МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НТУУ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Лабораторна робота №2

з дисципліни

«*Автоматизація обробки ІзОД*»

Варіант 4

**Виконав:**

студент 5 курсу ФТІ

групи ФЕ-91мп

Карнаух М.Ю.

**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.

**ХІД РОБОТИ**

Роботу виконуватимемо мовою Python за допомогою блокового інтерпретатора Jupyter.

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

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

np.random.RandomState(4)

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('mirflickr/' + filename)

loaded\_images.append(img\_data)

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

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

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:\tMatematuchne ochikuvannya - {0:.2f},\tDispersia - {1:.2f}"\

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

Результат:

Red: Matematuchne ochikuvannya - 107.82, Dispersia — 5984.72

Green: Matematuchne ochikuvannya - 99.48, Dispersia - 5440.11

Blue: Matematuchne ochikuvannya - 88.25, Dispersia - 5722.87

**b. Коефіцієнти асиметрії та ексцесу**

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) - 3

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

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

print('Red:\tAsymmetry - {0:.3f},\tEkscess - {1:.3f}'\

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

print('Green:\tAsymmetry - {0:.3f},\tEkscess - {1:.3f}'\

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

print('Blue:\tAsymmetry - {0:.3f},\tEkscess - {1:.3f}'\

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

Результат:

Red: Asymmetry - 0.245, Ekscess - -1.107

Green: Asymmetry - 0.389, Ekscess - -0.936

Blue: Asymmetry - 0.624, Ekscess - -0.772

**3. Побудувати вектори параметрів зображень.**

Для цього сформуємо вектор всіх потрібних нам значень, та для розрахунків будемо брати окремі частини 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)

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

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

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

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

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

Результат

Matematuchne ochikuvannya + colors:

[-4914.01197641]

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

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

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

Результат:

Matematuchne ochikuvannya + dispersion + colors:

[[5985.48793424 4990.60222595]

[4990.60222595 5440.74451201]]

Matematuchne ochikuvannya + dispersion + asymetry + colors:

[[5985.48793424 4990.60222595 4229.29320218]

[4990.60222595 5440.74451201 4975.77252585]

[4229.29320218 4975.77252585 5723.5728941 ]]

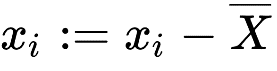
Matematuchne ochikuvannya + dispersion + asymetry + ekscess + colors:

[[5985.48793424 4990.60222595 4229.29320218]

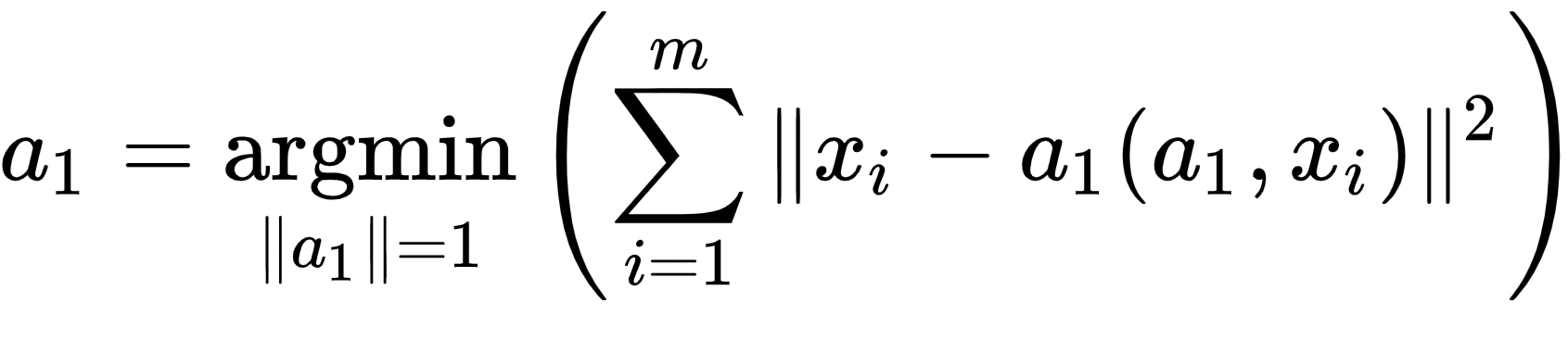
[4990.60222595 5440.74451201 4975.77252585]

[4229.29320218 4975.77252585 5723.5728941 ]]

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

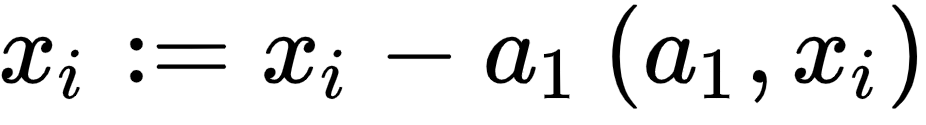
Для проведення декомпозиції каналів кольору сформуємо функцію що працюваниме по методу головних компонент (PCA). Алгоритм даної функції буде наступним:

1. Централізувати дані (відніманням середнього)

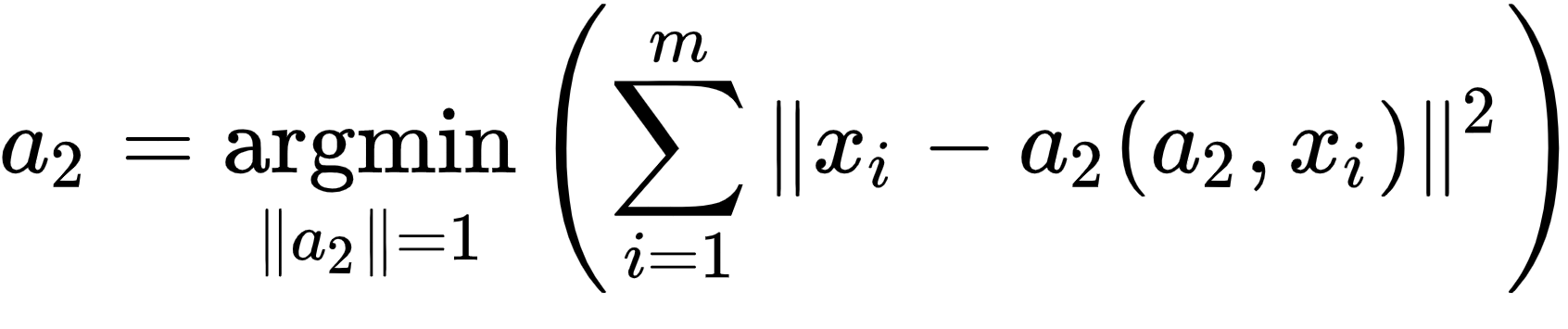
2. Знайти першу головну компоненту як рішення задачі: 

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.

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[0])

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()

Передавши параметри тестової фотографії отримаємо наступний результат, який зображено на Рисунку 1.

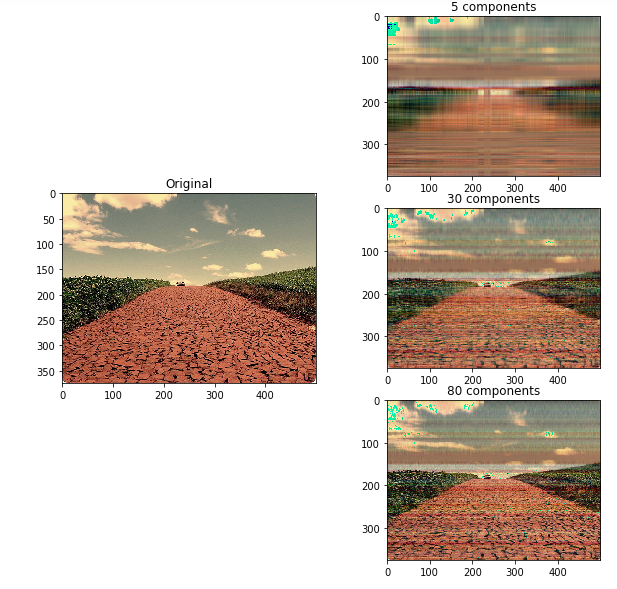


Рисунок 1 **–** відновлені фото з різної кількості компонентів

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 потрібна для відновлення таких частин.

Після цього виконаємо функцію для багатьох кроків та порівняємо 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:

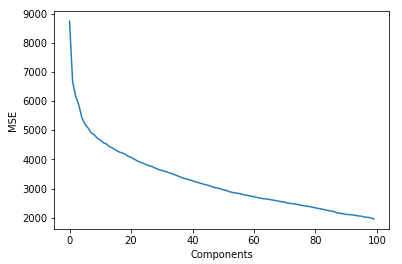


Рисунок 2 – залежність MSE відновлених фото від кількості компонент

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

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

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 пройшовши по всіх пікселях картинки запишемо кількість переходів між ними.

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))

Результат:  
Red matrix 1st oder:

[[0.17832647 0.06492913 0.04206676 ... 0. 0. 0. ]

[0.15693013 0.05612829 0.03894616 ... 0. 0. 0. ]

[0.20084567 0.07610994 0.0359408 ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.14375 0.0625 0.08125 ]

[0. 0. 0. ... 0.07194245 0.10071942 0.1294964 ]

[0. 0. 0. ... 0.06779661 0.05423729 0.17288136]]

Red matrix 2nd order:

[[1.06256040e-01 4.13083108e-02 2.42038838e-02 ... 8.08730971e-05

4.07558663e-05 1.72651470e-04]

[1.00491518e-01 3.94491932e-02 2.27409899e-02 ... 6.27122622e-05

2.67847610e-05 1.38397906e-04]

[1.09511554e-01 4.25160422e-02 2.54782611e-02 ... 5.42093565e-05

1.39051864e-05 1.68862534e-04]

...

[7.44969342e-04 4.77852920e-04 2.58544887e-04 ... 4.50342717e-02

3.22593128e-02 5.61927605e-02]

[2.44083321e-04 1.23193257e-04 4.14004598e-05 ... 3.76409480e-02

3.09957380e-02 5.81675903e-02]

[1.45137366e-03 5.21536125e-04 3.03660058e-04 ... 3.44736568e-02

2.68838440e-02 5.60463459e-02]]

Red matrix 2-nd type 1st oder:

[[0.09739369 0.04298125 0.0260631 ... 0. 0. 0.00045725]

[0.10767468 0.05956472 0.03665521 ... 0. 0. 0. ]

[0.1141649 0.05919662 0.0359408 ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.04375 0.0375 0.0625 ]

[0. 0. 0. ... 0.05755396 0.03597122 0.10791367]

[0. 0. 0. ... 0.03389831 0.0440678 0.09830508]]

Red matrix 2-nd type 2nd order:

[[0.06008802 0.02517258 0.01443823 ... 0.00018759 0.00019973 0.00051208]

[0.06325564 0.02693097 0.01562044 ... 0.0001679 0.00013175 0.00045045]

[0.06552881 0.02835714 0.01612735 ... 0.00012579 0.00022912 0.00034043]

...

[0.00188403 0.00099499 0.00023942 ... 0.02233869 0.01721226 0.02701799]

[0.00292634 0.00093572 0.0004892 ... 0.02260101 0.0173546 0.0300109 ]

[0.00568499 0.00199143 0.00121066 ... 0.0162625 0.01427334 0.02515076]]

Green matrix 1st oder:

[[0.15315076 0.07870247 0.06088806 ... 0. 0. 0. ]

[0.16699155 0.09031839 0.05523067 ... 0. 0. 0. ]

[0.17549407 0.07351779 0.05296443 ... 0. 0. 0. ]

...

[0.07142857 0. 0. ... 0. 0. 0. ]

[0. 0. 0. ... 0. 0.05882353 0.17647059]

[0. 0. 0. ... 0.04651163 0.11627907 0.1627907 ]]

Green matrix 2nd order:

[[1.00951828e-01 4.40135476e-02 3.54349481e-02 ... 7.11665851e-06

3.08413674e-06 8.64892262e-05]

[1.04840456e-01 4.62931414e-02 3.73410120e-02 ... 3.35571570e-06

2.40457029e-06 1.74971448e-05]

[1.02913211e-01 4.49903536e-02 3.68760689e-02 ... 3.78022340e-06

2.69463457e-06 1.55348179e-05]

...

[1.64780630e-02 9.22992411e-03 6.62745673e-03 ... 1.46683673e-02

2.75456562e-02 8.16326531e-02]

[4.32905936e-04 1.97987687e-04 2.81901898e-04 ... 1.99726402e-02

6.14800435e-02 1.18386468e-01]

[7.56496083e-03 1.03955484e-03 9.45366463e-04 ... 2.85399605e-02

5.23307511e-02 9.69375906e-02]]

Blue matrix 1st oder:

[[0.15315076 0.07870247 0.06088806 ... 0. 0. 0. ]

[0.16699155 0.09031839 0.05523067 ... 0. 0. 0. ]

[0.17549407 0.07351779 0.05296443 ... 0. 0. 0. ]

...

[0.07142857 0. 0. ... 0. 0. 0. ]

[0. 0. 0. ... 0. 0.05882353 0.17647059]

[0. 0. 0. ... 0.04651163 0.11627907 0.1627907 ]]

Blue matrix 2nd order:

[[1.00951828e-01 4.40135476e-02 3.54349481e-02 ... 7.11665851e-06

3.08413674e-06 8.64892262e-05]

[1.04840456e-01 4.62931414e-02 3.73410120e-02 ... 3.35571570e-06

2.40457029e-06 1.74971448e-05]

[1.02913211e-01 4.49903536e-02 3.68760689e-02 ... 3.78022340e-06

2.69463457e-06 1.55348179e-05]

...

[1.64780630e-02 9.22992411e-03 6.62745673e-03 ... 1.46683673e-02

2.75456562e-02 8.16326531e-02]

[4.32905936e-04 1.97987687e-04 2.81901898e-04 ... 1.99726402e-02

6.14800435e-02 1.18386468e-01]

[7.56496083e-03 1.03955484e-03 9.45366463e-04 ... 2.85399605e-02

5.23307511e-02 9.69375906e-02]]

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

data = markov\_matrix

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

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

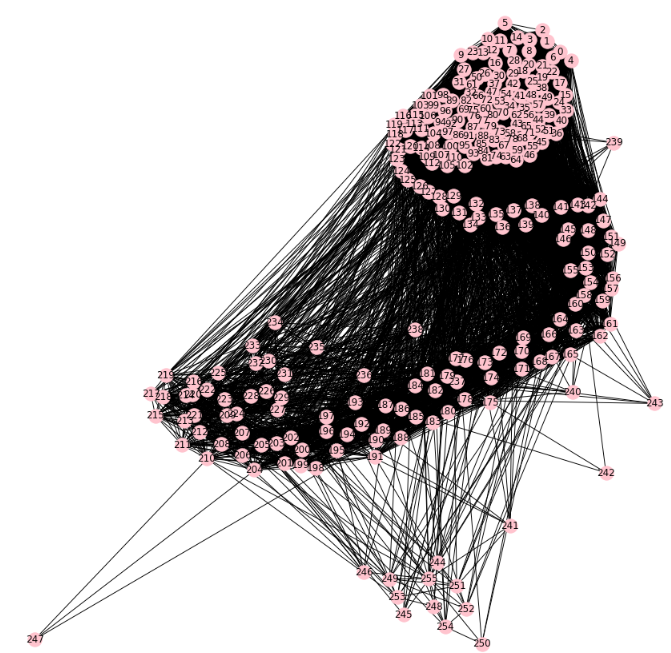
df2 = pd.DataFrame(data, index=ind, columns=ind)

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

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

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

plt.show()

Рисунок 3 - Вигляд марківського ланцюга

Перевірка властивостей регулярності, рекурентності і незворотності для отриманих марковских моделей для 5 ітерацій.

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-ти ітерації, виконується умова регулярності. \nМодель регулярна ')

P = markov\_matrix

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

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

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

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

print('Всі стани рекурентні, модель рекурентна. ')

else :

print('Модель не рекурентна' )

Результат:  
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-ти ітерації, виконується умова регулярності.

Модель регулярна

Модель незворотня - True

Кількість рекурентних станів 256

Всі стани рекурентні, модель рекурентна.

**ВИСНОВКИ**

В даній лабораторній роботі було проаналізовано вибірку з 250 зображень. Було знайдено що всі канали охоплюють увесь спектр значень. Було знайдено мат. очікування і дисперсію.

Було побудовано вектори даних та гаусові 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.

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 та майже непомітні при великих значеннях.

Було проведено моделювання окремих каналів кольору зображень з використанням марковських ланцюгів, та сформовано стохастичні матриці за різними типами обходів.

Була виконана перевірка властивостей регулярності, рекурентності і незворотності для отриманих марковских моделей для 5 ітерацій. Всі умови для отриманої моделі були виконані.