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»

Керівник: Виконав:

Прогонов Дмитро Олександрович студент 5 курсу групи ФЕ-91мп

Захищено з оцінкою Павлусь Олександр Сергійович

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

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

дата, підпис

Київ – 2020 р.

Завдання лабораторної роботи:

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

Хід роботи

1) Сформувати тестову вибірку зображень з вихідного пакета;

"from os import listdir\n",

"from matplotlib import image\n",

"import random\n",

"import numpy as np \n",

"from dataclasses import dataclass\n",

"\n",

"# load images in a directory \n",

"random.seed(8)\n",

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

"loaded\_images = list()\n",

"for i in range(250):\n",

"\t# load image\n",

"\tfilename = 'im' + str(random\_indexes[i]) + '.jpg'\n",

"\timg\_data = image.imread('E:/Labs/mirflickr/' + filename)\n",

"\t# store loaded image\n",

"\tloaded\_images.append(img\_data)\n",

"\tprint('> loaded %s %s' % (filename, img\_data.shape))"

2) Для кожного каналу кольору кожного зображення з тестового пакета обчислити наступні характеристики: Математичне сподівання і дисперсію; Коефіцієнти асиметрії та ексцесу (нормалізований);

***1. Математичне сподівання та дисперсія;***

# 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))

Отримуємо:  
Red: Expected value - 113.48, Variance - 6126.48

Green: Expected value - 104.92, Variance - 5634.68

Blue: Expected value - 97.43, Variance - 6118.77

***2. Коефіцієнти асиметрії та ексцесу (нормалізований)***

#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))

Отримуємо:  
 Red: Asymmetry - 0.162, Excess - 1.811

Green: Asymmetry - 0.299, Excess - 1.959

Blue: Asymmetry - 0.426, Excess - 1.942

3) Побудувати вектори параметрів зображень, що складаються з:

* + Математичних очікувань значень яскравості для кожного каналу кольору;
  + Математичних очікувань і дисперсії значень яскравості для кожного каналу кольору;
  + Математичних очікувань, дисперсії і коефіцієнта асиметрії значень яскравості для кожного каналу кольору;
  + Математичних очікувань, дисперсії, коефіцієнтів асиметрії та ексцесу значень яскравості для кожного каналу кольору;

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

Код програми:

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)

Vector\_All\_DATA – містить дані трьох каналів

Отримуємо:

Vector\_A:

[[1.13479336e+02 6.12647805e+03 1.61803881e-01 1.81072875e+00]

[1.04923001e+02 5.63468056e+03 2.98685452e-01 1.95934798e+00]

[9.74325979e+01 6.11877257e+03 4.25669692e-01 1.94242659e+00]]

10 img

\*\*\*

250 img

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

Тепер сформуємо гаусові моделі.

#a Expected value

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]))

Вихідні значення:

Expected value + colors:

[-4665.36960346]

Expected value + dispersion + colors:

[[6127.27295305 5160.31674988]

[5160.31674988 5635.35280141]]

Expected value + dispersion + asymetry + colors:

[[6127.27295305 5160.31674988 4496.88611699]

[5160.31674988 5635.35280141 5366.92147338]

[4496.88611699 5366.92147338 6119.56953015]]

Expected value + dispersion + asymetry + excess + colors:

[[6127.27295305 5160.31674988 4496.88611699]

[5160.31674988 5635.35280141 5366.92147338]

[4496.88611699 5366.92147338 6119.56953015]]

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

* + Варіюючи кількість компонент, провести реконструкцію окремих каналів кольору зображень (від компонент з найбільшою енергією поступово переходячи до компонентів з мінімальною енергією).
  + Побудувати залежність помилки відновлення (середнє відхилення вихідного зображення відреконструйованого, MSE) від кількості використаних компонент.

"import scipy\n",

"import scipy.ndimage\n",

"import matplotlib.pyplot as plt \n",

"import numpy as np \n",

"from PIL import Image\n",

"\n",

"test\_img = loaded\_images[8]\n",

"a\_np = np.array(test\_img)\n",

"a\_r = a\_np[:,:,0]\n",

"a\_g = a\_np[:,:,1]\n",

"a\_b = a\_np[:,:,2]\n",

"\n",

**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()

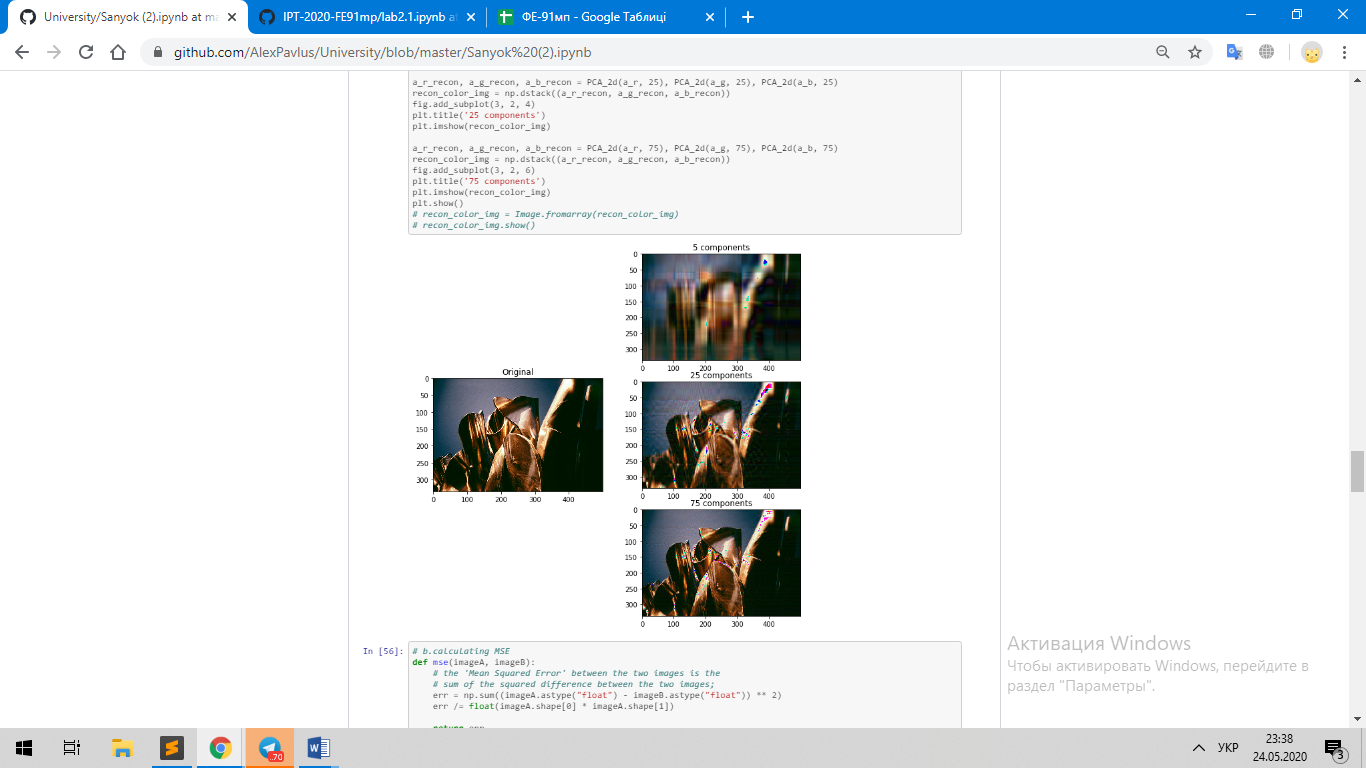


Рисунок 1 — Зображення для аналізу методу головних компонент

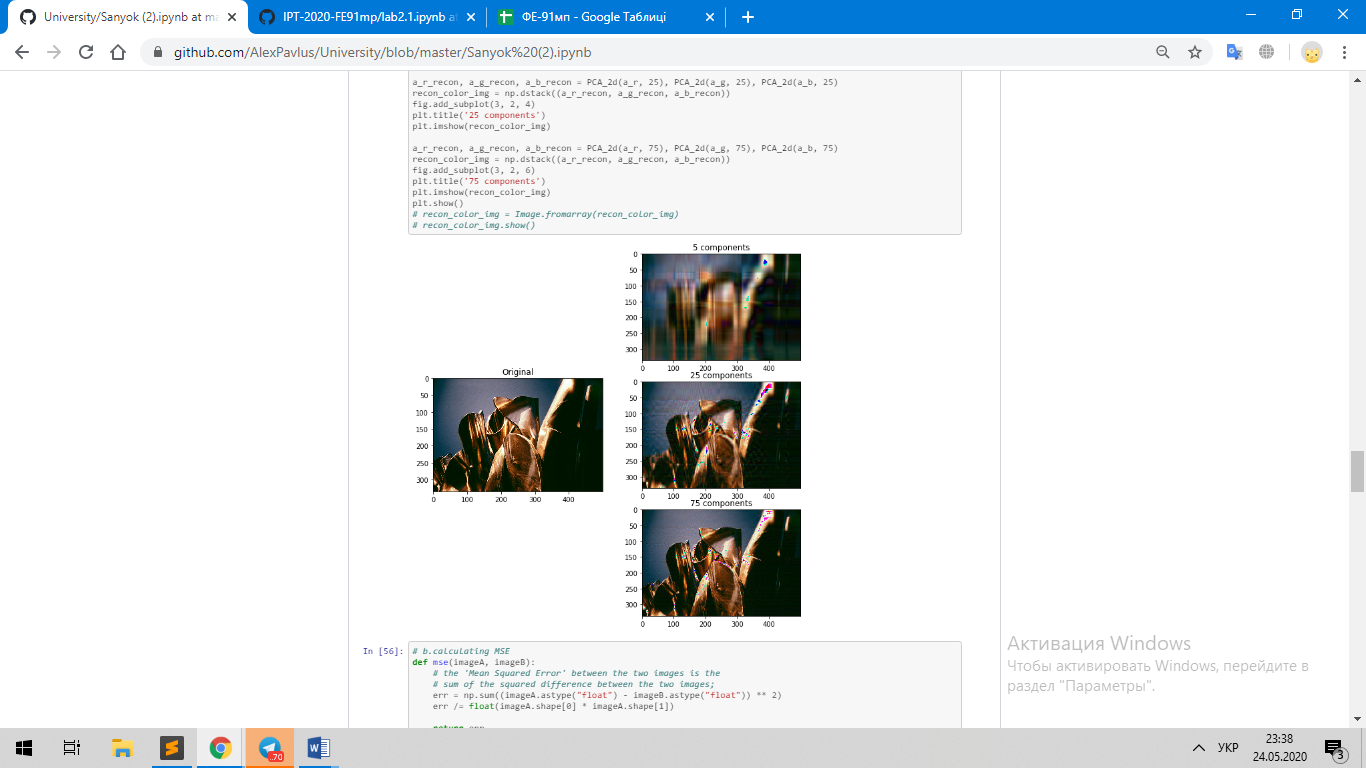
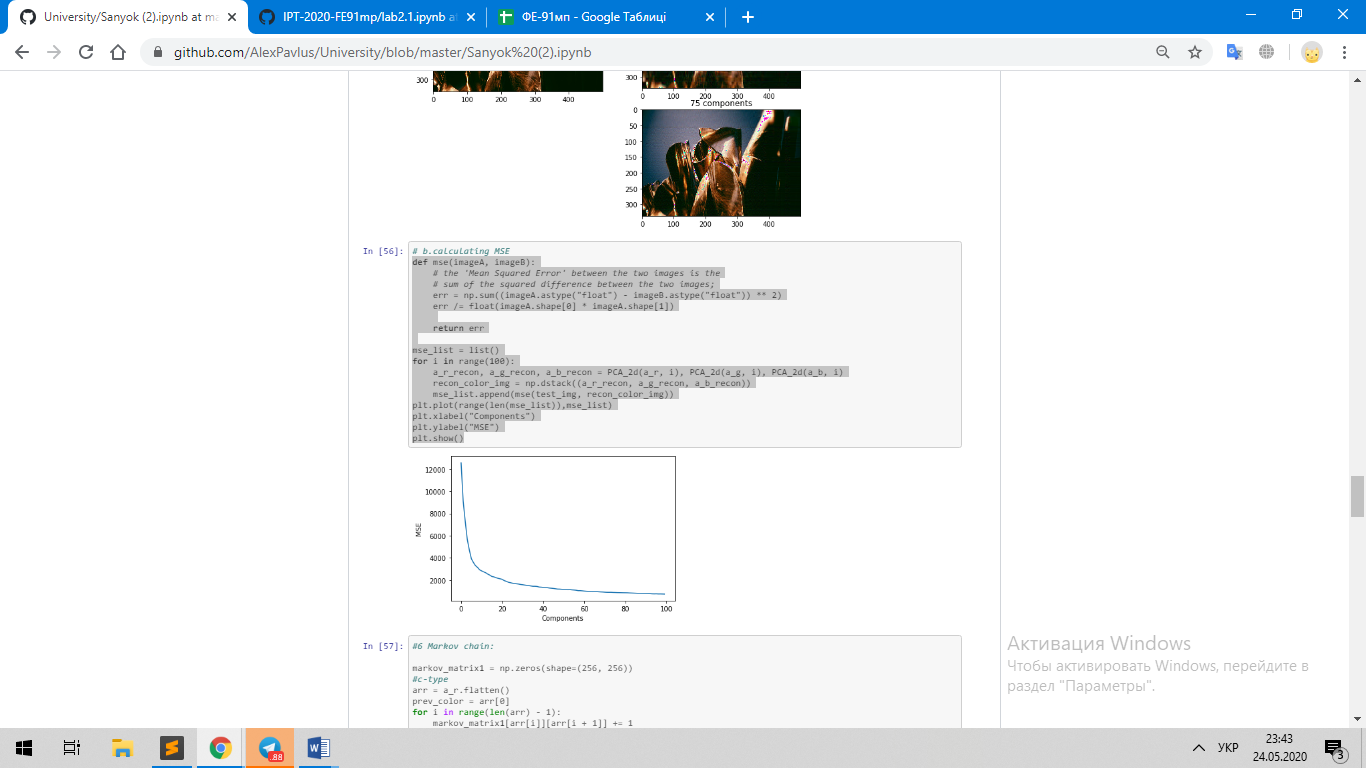


Рисунок 2 – Відновлені зображення при різній кількості головних компонент

Покажемо на графіку залежність помилки відновлення від кількості використаних компонент.



Код програми:

**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()

6) Провести моделювання окремих каналів кольору зображень з використанням марковських ланцюгів:

* + 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.

"markov\_matrix1 = np.zeros(shape=(256, 256))\n",

"#c-type\n",

"arr = a\_r.flatten()\n",

"prev\_color = arr[0]\n",

"for i in range(len(arr) - 1):\n",

" markov\_matrix1[arr[i]][arr[i + 1]] += 1\n",

"markov\_matrix = markov\_matrix1[0] / sum(markov\_matrix1[0])\n",

"for i in range(1, 256):\n",

" markov\_matrix = np.vstack((markov\_matrix, markov\_matrix1[i] / sum(markov\_matrix1[i])))\n",

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

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

"\n",

"markov\_matrix1 = np.zeros(shape=(256, 256))\n",

"#Fortran-type\n",

"arr = a\_r.flatten('F')\n",

"prev\_color = arr[0]\n",

"for i in range(len(arr) - 1):\n",

" markov\_matrix1[arr[i]][arr[i + 1]] += 1\n",

"markov\_matrix = markov\_matrix1[0] / sum(markov\_matrix1[0])\n",

"for i in range(1, 256):\n",

" markov\_matrix = np.vstack((markov\_matrix, markov\_matrix1[i] / sum(markov\_matrix1[i])))\n",

"print(\"Red matrix 2-nd type 1st oder:\\n\", markov\_matrix)\n",

"print(\"\\nRed matrix 2-nd type 2nd order:\\n\", np.linalg.matrix\_power(markov\_matrix, 2))\n",

"\n",

"markov\_matrix2 = np.zeros(shape=(256, 256))\n",

"arr = a\_g.flatten()\n",

"prev\_color = arr[0]\n",

"for i in range(len(arr) - 1):\n",

" markov\_matrix2[arr[i]][arr[i + 1]] += 1\n",

"markov\_matrix = markov\_matrix2[0] / sum(markov\_matrix2[0])\n",

"for i in range(1, 256):\n",

" markov\_matrix = np.vstack((markov\_matrix, markov\_matrix2[i] / sum(markov\_matrix2[i])))\n",

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

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

"\n",

"markov\_matrix3 = np.zeros(shape=(256, 256))\n",

"arr = a\_g.flatten()\n",

"prev\_color = arr[0]\n",

"for i in range(len(arr) - 1):\n",

" markov\_matrix3[arr[i]][arr[i + 1]] += 1\n",

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

"for i in range(1, 256):\n",

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

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

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

Отримуємо такі дані:  
  
Red matrix 1st oder:

[[7.64197119e-01 1.19055565e-01 4.09644585e-02 ... 0.00000000e+00

0.00000000e+00 2.78102230e-05]

[2.13454198e-01 6.42795802e-01 7.91507634e-02 ... 0.00000000e+00

0.00000000e+00 0.00000000e+00]

[2.00667881e-01 2.70188221e-01 2.94626594e-01 ... 0.00000000e+00

0.00000000e+00 0.00000000e+00]

...

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 3.23741007e-02

2.87769784e-02 7.55395683e-02]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 2.42214533e-02

1.73010381e-02 2.76816609e-02]

[0.00000000e+00 0.00000000e+00 0.00000000e+00 ... 1.93236715e-02

2.17391304e-02 5.31400966e-02]]

Red matrix 2nd order:

[[6.29050053e-01 1.85802339e-01 6.07096130e-02 ... 1.45982868e-05

2.05760152e-05 8.31629664e-05]

[3.27326388e-01 4.67400071e-01 9.11941054e-02 ... 4.56741181e-06

4.45261287e-06 2.62930678e-05]

[3.11696739e-01 3.05131820e-01 1.47705402e-01 ... 6.40654166e-06

1.27960651e-05 4.58986323e-05]

...

[7.79013329e-04 1.29662043e-04 2.37982554e-04 ... 1.53765051e-02

1.44203842e-02 3.51156602e-02]

[8.76624920e-04 1.04676451e-04 2.58655774e-04 ... 1.26942287e-02

1.21752947e-02 3.12777075e-02]

[1.20864166e-03 2.41201147e-04 2.31002231e-04 ... 1.21030131e-02

1.17006127e-02 3.11519293e-02]]

Red matrix 2-nd type 1st oder:

[[0.81739808 0.106096 0.02889482 ... 0. 0. 0. ]

[0.1802958 0.69103053 0.07943702 ... 0. 0. 0. ]

[0.15543412 0.24119611 0.36612022 ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.02877698 0.05755396 0.1618705 ]

[0. 0. 0. ... 0.04152249 0.03806228 0.13148789]

[0. 0.00120773 0. ... 0.06642512 0.07608696 0.17753623]]

Red matrix 2-nd type 2nd order:

[[6.97138947e-01 1.70657911e-01 4.77575160e-02 ... 1.45614648e-05

1.29353377e-05 3.77847514e-05]

[2.90808945e-01 5.20750333e-01 9.64218609e-02 ... 2.43830466e-06

3.30966656e-06 8.63456239e-06]

[2.60405000e-01 2.97679565e-01 1.96472941e-01 ... 1.99572721e-06

4.76846393e-06 1.99568407e-05]

...

[4.85782361e-04 3.04334262e-04 7.25841390e-05 ... 2.76059327e-02

2.78665547e-02 7.44263632e-02]

[6.12652983e-04 3.11393765e-04 2.25554207e-04 ... 2.55002795e-02

2.56251644e-02 6.83376340e-02]

[9.30097825e-04 1.27004888e-03 2.72773338e-04 ... 2.88873739e-02

2.98300121e-02 8.02604081e-02]]

Green matrix 1st oder:

[[0.01162791 0.00581395 0.01744186 ... 0. 0. 0. ]

[0.025 0.0125 0.025 ... 0. 0. 0. ]

[0.01428571 0.04285714 0.01428571 ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.13853904 0.15365239 0.2418136 ]

[0. 0. 0. ... 0.14512922 0.21073559 0.26640159]

[0. 0. 0. ... 0.07665198 0.12599119 0.26519824]]

Green matrix 2nd order:

[[3.05558915e-03 3.65689263e-03 1.73367671e-03 ... 4.40251006e-04

1.02648033e-03 3.18388835e-03]

[3.74185791e-03 3.34588017e-03 1.60107993e-03 ... 7.69296950e-04

3.41739816e-04 1.79936072e-03]

[3.73550745e-03 2.72177021e-03 2.07616132e-03 ... 5.97142288e-04

5.95494036e-04 2.02431706e-03]

...

[5.46184777e-04 1.82912059e-04 1.89928856e-04 ... 7.39089836e-02

1.00686155e-01 1.82314072e-01]

[3.09395408e-04 8.72460983e-05 1.63095593e-04 ... 8.25875324e-02

1.13484669e-01 1.95908192e-01]

[7.51730560e-04 2.16026071e-04 2.13359125e-04 ... 6.25408508e-02

8.73842432e-02 1.64327913e-01]]

Blue matrix 1st oder:

[[0.01162791 0.00581395 0.01744186 ... 0. 0. 0. ]

[0.025 0.0125 0.025 ... 0. 0. 0. ]

[0.01428571 0.04285714 0.01428571 ... 0. 0. 0. ]

...

[0. 0. 0. ... 0.13853904 0.15365239 0.2418136 ]

[0. 0. 0. ... 0.14512922 0.21073559 0.26640159]

[0. 0. 0. ... 0.07665198 0.12599119 0.26519824]]

Blue matrix 2nd order:

[[3.05558915e-03 3.65689263e-03 1.73367671e-03 ... 4.40251006e-04

1.02648033e-03 3.18388835e-03]

[3.74185791e-03 3.34588017e-03 1.60107993e-03 ... 7.69296950e-04

3.41739816e-04 1.79936072e-03]

[3.73550745e-03 2.72177021e-03 2.07616132e-03 ... 5.97142288e-04

5.95494036e-04 2.02431706e-03]

...

[5.46184777e-04 1.82912059e-04 1.89928856e-04 ... 7.39089836e-02

1.00686155e-01 1.82314072e-01]

[3.09395408e-04 8.72460983e-05 1.63095593e-04 ... 8.25875324e-02

1.13484669e-01 1.95908192e-01]

[7.51730560e-04 2.16026071e-04 2.13359125e-04 ... 6.25408508e-02

8.73842432e-02 1.64327913e-01]]

Графічне представлення 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])]

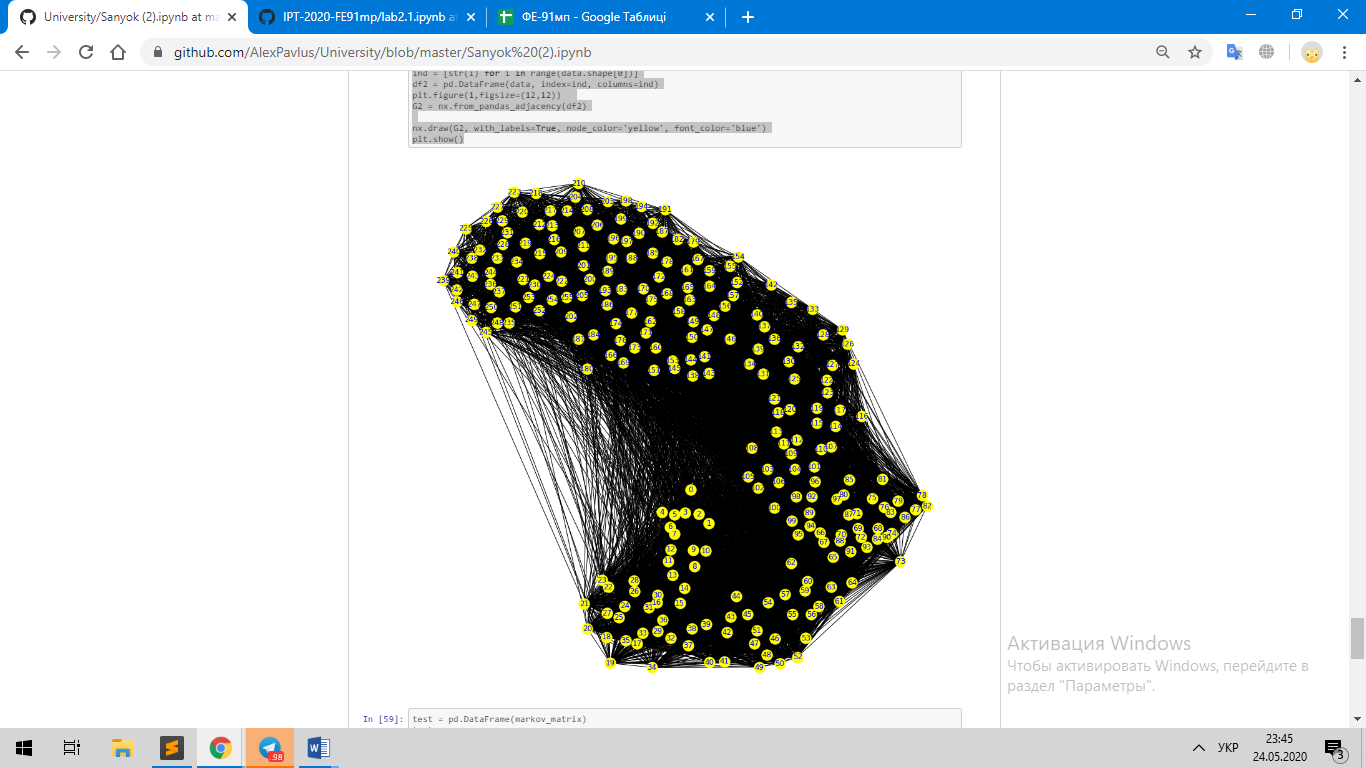
df2 = pd.DataFrame(data, index=ind, columns=ind)

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()



**Перевірка коректності гіпотез:**

**import** **quantecon** **as** **qe**

**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 ітерації ,виконується умова регулярності.Модель регулярна ')

P = markov\_matrix

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

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

print('Рекурентную називають модель M яка з ймовірністю в P = 1 при покиданні стану вернеться в цей стан за час t < inf,у майбутньму')

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

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

print('Усі стани моделі рекурентні, модель рекурентна ')

**else** :

print('Модель не рекурентна' )

Отримуємо:

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

Рекурентну 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 .

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 математичне очікування – 114 для чевоного каналу, 105 для зеленого і 97 для синього каналу відповідно і дисперсію 6126.48, 5634.68 та 6118.7.

Було побудовано вектори даних та знайдено Гаусовські моделі для одновимірного та багатовимірних варіантів в залежності від кількості даних.

Було проведено декомпозицію зображення за допомогою метода головних компонент (РСА) для різної кількості компонент. 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 проведено моделювання окремих каналів кольору зображень з використанням марківських ланцюгів, та сформовано стохастичні матриці за різними типами обходів. Також за даними було побуловано графічну модель марківського ланцюга.