Міністерство освіти і науки України

НТУУ «Київський політехнічний інститут ім. Ігоря Сікорського»

Фізико-технічний інститут

Лабораторна робота №2

З дисципліни

«Автоматизація обробки ІзОД»

Варіант 6

**Виконав:**

Студент 5 курсу ФТІ

групи ФЕ-91мп

Назаров О.В.

**Перевірив:**

Прогонов Д. О.

Київ-2020

**І. Підготовка**

***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;
   2. 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

Завантаження вибірки зображень відносно варіанту

test\_index = random.sample(range(6,25001),250)

# Path to where my test images are stored

img\_folder = os.path.join(os.getcwd(), 'mirflickr/')

test\_images = []

for i in test\_index:

test\_images.append('C:\\Users\\malin\\Desktop\\Diplom\\datasets\\mirflickr/im'+ str(i) + '.jpg')

def convert(filename):

im = Image.open(filename,mode = "r")

return np.array(im)

def img\_list():

t = time.time()

np\_arrays = map(convert, test\_images)

arr\_list = [i for i in np\_arrays]

return arr\_list

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

params = ['Image Number','Channel Number','ExpValue(UnifDist)','Variance','Skew',"Kurtosis","Intensity of channel",'Img']

def part1(img):

param\_list = []

for i in range(len(img)):

temp = img[i]# інтенсивність яскравості кожного каналу

mn = temp.sum(axis=0).sum(axis=0)/(temp.shape[0]\*temp.shape[1])

intensity\_of\_channels = mn/np.linalg.norm(mn, ord=None)

for Channel in range(3):

img\_ch = img[i][:,:,Channel]

param\_list.append(list((test\_index[i],Channel,np.mean(img\_ch),np.var(img\_ch)

,skew(img\_ch,None),kurtosis(img\_ch,None),intensity\_of\_channels[Channel],img\_ch)))

return pd.DataFrame(param\_list,columns = params)

images = img\_list()

Data = part1(images)

x\_i = Data[['ExpValue(UnifDist)','Variance','Skew','Kurtosis','Intensity of channel']]

labels = x\_i.columns

Дані трасформуємо у проміжок від 0 до 1

ss = MinMaxScaler()

ss.fit(x\_i)

x\_i\_scaled = pd.DataFrame(ss.transform(x\_i),columns=labels)

std = np.array(x\_i\_scaled.std())

cov = np.cov(x\_i\_scaled.T)

Будуємо коваріаційну матрицю

cov\_diag = np.eye(5,5) \* np.array(x\_i\_scaled.var())

gauss\_samples\_from\_distributions = np.random.multivariate\_normal(std,cov\_diag,750)

Приклади розподілів Гаусівської моделі параметрів зображень

plt.figure(5,figsize=(18,9))

plt.subplot('521');plt.plot(gauss\_samples\_from\_distributions[:,0],x\_i\_scaled['ExpValue(UnifDist)'],'.')

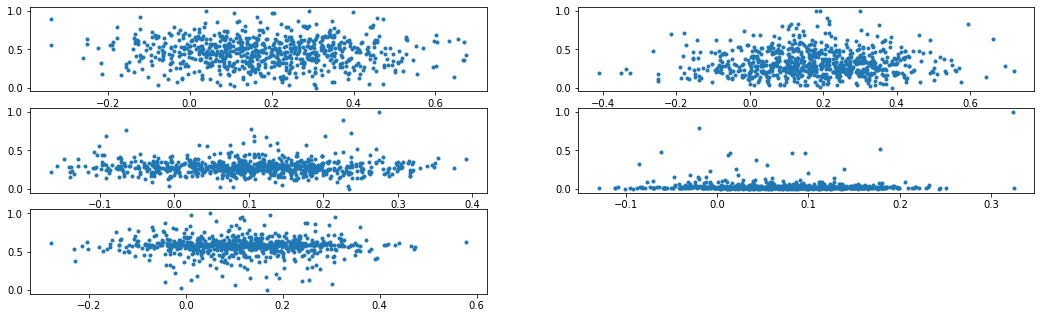
plt.subplot('522');plt.plot(gauss\_samples\_from\_distributions[:,1],x\_i\_scaled['Variance'],'.')

plt.subplot('523');plt.plot(gauss\_samples\_from\_distributions[:,2],x\_i\_scaled['Skew'],'.')

plt.subplot('524');plt.plot(gauss\_samples\_from\_distributions[:,3],x\_i\_scaled['Kurtosis'],'.')

plt.subplot('525');plt.plot(gauss\_samples\_from\_distributions[:,4],x\_i\_scaled['Intensity of channel'],'.')

plt.show()



Будуємо гаусові моделі зображень з використанням розрахованих раніше параметрів.

gauss\_pdf = multivariate\_normal.pdf(gauss\_samples\_from\_distributions,std,cov\_diag)

plt.figure(5,figsize=(18,9))

plt.subplot('521');plt.plot(gauss\_samples\_from\_distributions[:,0],gauss\_pdf,'.')

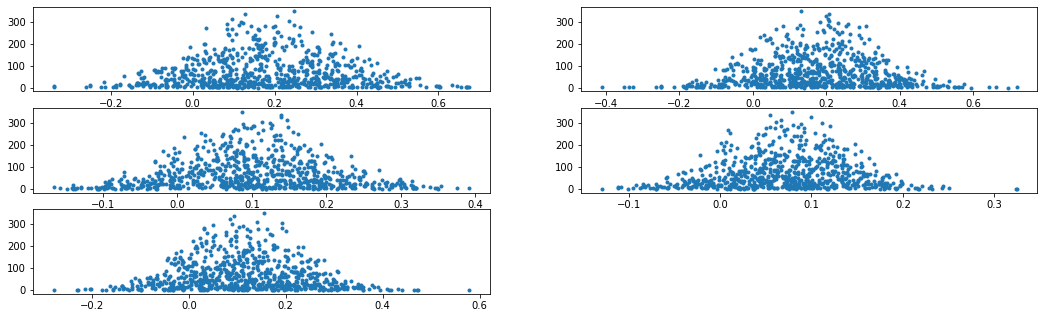
plt.subplot('522');plt.plot(gauss\_samples\_from\_distributions[:,1],gauss\_pdf,'.')

plt.subplot('523');plt.plot(gauss\_samples\_from\_distributions[:,2],gauss\_pdf,'.')

plt.subplot('524');plt.plot(gauss\_samples\_from\_distributions[:,3],gauss\_pdf,'.')

plt.subplot('525');plt.plot(gauss\_samples\_from\_distributions[:,4],gauss\_pdf,'.')

plt.show()



1. 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.

Уніфікуємо розміри зображень для PCA model (sklearn).

Вибираєм розміри найменшого зображення щоб мінімізувати втрату інформації .

images = Data['Img']

shape\_0 = min([i.shape[0] for i in images])

shape\_1 = min([i.shape[1] for i in images])

def resize(images,min\_x,min\_y):

resized = []

for i in images:

img\_rez = cv2.resize(i,(shape\_0,shape\_1),interpolation=cv2.INTER\_CUBIC)

resized.append(img\_rez)

return np.array(resized).reshape((-1,shape\_0,shape\_1))

перетворюємо набір в одновимірний до виду (n\_images,n\_features)

і стандартизуємо в межах 0 - 1

resized\_images = resize(images,shape\_0,shape\_1)

flatened\_images = resized\_images.reshape((750,shape\_0\*shape\_1))

standarting\_images = flatened\_images/255

def image\_PCA(images,energy):

model = PCA(energy)

model.fit(images)

tr\_images = model.transform(images)

return model,tr\_images

Будуємо репрезентації з вибраною енергією

image\_PCA(standarting\_images,200)

відбудовуємо деякий набір зображень із репрезентацій і вираховуємо MSE похибку між відбудованим і оригінальним

def mse(imageA, imageB):

err = np.sum((imageA.astype("float") - imageB.astype("float")) \*\* 2)

err /= float(imageA.shape[0] \* imageA.shape[1])

return err

some\_channel\_index = np.random.choice(750,50)

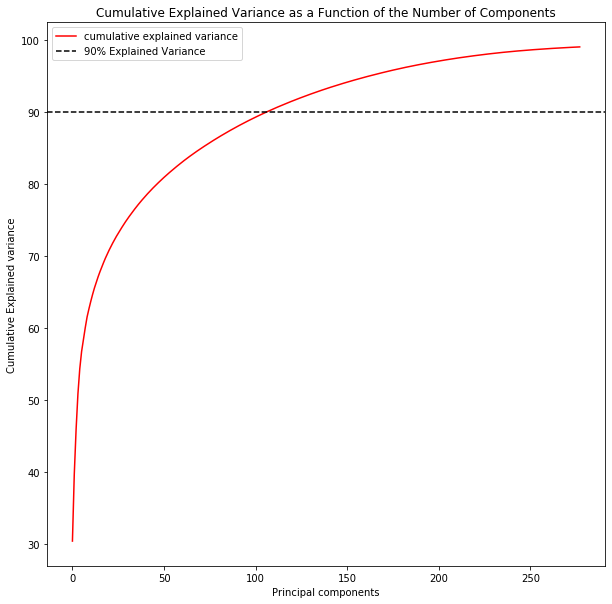
model,compressed\_img = image\_PCA(standarting\_images,.99)

reconstr = model.inverse\_transform(compressed\_img \* 255)

mse\_log = np.array(list(map(mse,reconstr.reshape((750,shape\_0,shape\_1)),standarting\_images.reshape((750,shape\_0,shape\_1)))))

print("Mean MSE - " ,np.mean(mse\_log))

Mean MSE - 5797.824853968864



Побудова графіку залежності MSE від кількості компонент

some\_channel\_index = np.random.choice(750,300)

some\_set = standarting\_images[some\_channel\_index]

some\_set.shape

n\_components = np.arange(50,300,25)

def func(n\_components,some\_set,shape\_0,shape\_1):

models\_data = []

compressed\_list\_by\_ncomponents = []

mse\_log\_list = []

mean\_mse = []

for i in n\_components:

model1,compressed\_images = image\_PCA(some\_set,i)

models\_data.append(model)

compressed\_list\_by\_ncomponents.append(compressed\_images)

reconstr = model1.inverse\_transform(compressed\_images)

mse\_log = np.array(list(map(mse,reconstr.reshape((300,shape\_0,shape\_1)),some\_set.reshape((300,shape\_0,shape\_1)))))

mse\_log\_list.append(mse\_log)

mean\_mse.append(np.mean(mse\_log))

return n\_components,np.array(mean\_mse),np.array(mse\_log\_list)

ar = func(n\_components,some\_set,shape\_0,shape\_1)

plt.figure(figsize=(10, 10))

plt.plot(ar[0],ar[1],color = 'red',label='dependence of mse from n\_components')

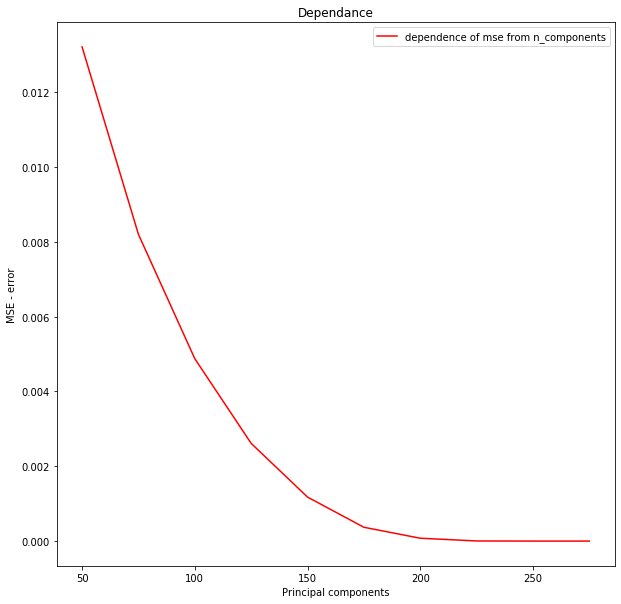
plt.title('Dependance')

plt.ylabel('MSE - error')

plt.xlabel('Principal components')

plt.legend(loc='best')

plt.show()



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.

Функція для розрахування стохастичної матриці марковського ланцюга 1 і 2 порядків

def transition\_matrix(arr, order):

\_shape = (max(arr) + 1,) \* (order + 1)

M = np.zeros(\_shape)

for \_ind in zip(\*[arr[\_x:] for \_x in range(order + 1)]):

M[\_ind] += 1

return np.array(M)

Функція для розрахування матриць для кожного каналу кожного зображення

def images\_tranition(images,order):

final = []

for i in images:

TM\_vert = []

TM\_hor = []

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

hor = np.array(list(reversed(i[x,:])))

hor\_tran = transition\_matrix(hor,order)

TM\_hor.append(hor\_tran)

for y in range(i.shape[1]):

vert = np.array(list(reversed(i[:,y])))

vert\_tran = transition\_matrix(vert,order)

TM\_vert.append(vert\_tran)

TM\_vert = np.array(TM\_vert)

TM\_hor = np.array(TM\_hor)

final.append(list((TM\_vert,TM\_hor)))

return np.array(final)

result\_1order = images\_tranition(test,1)# 1st order transition\_matrix

result\_2order = images\_tranition(test,2)# 2st order transition\_matrix

Приклад марковського ланцюга

states = (list(reversed(test[:,0])))

Q = transition\_matrix(states,1)

states = [tuple(i) for i in states]

G = nx.MultiDiGraph()

labels={}

edge\_labels={}

for i, origin\_state in enumerate(states):

for j, destination\_state in enumerate(states):

rate = Q[i][j]

if rate > 0:

G.add\_edge(origin\_state,

destination\_state,

weight=rate,

label="{:.02f}".format(rate))

edge\_labels[(origin\_state, destination\_state)] = label="{:.02f}".format(rate)

plt.figure(figsize=(14,7))

node\_size = 200

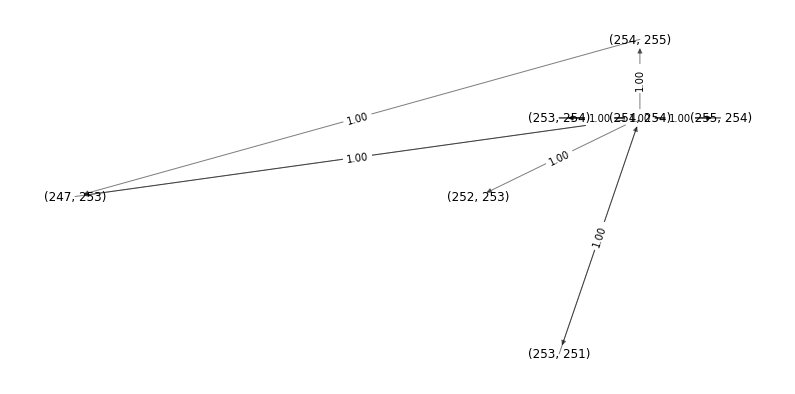
pos = {state:list(state) for state in states}

nx.draw\_networkx\_edges(G,pos,width=1.0,alpha=0.5)

nx.draw\_networkx\_labels(G, pos, font\_weight=2)

nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels)

plt.axis('off');



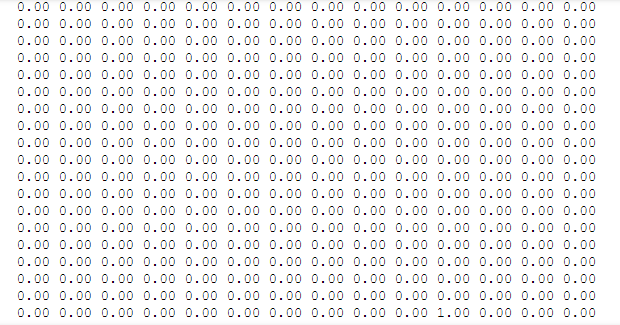
Перевірка гіпотез

test = np.array(list(reversed(Data["Img"][34][:,5])))

m = transition\_matrix(test,1)

m = np.array(m)

for row in m: print(' '.join('{0:.2f}'.format(x) for x in row))



Матриця не рекурентна ,регулярна і незворотня