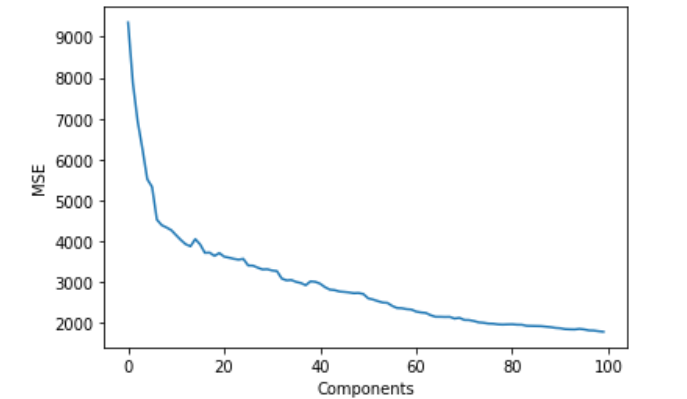


Рисунок 2 – залежність MSE відновлених фото від кількості компонент

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 однієї марковської 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("**\n**Red 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("**\n**Red 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\n**Green matrix 1st oder:**\n**", markov\_matrix)

print("**\n**Green 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

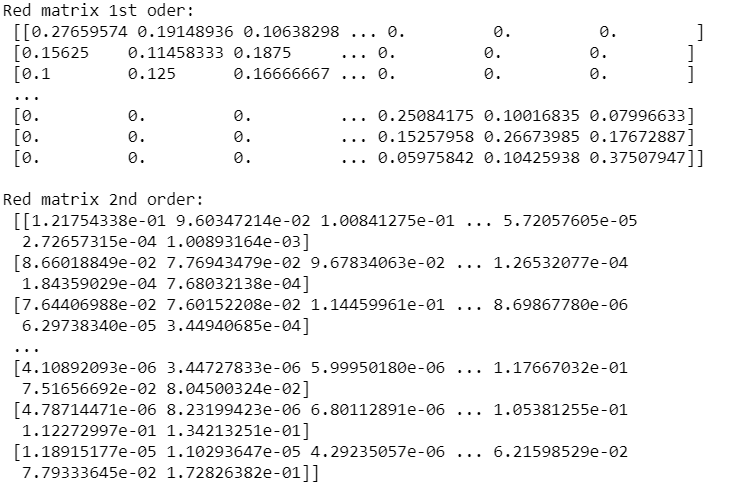
markov\_matrix = markov\_matrix3[0] / sum(markov\_matrix3[0])

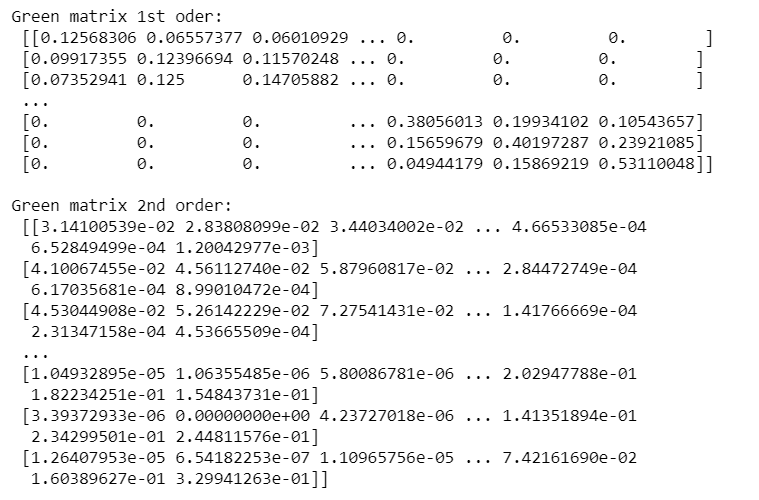
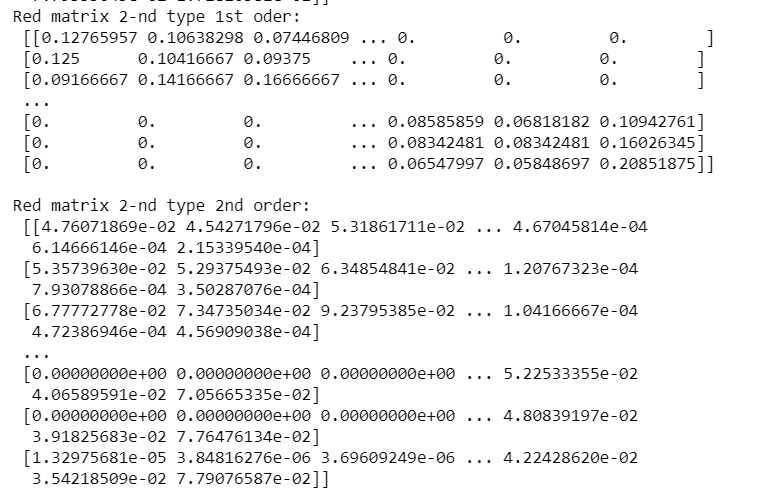
**for** i **in** range(1, 256):

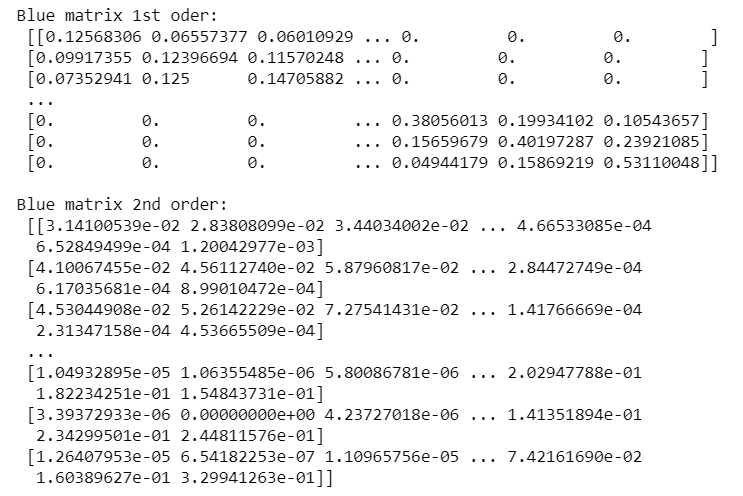
markov\_matrix = np.vstack((markov\_matrix, markov\_matrix3[i] / sum(markov\_matrix3[i])))

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

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







Приклад марковського ланцюга

**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='yellow', font\_color='blue')

plt.show()

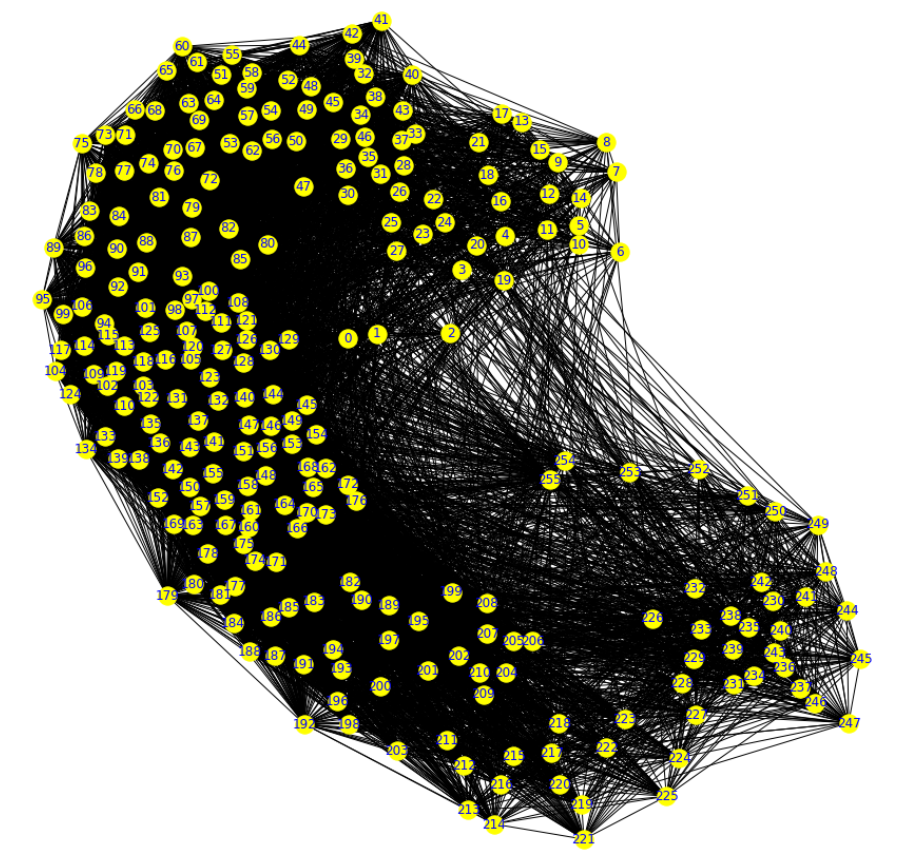
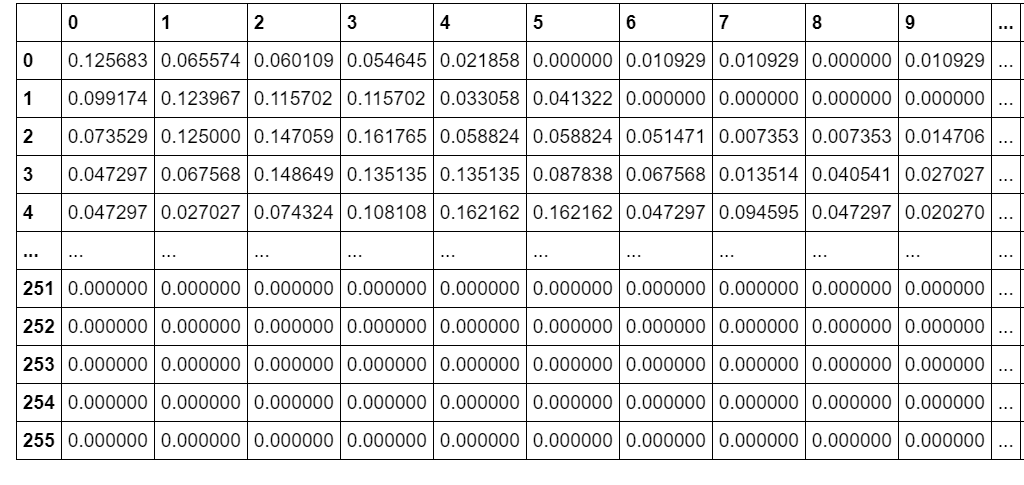


Рисунок 3 - Вигляд марківського ланцюга

Перевірка гіпотез



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

Матриця регулярна, незворотня і рекурентна .

ВИСНОВОК

В лабораторній роботі було проаналізовано вибірку з 250 зображень датасету MIRFlickr-20k. Було знайдено що всі канали охоплюють увесь спектр значень. Було знайдено мат. очікування – 111.54 для чевоного каналу, 102.63 для зеленого і 92.00 для синьогоканалу відповідно і дисперсію 6052.51, 5560.91 і 5701.13.

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.