МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних занять з дисципліни
«ТЕОРІЯ РОЗПІЗНАВАННЯ ОБРАЗІВ»

Одеса: Одеська політехніка, 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

МЕТОДИЧНІ ВКАЗІВКИ

до практичних занять з дисципліни
«ТЕОРІЯ РОЗПІЗНАВАННЯ ОБРАЗІВ»
для здобувачів першого (бакалаврського) рівня вищої освіти

за спеціальностями

124 – Системний аналіз

113 – Прикладна математика

Затверджено
на засіданні кафедри ПМІТ
Протокол № 1 від 30.08.2024 р.

Одеса: Одеська політехніка, 2025

Методичні рекомендації до лабораторних занять з дисципліни «Теорія розпізнавання образів» для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 124 – Системний аналіз (освітньо-професійна програма: «Системний аналіз та наука про дані») та за спеціальністю 113 – Прикладна математика (освітньо-професійна програма: «Математичне забезпечення комп'ютерних систем»)/ Укл.: *М. В. Полякова.* – Одеса: Одеська політехніка, 2025. – 32 с.

Укладач: **Полякова М. В.**, докт. техн. наук, доц.

Методичні вказівки містять теоретичні відомості, які необхідні і достатні для роботи на лабораторних заняттях з дисципліни «Теорія розпізнавання образів».

Призначаються для студентів денної форми навчання.

ЗМІСТ

Вступ

Методичні вказівки до лабораторних робіт з дисципліни «Теорія розпізнавання образів» призначені для систематизації знань і навичок студентів у галузі автоматизованого аналізу та класифікації зображень і сигналів. Розпізнавання образів є важливою складовою сучасних інформаційних технологій, застосовних у машинному зорі, біометрії, медичній діагностиці, робототехніці, системах штучного інтелекту та багатьох інших сферах.

Метою цих лабораторних робіт є закріплення теоретичних знань, ознайомлення з основними алгоритмами розпізнавання, а також розвиток практичних навичок застосування методів обробки даних і побудови моделей класифікації. В процесі виконання завдань студенти навчаться аналізувати вхідну інформацію, виділяти характерні ознаки, розробляти алгоритми класифікації та оцінювати їх ефективність.

Дані методичні вказівки допоможуть студентам краще розуміти основні принципи теорії розпізнавання образів та застосовувати набуті знання у практичній діяльності, що є важливим кроком на шляху формування фахівця у сфері сучасних інформаційних технологій.

Лабораторне заняття 1

Предобробка даних

Мета заняття: ознайомитися з методами предобробки даних з бібліотеки ScikitLearn.
Зміст заняття: побудова програми, яка виконує предобробку даних.

Теоретичні основи

Завантаження даних

Як приклад використовується множина даних, яка доступна за посиланням: https://www.kaggle.com/andrewmvd/heart-failure-clinical-data. Дані представлені як таблиця csv. Для того, щоб завантажити дані у датафрейм, використовується метод `read_csv`. Вилучення окремих ознак виконується методом `drop`.

Приклад 1.1.Завантаження даних у датафрейм, виключення бінарних ознак та ознаки часу.

```
import pandas as pd
import numpy as np
df=pd.read_csv('heart_failure_clinical_records_dataset.csv')
df=df.drop(columns=['anaemia','diabetes','high_blood_pressure','sex',
'smoking','time','DEATH_EVENT'])
#Виведення датафрейму з даними для лаб. роботи.
#Має бути 299 спостережень та 6 ознак
print(df)
```

Попередній аналіз окремих ознак

Побудова гістограм значень ознак виконується методом `hist`, для якого необхідно встановити параметр `n_bins` – число інтервалів гістограми.

Приклад 1.2. Побудова гістограм значень окремих ознак із числом інтервалів 20.

```
import matplotlib.pyplot as plt
n_bins = 20
fig, axs = plt.subplots(2,3)
axs[0, 0].hist(df['age'].values, bins = n_bins)
axs[0, 0].set_title('age')
axs[0, 1].hist(df['creatinine_phosphokinase'].values, bins = n_bins)
axs[0, 1].set_title('creatinine_phosphokinase')
axs[0, 2].hist(df['ejection_fraction'].values, bins = n_bins)
axs[0, 2].set_title('ejection_fraction')
axs[1, 0].hist(df['platelets'].values, bins = n_bins)
axs[1, 0].set_title('platelets')
axs[1, 1].hist(df['serum_creatinine'].values, bins = n_bins)
axs[1, 2].hist(df['serum_sodium'].values, bins = n_bins)
axs[1, 2].set_title('serum_sodium')
plt.show()
data = df.to_numpy(dtype='float')
```

Так як бібліотека Sklearn працює з NumPy масивами, то перетворили дані на двовимірний масив NumPy, де рядок відповідає спостереженню, а стовпець –ознаці.

Нормалізація даних

Методи нормалізації даних реалізовані у бібліотеці ScikitLearn. Для налаштування

нормалізації на основі частини спостережень використовується метод `fit`, для нормалізації даних – метод `transform`. Замість двох методів `fit` і `transform` можна застосувати метод `fit_transform`, щоб одразу налаштувати параметри нормалізації та перетворити дані.

Приклад 1.3. Стандартна нормалізація даних, яка налаштовується з урахуванням перших 150 спостережень. Для нормалізованих даних будується гістограма окремих ознак.

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(data[:150,:])
data_scaled = scaler.transform(data)
fig, axs = plt.subplots(2,3)
axs[0, 0].hist(data_scaled[:,0], bins = n_bins)
axs[0, 0].set_title('age')
axs[0, 1].hist(data_scaled[:,1], bins = n_bins)
axs[0, 1].set_title('creatinine_phosphokinase')
axs[0, 2].hist(data_scaled[:,2], bins = n_bins)
axs[0, 2].set_title('ejection_fraction')
axs[1, 0].hist(data_scaled[:,3], bins = n_bins)
axs[1, 0].set_title('platelets')
axs[1, 1].hist(data_scaled[:,4], bins = n_bins)
axs[1, 1].set_title('serum_creatinine')
axs[1, 2].hist(data_scaled[:,5], bins = n_bins)
axs[1, 2].set_title('serum_sodium')
plt.show()
```

Завдання для виконання роботи

1. Завантажити множину даних згідно з варіантом у датафрейм, виключити якісні ознаки, бінарні ознаки та ознаки часу, якщо такі є (приклад 1.1).
2. Побудувати гістограми ознак, що залишилися. На підставі гістограм визначити діапазони значень для кожної з ознак, а також біля якого значення лежить найбільша кількість спостережень (приклад 1.2). Обчислити середнє значення та середньоквадратичне відхилення для кожної з ознак.
3. Виконати нормалізацію даних методом, що зазначений у варіанті завдання (приклад 1.3). Побудувати гістограми ознак після нормалізації.
4. Визначити діапазони значень для кожної з ознак після нормалізації, а також визначити, біля якого значення лежить найбільша кількість спостережень. Розрахувати середнє значення та середньоквадратичне відхилення для кожної з ознак. Порівняти характеристики даних до та після нормалізації.
5. Наведіть формулу, яка використовувалася для нормалізації.
6. Провести налаштування нормалізації на всіх даних та порівняти з результатами налаштування на підставі 50% спостережень. Визначити, як змінилися середнє значення, середньоквадратичне відхилення, діапазон значень для кожної з ознак.

Таблиця 1.1 – Варіанти завдань

Варіант	Дані	Нормалізація
1	https://www.kaggle.com/datasets/gabrielsantello/cars-purchase-decision-dataset	StandartScaler
2	https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification	RobustScaler
3	https://www.kaggle.com/datasets/mohdnazuan/harumanis-mango-physical-measurement	MinMaxScaler
4	https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset	MaxAbsScaler
5	https://www.kaggle.com/datasets/gabrielsantello/cars-purchase-decision-dataset	Normalizer

6	https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification	StandartScaler
7	https://www.kaggle.com/datasets/mohdnazuan/harumanis-mango-physical-measurement	MaxAbsScaler
8	https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset	Normalizer
9	https://www.kaggle.com/datasets/gabrielsantello/cars-purchase-decision-dataset	MaxAbsScaler
10	https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification	MinMaxScaler
11	https://www.kaggle.com/datasets/mohdnazuan/harumanis-mango-physical-measurement	RobustScaler
12	https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset	RobustScaler
13	https://www.kaggle.com/datasets/gabrielsantello/cars-purchase-decision-dataset	StandartScaler
14	https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification	Normalizer
15	https://www.kaggle.com/datasets/mohdnazuan/harumanis-mango-physical-measurement	StandartScaler

Контрольні питання

1. Що таке предобробка даних і яку роль вона відіграє в системному аналізі?
2. Які основні етапи предобробки даних ви знаєте?
3. Які методи усунення пропущених даних використовуються на етапі предобробки?
4. Як здійснюється нормалізація або стандартизація даних і навіщо вона потрібна?
5. Що таке виявлення та видалення викидів у даних?
6. Які операції перетворення можна застосувати до текстових і числових даних?
7. Як працюють методи кодування категоріальних змінних?
8. Які інструменти або програмне забезпечення зазвичай використовують для обробки даних у лабораторних роботах?
9. Як оцінити якість даних після предобробки?
10. Чому важливо коректно і послідовно виконувати предобробку даних перед аналізом?

Лабораторне заняття 2

Класифікація даних байєсівськими методами

Мета заняття: ознайомитись з байєсівськими методами класифікації бібліотеки scikit-learn.

Зміст заняття: побудова програми, яка виконує байєсівську класифікацію даних за допомогою бібліотеки scikit-learn.

.

Теоретичні основи

Підготовка данних

Включає завантаження даних із файлу в таблицю об'єкт-ознака, виділення даних та їх міток, перетворення міток у числа.

Приклад 2.1. Підготовка данних.

```
#Завантаження даних у датафрейм
```

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
data = load_iris()

# Виділимо дані та їх мітки в масиви NumPy
X = data.data
labels = data.target

#Перетворимо тексти міток до чисел
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
Y = le.fit_transform(labels)
#Розіб'ємо вибірку на навчальну та тестову
from sklearn.model_selection import train_test_split
X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  Y,
test_size=0.5)
```

Байєсівські методи класифікації даних

Наївні байєсовські класифікатори є сімейством класифікаторів, які схожі з лінійними моделями, наприклад, лінійною машиною опорних векторів. Однак байєсівські класифікатори навчаються швидше, але мають нижчу узагальнюючу здатність порівняно з лінійними класифікаторами типу логістичної регресії або лінійної машини опорних векторів. Причина, через яку наївні байєсовські моделі настільки ефективні, у тому, що вони розглядають кожну ознаку окремо й за кожною ознакою обчислюють статистики класів. У scikit-learn реалізовано чотири наївні байєсовські класифікатори: GaussianNB, ComplementNB, BernoulliNB і MultinomialNB. GaussianNB можна застосувати до будь-яких безперервних даних, у той час як BernoulliNB приймає бінарні дані, MultinomialNB приймає лічильні чи дискретні дані (тобто кожна ознака є підрахунок цілочисельних значень якоїсь характеристики, наприклад, частоти слова у реченні). BernoulliNB та MultinomialNB в основному використовуються для класифікації текстових даних. Класифікатор BernoulliNB підраховує ненульові частоти ознак кожного класу. MultinomialNB і GaussianNB відрізняються з погляду обчислюваних статистик. MultinomialNB розраховує середнє значення кожної ознаки кожного класу, тоді як GaussianNB записує середнє значення, і навіть стандартне відхилення кожної ознаки кожного класу. При класифікації вектор ознак об'єкта порівнюється зі статистиками кожного класу і вибирається найбільш підходящий клас.

ComplementNB є удосконаленням поліноміального наївного байєсівського методу MultinomialNB для незбалансованих наборів даних, тобто даних, у яких кількість об'єктів різних класів суттєво різниться. Зокрема, ComlementNB використовує статистику доповнення кожного класу для обчислення ваг моделі. Емпірично показано, що оцінки параметрів для ComplementNB більш стабільні, ніж для MumtinomialNB. Крім того, ComplementNB часто значно перевершує MultinomialNB за якістю класифікації у завданнях обробки текстів.

Параметр coef_ для наївних байєсівських методів має інший зміст, ніж coef_ для лінійних моделей, тут coef_ не тотожний вектору вагових коефіцієнтів w.

MultinomialNB і BernoulliNB мають ще параметр alpha, який контролює складність методу: до даних додається в залежності від значення alpha певна кількість штучних спостережень з позитивними значеннями всім ознак. Це призводить до "згладжування" статистик. Більше значення alpha означає більш високий рівень згладжування, що призводить до побудови менш складного класифікатора. Однак наївні байєсовські класифікатори стійкі до різних значень alpha, тобто його значення особливо не впливає на якість класифікації, але може трохи збільшити її при відповідному налаштуванні.

Класифікатор GaussianNB в основному використовується для даних з високою розмірністю, тоді як інші наївні методи Байєса широко використовуються для розріджених дискретних даних, наприклад, для тексту. MultinomialNB зазвичай працює краще, ніж BernoulliNB, особливо на наборах даних з відносно великою кількістю ознак, що мають ненульові частоти (тобто великі документи).

Наївні методи Байєса, як і лінійні моделі, швидко навчаються, процес навчання легко інтерпретувати. Працюють із високорозмірними розрідженими даними та відносно стійкі до змін параметрів. Використовуються на великих наборах даних, де навчання навіть лінійної моделі може забрати багато часу.

Приклад 2.2. Проведемо класифікацію спостережень наївним байєсівським методом.

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
#кількість спостережень, які були неправильно класифіковані
print((y_test != y_pred).sum())
```

Завдання для виконання роботи

1. Завантажити множину даних згідно з варіантом (табл. 2.1).
2. Нормалізувати ці дані також згідно з варіантом. Навести формулу нормалізації.
3. Відобразити на координатній площині двох вибраних ознак по 10 об'єктів кожного класу у вигляді точок певної форми та кольору.
4. Описати особливості використаного згідно з варіантом байєсівського класифікатора.
5. Розбити дані на навчальну та тестову вибірку. Виконати навчання байєсівського класифікатора згідно з варіантом на навчальній вибірці.
6. Виконати класифікацію спостережень тестової вибірки. Побудувати графік залежності неправильно класифікованих спостережень від розміру тестової вибірки. Розмір тестової вибірки змінюватиметься від 0.05 до 0.5 з кроком 0.05. Обґрунтувати отримані результати.

Таблиця 2.1 - Варіанти завдань на лабораторну роботу

Варіант	Дані для класифікації	Нормалізація	Байєсівський класифікатор
1	`load_wine()`	`MaxAbsScaler`	BernoulliNB
2	`load_breast_cancer()`	`StandartScaler`	GaussianNB
3	`load_iris()`	`RobustScaler`	MultinomialNB
4	`load_wine()`	`MinMaxScaler`	ComplementNB
5	`load_breast_cancer()`	`RobustScaler`	BernoulliNB
6	`load_iris()`	`MaxAbsScaler`	GaussianNB
7	`load_wine()`	`StandartScaler`	MultinomialNB
8	`load_breast_cancer()`	`MinMaxScaler`	ComplementNB
9	`load_iris()`	`StandartScaler`	BernoulliNB
10	`load_wine()`	`RobustScaler`	GaussianNB
11	`load_breast_cancer()`	MaxAbsScaler	MultinomialNB
12	`load_iris()`	`RobustScaler`	ComplementNB
13	`load_wine()`	`MinMaxScaler`	BernoulliNB

Контрольні питання

1. Як завантажити множину точок для класифікації?
2. Як виконується класифікація наївним байєсівським методом?
3. Як розбити вибірку на навчальну та тестову?
4. Як під час вирішення практичних завдань оцінити якість класифікації?

Лабораторне заняття 3

Методи контурної сегментації півтонових зображень

Мета заняття: ознайомитися з методами контурної сегментації зображень із бібліотеки scikit-image.

Зміст заняття: вивчення функцій бібліотеки scikit-image для контурної сегментації зображень на Python та набуття практичних навичок їх використання.

Теоретичні основи

Бібліотека scikit-image дозволяє завантажувати зображення із зовнішніх джерел, які зберігаються у форматах jpeg, bmp, png та ін.

Приклад 3.1. Завантаження зображення із зовнішнього файлу (рис. 3.1, а). Перетворення зображення на відтінки сірого за допомогою rgb2gray.

```
# import the image
from skimage import io
image = io.imread('room.jpg')
plt.imshow(image);
from skimage.color import rgb2gray
image_gray = color.rgb2gray(image)
image_show(image_gray);
```

Сегментація зображень та її застосування

Сегментація зображень застосовується в робототехніці, системах розпізнавання обличь, в медицині для діагностики захворювань за результатами рентгенографії, МРТ, термографії, ендоскопії, ультразвукового дослідження клітин і тканин.

Під сегментацією розуміють процес поділу зображення різні об'єкти. Пікселі однієї області (об'єкти зображення) схожі за якоюсь ознакою, наприклад, кольором, інтенсивністю, розташуванням або текстурою, що дозволяє знижувати складність зображення, спрощуючи його аналіз. При побудові методів сегментації використовуються дві основні властивості зображення: 1) зміна ознак сегментації, наприклад перепад інтенсивності (контурна сегментація – edge detection); 2) схожість ознак сегментації, наприклад, схожість кольору чи текстури.

Рисунок 3.1 - Початкове зображення (а), результат семантичної сегментації (б), результат сегментації об'єктів.

На рис. 3.1, а зображені стільці, стіл, вікно. Виділимо дані об'єкти за допомогою сегментації. На рис. 3.1 б у результаті семантичної сегментації отримуємо окремо виділені

об'єкти: стілець, стіл і вікно. На рис. 3.1, застосували сегментацію об'єктів.

Метод Канні контурної сегментації напівтонових зображень

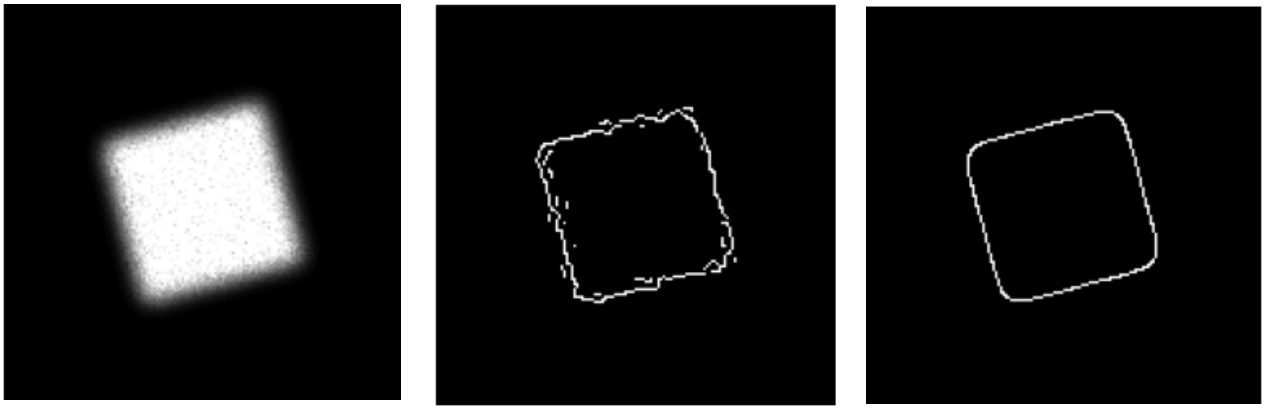
Виконання контурної сегментації зображень передбачає, що на границях об'єктів значення інтенсивності значно відрізняються один від одного, що дозволяє виявляти їх за локальними неоднорідностями інтенсивності (градацій сірого). Границі об'єктів мають важливу змістовну інформацію, що використовується при розпізнаванні.

Метод Канні є багатоступінчастим детектором контурів. Він використовує фільтр, заснований на похідній Гаусса, щоб обчислити інтенсивність градієнтів. Гауссіан зменшує вплив шуму, що є на зображенні. Потім потенційні контури проріджуються до однопіксельних кривих шляхом видалення немаксимальних значень величини градієнта. Нарешті, пікселі контуру зберігаються або видаляються з використанням гістерезиса для величини градієнта.

Таким чином, у методі Кані виробляється таке:

1) згладжування значень інтенсивності зображення за допомогою гаусівського фільтра з метою зменшення адитивних флуктуаційних завад;
2) оцінка градієнта зображення як квадратного кореня із суми квадратів похідних за двома ортогональними напрямками з метою підкреслення перепадів інтенсивності зображення;
3) порогова обробка оцінки градієнта значень інтенсивності зображення, яка включає немаксимальне пригнічення перепадів інтенсивності зображення, що полягає в наступному: величина перепаду інтенсивності в кожній точці контуру дорівнюється нулю, якщо вона не перевищує величину перепаду інтенсивності у двох сусідніх точках у напрямку градієнта зображення;
4) порівняння величини інтенсивності в отриманих точках контуру з двома попередньо заданими порогами значення перепаду інтенсивності;
5) гістерезис: точки, величина інтенсивності у яких вище меншого за величиною порога, приєднуються до сусідніх точок, величина інтенсивності в яких перевищує більший за величиною поріг.

Приклад 3.2. Сегментація синтетичного зображення методом Кані (рис. 3.2). Для функції `feature.canny()` задаються три регульовані параметри: параметр фільтру Гауса (чим зашумленіше зображення, тим більше параметр), а також нижній і верхній пороги гістерезиса.



а б в

Рисунок 3.2 - Початкове зображення зі спекл-шумом (а), результат сегментації методом Кані з σ=1 (б), результат сегментації методом Кані з σ=3 (в).

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
```

```
from skimage.util importrandom_noise
from skimage import feature
# Generate noisy image of a square
image=np.zeros((128, 128), dtype = float)
image[32:-32, 32:-32] = 1

image=ndi.rotate(image, 15, mode='constant')
image=ndi.gaussian_filter(image, 4)
image=random_noise(image, mode = 'speckle', mean = 0.1)

# Compute the Canny filter for two values of sigma
edges1=feature.canny(image)
edges2=feature.canny(image, sigma = 3)
# display results
fig,ax=plt.subplots(nrows = 1, ncols = 3, figsize = (8, 3))
ax[0].imshow(imagecmap = 'gray')
ax[0].set_title('noisy image', fontsize=20)
ax[1].edges1cmap = 'gray')
ax[1].set_title(r'Canny filter, $\sigma=1$', fontsize=20)
ax[2].edges2cmap = 'gray')
ax[2].set_title(r'Canny filter, $\sigma=3$', fontsize=20)
for a inax:
 a.axis('off')
fig.tight_layout()
plt.show()
```

Оцінка якості контурної сегментації

При оцінці якості контурної сегментації враховувалися три основні види помилок щодо положення перепадів інтенсивності: пропуск справжніх перепадів, помилка у визначенні положення, прийняття шумових викидів за перепад. Як показник якості контурної сегментації використовується критерій Претта (Figure Of Merit – FOM):

$$R = \frac{1}{I}\sum_{i=1}^{I_A}\frac{1}{1+\alpha d_i^2},$$

де I=max(I_I, I_A); I_I, I_A – кількість точок перепадів інтенсивності в контурах, отриманих відповідно за допомогою ідеального та реального детекторів; α – масштабний множник; d_i — відстань між точкою дійсного перепаду та лінією з точок ідеального перепаду, виміряного за нормаллю до цієї лінії. Значення критерію Претта нормалізовано, отже R =1 для точно виділеного перепаду. Множник 1/I характеризує розбиті контури.

Показником якості контурної сегментації зображення також обрано показник близькості між границями тестового ідеально сегментованого зображення I_0 та сегментованого досліджуваним методом обробки I:

$$СКО = \frac{1}{I_A}\sum_{x=1}^{m}\sum_{y=1}^{n}(I_0(x,y) - I(x,y)),$$

де I_A – довжина границь виділених сегментів у пікселах; m, n – розміри зображення.

Приклад 3.3. Обчислення показника якості контурної сегментації – значення критерію Претта між зображеннями `edges1, edges2`.

```
from scipy.ndimage import distance_transform_edt
DEFAULT_ALPHA = 1.0/9
def fom(edges_img, edges_gold, alpha = DEFAULT_ALPHA):
```

```
 #Computes Pratt's Figure of Merit for given edges_img and gold
standard #image edges_gold
# Compute distance transform for gold standard image.
dist = distance_transform_edt(np.invert(edges_gold))

fom = 1.0 / np.maximum(
np.count_nonzero(edges_img),
np.count_nonzero(edges_gold))
N, M = edges_img.shape

for i in range(0, N):
for j in range(0, M):
if edges_img[i, j]:
fom + = 1.0 / (1.0 + dist [i, j] * dist [i, j] * alpha)
fom /= np.maximum(
np.count_nonzero(edges_img),
np.count_nonzero(edges_gold))
return fom
print(fom(edges1, edges2, alpha = DEFAULT_ALPHA))
```

Завдання для виконання роботи

1. Підібрати три кольорові зображення у цифровому форматі, завантажити файли із цими зображеннями. Перетворити зображення на відтінки сірого за допомогою функції rgb2gray (приклад 3.1).
2. Виконати сегментацію досліджуваних зображень методом Кані, налаштувавши параметр σ та пороги – верхній та нижній (приклад 3.2). Навести зображення вхідні та зображення, отримані в результаті сегментації (рис. 3.2).
3. Додати шум на вхідні зображення у відтінках сірого. Тип шуму визначається варіантом (табл. 3.1). Параметри шуму вибрати самостійно. Привести тип шуму та його параметри у звіті. Охарактеризувати доданий шум (що він являє собою).
4. Виконати сегментацію зашумлених зображень методом Кані, налаштувавши параметр σ та пороги – верхній та нижній (приклад 3.2). Навести зображення зашумлені та зображення, отримані в результаті сегментації (рис. 3.2).
5. Оцінити якість сегментації, порівнявши результати сегментації вхідних та зашумлених зображень методом Кані. Для цього обчислити значення критерію Претта та середньоквадратичної помилки. Порівняти та проаналізувати отримані значення.

Таблиця 3.1. – Варіанти завдань

Варіант	Тип шуму	Варіант	Тип шуму
1	'gaussian'	9	'speckle'
2	's&p'	10	'salt'
3	'speckle'	11	'pepper'
4	'gaussian'	12	'salt'
5	's&p'	13	'pepper'
6	'speckle'	14	'gaussian'
7	'pepper'	15	'salt'
8	's&p'	16	's&p'

Контрольні питання

1. Як перетворити зображення у відтінки сірого?
2. За яким принципом виконується сегментація методом Кані?
3. Як оцінити якість контурної сегментації?
4. Як при вирішенні практичних завдань налаштувати параметри методу Кані?

Лабораторне заняття 4

Інтерактивні методи сегментації півтонових зображень

Мета заняття: ознайомитись з інтерактивними методами сегментації зображень із бібліотеки scikit-image.

Зміст заняття: вивчення функцій бібліотеки scikit-image для інтерактивної сегментації зображень на Python та набуття практичних навичок їх використання.

Теоретичні основи

Сегментація зображення є етапом обробки зображення, на якому виконується розбиття зображення на області за однією або декількома ознаками, що характеризують ці області. Це активна галузь досліджень з різними додатками, починаючи від комп'ютерного зору та закінчуючи медичними зображеннями, дорожнім рухом та відеоспостереженням. Python надає бібліотеку scikit-image, в якій, зокрема, реалізовані методи сегментації зображень.

Завантаження даних

Бібліотека scikit-image містить вбудовані зображення, які зазвичай зберігаються у форматі jpeg або png.

Приклад 4.1. Завантаження зображень. Перетворення зображення на відтінки сірого за допомогою rgb2gray (рис. 4.1, а).

```
# import the image
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from skimage import data
img = data.astronaut()
img = rgb2gray(img)
plt.imshow(img, cmap=plt.cm.gray)
plt.show()
```

Сегментація методом активних контурів

Активний контур, він також називається змійкою, ініціалізується шляхом визначення користувачем контуру (лінії) навколо області, що цікавить, а потім цей контур повільно стискається і притягується до контурів об'єкта, що цікавить.

Приклад 4.2. Сегментація зображення методом активних контурів.

```
import numpy as np
from skimage.filters import gaussian
from skimage.segmentation import active_contour
s = np.linspace(0, 2*np.pi, 400)
r = 100 + 100*np.sin(s)
c = 220 + 100 * np.cos (s)
```

```
init = np.array([r, c]).T
snake = active_contour(gaussian(img, 3, preserve_range=False),
init, alpha=0.015, beta=10, gamma=0.001)
fig, ax = plt.subplots(figsize=(7, 7))
ax.imshow(img, cmap=plt.cm.gray)
ax.plot(init[:, 1], init[:, 0], '--r', lw=3)
ax.plot(snake[:, 1], snake[:, 0], '-b', lw=3)
ax.set_xticks([]), ax.set_yticks([])
ax.axis([0, img.shape[1], img.shape[0], 0])
plt.show()
```

Приклад 4.3. Створення функції circle_points() для того, щоб намалювати коло навкруги голови людини та ініціалізувати змійку (рис. 4.1, б).

```
def circle_points(resolution, center, radius):
# Generate points which define a circle on an image. Center refers to the
# center of the circle
     radians = np.linspace(0, 2*np.pi, resolution)
     c = center[1] + radius*np.cos(radians)#polar co-ordinates
     r = center[0] + radius*np.sin(radians)
     #  Exclude  last  point  because  a  closed  path  should  not  have
duplicate # points
     return np.array([c, r]).T

#Обчислення координат x і y точок кола. Оскільки задали resolution
# 400, отримуємо 400 таких точок.
points = circle_points(400, [100, 220], 100)[:-1]
plt.imshow(img, cmap=plt.cm.gray)
plt.plot(points[:, 0], points[:, 1], '--r', lw=3)
plt.show()
```

У процесі сегментації шляхом активних контурів початкова замкнута крива підганяється до контурів обличчя людини, відокремлюючи його від решти зображення (рис. 4.1, в). Параметри alpha та beta призначені для налаштування методу сегментації. Вищі значення alpha змушують вихідну криву швидше наближатися до контуру об'єкта, тоді як збільшення beta робить криву більш гладкою (рис 4.1, г).

```
snake = active_contour(gaussian(img, 3, preserve_range=False),
init, alpha=0.015, beta=0.010, gamma=0.001)
fig, ax = plt.subplots(figsize=(7, 7))
ax.imshow(img, cmap=plt.cm.gray)
ax.plot(init[:, 1], init[:, 0], '--r', lw=3)
ax.plot(snake[:, 1], snake[:, 0], '-b', lw=3)
ax.set_xticks([]), ax.set_yticks([])
ax.axis([0, img.shape[1], img.shape[0], 0])
plt.show()
```

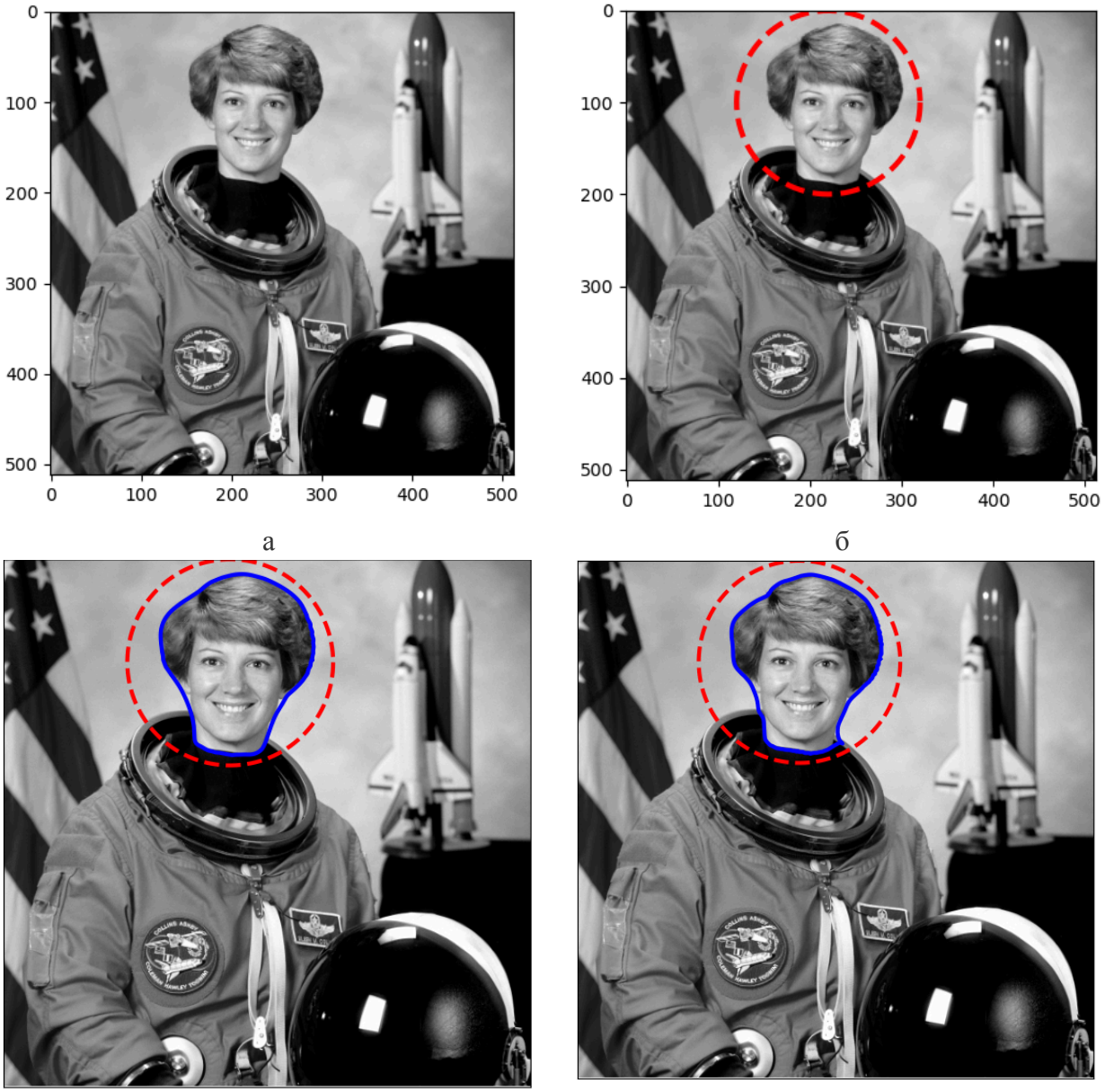


Рисунок 4.1 - Початкове зображення у відтінках сірого (а), ініціалізація активного контуру (б), результат сегментації методом активних контурів з різними значеннями бета (10 та 0,01)(в, г).

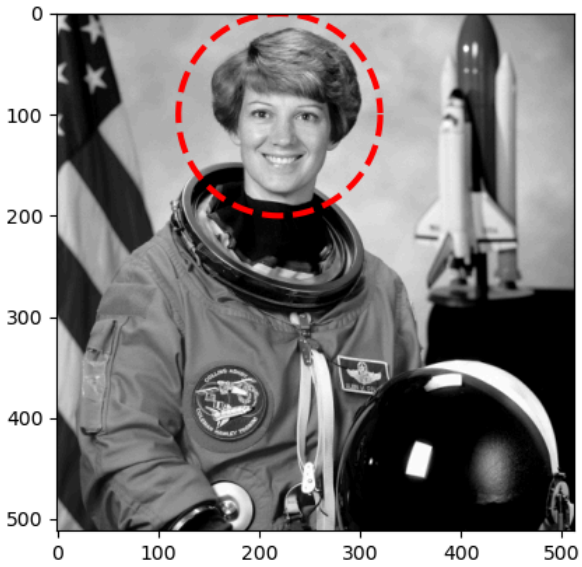
Сегментація методом випадкового блукання

Відповідно до методу випадкового блукання (англ. Random walker segmentation) користувач інтерактивно маркує невелику кількість пікселів відомими мітками (називаються початковими числами), наприклад, «об'єкт» і «фон». Передбачається, що кожен немаркований піксель вивільняє випадково блукаючий елемент. Для кожного немаркованого пікселя обчислюється ймовірність того, що відповідний йому випадково блукаючий елемент досягне початкового числа із заданою міткою. Ці ймовірності визначаються аналітично шляхом розв'язання системи лінійних рівнянь. Після обчислення цих ймовірностей для кожного немаркованого пікселя, цьому пікселю надається мітка класу, якого, швидше за все, досягне випадково блукаючий елемент. Це позначка початкового числа, якому відповідає найбільше значення ймовірності. Зображення моделюється як граф, у якому кожен піксель відповідає вершині, яка ребрами з'єднана з сусідніми пікселями-вершинами, причому ребра зважуються для відображення подібності між пікселями. Отже, випадкове блукання відбувається на зваженому графі.

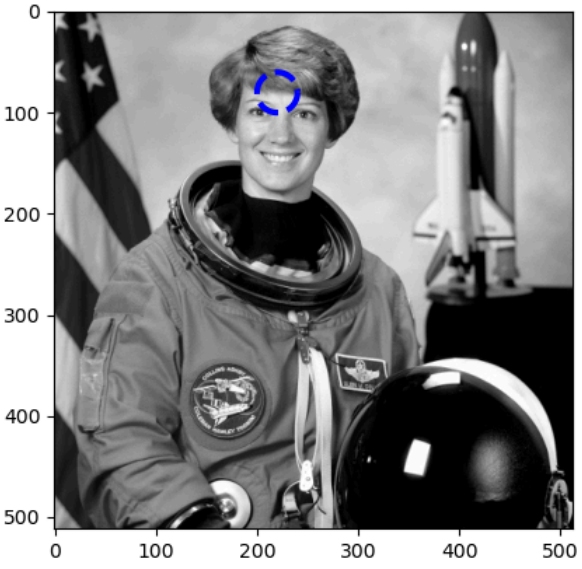
Приклад 4.4. Сегментація методом випадкового блукання. Функція random_walker(), що виконує сегментацію зображення методом випадкового блукання, приймає мітки як вхідні параметри. У прикладі як позначення фону використовується велике коло (рис. 4.2, а), що охоплює все обличчя людини, а як позначення об'єкту – ще одне менше коло біля середини обличчя (рис. 4.2, б).

```
#Обчислення координат x і y точок кола. Оскільки задали resolution
# 400, отримуємо 400 таких точок.
points = circle_points(400, [100, 220], 100)[:-1]
plt.imshow(img, cmap=plt.cm.gray)
plt.plot(points[:, 0], points[:, 1], '--r', lw=3)
plt.show()
image_labels = np.zeros(img.shape, dtype=np.uint8)
image_labels[points[:, 1].astype(np.int), points[:,0].astype(np.int)] =
2
points = circle_points(400, [80, 250], 20)[:-1]
plt.imshow(img, cmap=plt.cm.gray)
plt.plot(points[:, 0], points[:, 1], '--b', lw=3)
plt.show()
image_labels[points[:, 1].astype(np.int), points[:,0].astype(np.int)] =
1
plt.imshow(image_labels);
plt.show()
image_segmented = random_walker(img, image_labels, mode='bf')
# Check our results
fig, ax = plt.subplots(figsize=(7, 7))
plt.imshow(img, cmap=plt.cm.gray)
ax.imshow(image_segmented == 1, alpha=0.2, cmap=plt.cm.gray);
plt.show()
```

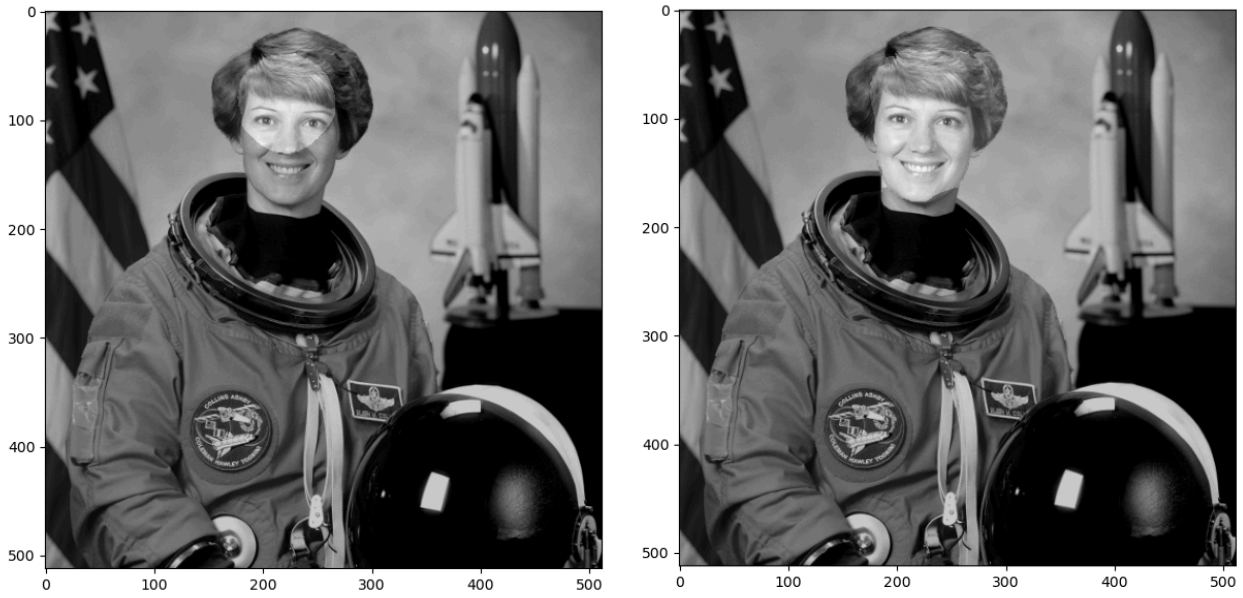
З використанням значення beta за умовчанням, можна отримати результат не найкращий, наприклад, на рис. 4.2, в залишилися неохопленими контуром края обличчя. Щоб виправити цю ситуацію, параметр beta налаштовувався, доки не було отримано бажаного результату. Після кількох спроб встановлено значення 1000, для якого отримано досить коректний результат (рис. 4.2, г).



а



б



в г

Рисунок 4.2 - Ініціалізація методу випадкового блукання (а, б), результат сегментації методом випадкового блукання з різними значеннями бета (за умовчанням та 1000) (в, г).

Завдання для виконання роботи

1. Отримати вихідне кольорове зображення – фотографію свого обличчя, постаратися забезпечити рівномірне висвітлення як на рис. 4.1, а.
2. Завантажити отримане зображення обличчя. Перетворити зображення на відтінки сірого за допомогою функції rgb2gray (приклад 4.1).
3. Виконати сегментацію обличчя за допомогою активних контурів, налаштувавши параметри alpha і beta (приклад 4.2). Навести зображення з початковим наближенням та отримане в результаті сегментації зображення (рис. 4.1, в, г).
4. Виконати сегментацію обличчя методом випадкового блукання, налаштувавши параметр beta (приклади 4.3, 4.4). Привести зображення з початковим наближенням та отримане в результаті сегментації зображення (рис. 4.2).

Контрольні питання

1. Як створити множину точок для ініціалізації контуру?
2. За яким принципом виконується сегментація методом активних контурів?
3. За яким принципом виконується сегментація методом випадкового блукання?
4. Як під час вирішення практичних завдань налаштувати параметри інтерактивних методів сегментації?

Лабораторне заняття 5

Методи сегментації півтонових зображень із використанням кластеризації

Мета заняття: ознайомитись із методами сегментації зображень з використанням кластеризації з бібліотеки scikit-image.

Зміст заняття: побудувати програму для сегментації зображень з використанням кластеризації з бібліотеки scikit-image.

Теоретичні основи

Сегментація методом SLIC

Основна ідея методу сегментації SLIC полягає в кластеризації пікселів в обмежених областях, на які регулярно розбивається аналізоване зображення. Кожна точка зображення характеризується п'ятивимірним вектором $p = (c_1, c_2, c_3, x, y)$, де c_1, c_2, c_3 – координати точки у вибраному колірному просторі; x та y – просторові координати точки зображення. Метод включає такі кроки.

1. Зображення розбивається на K фрагментів розміру $a \times a$, які задають початкове наближення кластерів-суперпікселів. Як початкові центри суперпіксельних фрагментів вибираються їх центри важкості C_k.

2. Коригуються координати центрів фрагментів з умови мінімального значення колірного градієнта в 3 × 3 околиці геометричного центру.

3. Формування локальних кластерів у $2a \times 2a$ околиці центрів C_k методом k-середніх. Відстань D між центром і точками фрагмента обчислюється як комбінація евклідових відстаней за колірною d_c та просторовою d_s складовими ознак точки p:

$$d_c^2 = (c_{j1} - c_{i1})^2 + (c_{j2} - c_{i2})^2 + (c_{j3} - c_{i3})^2 ;$$

$$d_s^2 = (x_j - x_i)^2 + (y_j - y_i)^2 ;$$

$$D = (d_c^2 + d_s^2 (m/a)^2)^{1/2} ,$$

де m – параметр, що задає відношення вкладів двох складових опису зображення у величину відстані D; i та j – номери точок, між якими обчислюється відстань.

4. Визначення нових центрів кластерів та обчислення зсувів центрів.

5. Повтор кроків 3 і 4, доки зміщення центрів між ітераціями не стане меншим за задане значення. Щоб виділити однорідні області, які відповідають об'єктам, зафіксованим на зображенні, необхідно об'єднати окремі суперпіксельні кластери.

Двоетапна постобробка

Метою постобробки є об'єднання отриманих суперпікселів у однорідні області, що відповідають об'єктам вихідного зображення. На першому етапі поєднуються сусідні суперпікселі. Для прийняття рішення про об'єднання використовується граничне вирішальне правило, що дозволяє об'єднання, якщо виконується нерівність:

$$d(C_i, C_j) \le \Delta_1 ; \qquad (5.1)$$

где $d(C_i, C_j) = ((c_{j1} - c_{i1})^2 + (c_{j2} - c_{i2})^2 + (c_{j3} - c_{i3})^2)^{1/2}$ – відстань між центрами сусідніх суперпікселів з номерами i та j у вибраному колірному просторі; c_{i1}, c_{i2} та c_{i3} – координати центру i; Δ_1 – граничне значення.

На другому етапі поєднуються суперпіксельні кластери в межах всього зображення. Для прийняття рішення про об'єднання, так само як і на першому етапі, використовується граничне вирішальне правило, що дозволяє об'єднання, якщо виконується нерівність:

$$d(C_i, C_j) \le \Delta_2, ; \qquad (5.2)$$

где Δ_2 – граничне значення.

Тоді постобробка включає такі кроки: (а) перегляд масиву центрів суперпікселів зображення та формування матриці об'єднання сусідніх суперпікселів за правилом (5.1); (б) об'єднання сусідніх суперпікселів; (в) знаходження центрів нових кластерів; (г) перегляд

масиву центрів кластерів для формування матриці об'єднання суперпіксельних кластерів за правилом (5.2); (д) об'єднання подібних суперпіксельних кластерів.

Метод суперпіксельної сегментації SLIC містить чотири параметри: початкова довжина сторони суперпіксельної області *a*, параметр *m*, що задає співвідношення вкладів колірної та просторової складових ознак точок зображення у величину відстані *D* та два порогових значення Δ_1 та Δ_2 у вирішальних правилах (5.1) та (5.2)).

Приклад 5.1.Сегментація зображення методомSLIC.

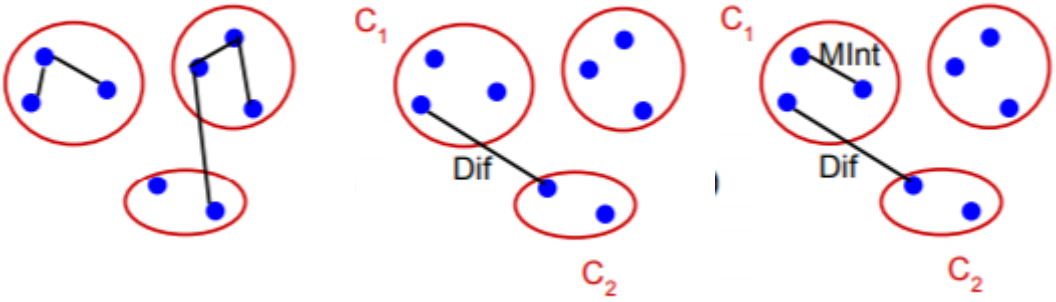
```
from skimage.data import astronaut
from skimage.color import rgb2gray, label2rgb
from skimage.segmentation import slic
from skimage.segmentation import mark_boundaries
from skimage.util import img_as_float
import numpy as np

img = img_as_float(astronaut()[::2, ::2])

segments_slic50 = slic(img, n_segments=50, compactness=10, sigma=1,
start_label=1)
segments_slic100 = slic(img, n_segments=100, compactness=10, sigma=1,
start_label=1)

fig, ax = plt.subplots(2, 2, figsize=(10, 10), sharex=True, sharey=True)
ax[0, 0].imshow(mark_boundaries(img, segments_slic50))
ax[0, 0].set_title('SLIC boundaries with n_segments=50')
# label2rgb replaces each discrete label with average interior color
ax[0, 1].imshow(label2rgb(segments_slic50, img, kind='avg',bg_label=0));
ax[0, 1].set_title('SLIC regions with n_segments=50')
ax[1, 0].imshow(mark_boundaries(img, segments_slic100))
ax[1, 0].set_title('SLIC boundaries with n_segments=100')
ax[1, 1].imshow(label2rgb(segments_slic100, img, kind='avg',bg_label=0))
ax[1, 1].set_title('SLIC regions with n_segments=100')

for a in ax.ravel():
a.set_axis_off()
plt.tight_layout()
plt.show()
```

Графовий метод сегментації зображень Фельценшвальба та Хаттенлохера

Більшість існуючих методів сегментації зображень при високій якості результатів працюють повільно і не враховують, що об'єкт може бути інваріантним за інтенсивністю і, отже, неправильно сегментують цю область (рис. 5.1). Тому Фельценшвальбом і Хаттенлохером було розроблено метод сегментації зображень з урахуванням графів, що виконує розбиття зображення на області, які відбивають глобальні аспекти зображення, практично за лінійний час за кількістю пікселів зображення.

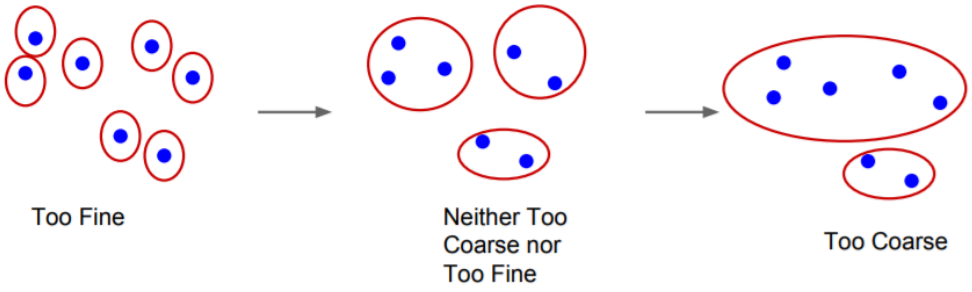
Рисунок 1.1 - Початкове зображення (а), результат неправильної сегментації (б), результат правильної сегментації (в).

Завдання сегментації зображень за допомогою теорії графів формулювалося так. Нехай $G=(V, E)$ – неорієнтований граф з вершинами $v_i \in V$ і ребрами $(v_i, v_j) \in E$, які відповідають парам сусідніх вершин. Кожному ребру $(v_i, v_j) \in E$ відповідає вага $w(v_i, v_j)$, що є невід'ємною мірою відмінності сусідніх вершин v_i, v_j. Для завдання сегментації зображення вершинами є пікселі, а вага ребра – це певна міра відмінності між двома пікселями, з'єднаними цим ребром. Наприклад, це може бути різниця значень інтенсивності, кольору, руху, координат пікселів або будь-якої іншої локальної ознаки. Тоді сегментація зображення S розбиває граф G на множину підграфів $C_1, ..., C_r$ (рис. 5.2, а).



а б в

г



д

Рисунок 5.2 - Сегментація зображення розбиває граф на множину підграфів (а), різниця між двома підграфами Dif (б), внутрішня різниця підграфа (в), результат сегментації залежить від значення параметра k (г), сегментація може включати більш або менш дрібні області (д).

Було розглянуто два підходи до побудови графа для зображення.

1. Граф решітки: кожен піксель пов'язаний тільки з сусідами (всього 8 інших пікселів). Вага ребра – це абсолютна різниця між значеннями інтенсивності пікселів.
2. Граф найближчих сусідів: кожен піксель є точкою у просторі ознак (x, y, r, g, b), у якій (x, y) – це координати пікселя, а (r, g, b) – значення кольору у колірному просторі RGB. Вага ребра – це евклідова відстань між векторами ознак двох пікселів.

Для розбиття вихідного графа на підграфи вводяться наступні поняття.

Внутрішня різниця – це максимальна вага ребра, що з'єднує дві вершини одного і того ж підграфа. Різниця між двома подграфами – мінімальна вага ребра, що з'єднує вершину v_i підграфа C_1 і вершину v_j підграфа C_2: $\mathrm{Dif}(C_1,C_2)=\min\{w(v_i,v_j), (v_i,v_j)\in E, v_i\in C_1, v_j\in C_2\}$ та $\mathrm{Dif}(C_1, C_2)=\infty$,, якщо підграфи не пов'язані ребром (рис. 5.2, б).

Мінімальна внутрішня різниця між підграфами C_1 і C_2 визначається за формулою

MInt(C_1, C_2) = min(Int(C_1) + τ(C_1) , Int(C_2) + τ(C_2)), де τ(C)=k/|C| – поріг розбиття на підграфи, Int(C_1), Int(C_2) – внутрішня різниця підграфів C_1 і C_2 відповідно (рис. 5.2, в). Коефіцієнт k у визначенні MInt поділяє значення внутрішньої різниці та мінімальної внутрішньої різниці. Якщо k велике, сегментація включає більші області (рис. 5.2, г), але k явно не встановлює мінімальний розмір внутрішньої різниці підграфу.

У процесі сегментації попарно порівнюються наявні підграфи, наприклад, C_1 і C_2. Коли різниця між двома підграфами Dif(C_1,C_2) більша, ніж мінімальна внутрішня різниця двох підграфів MInt(C_1, C_2), ці два підграфи C_1 і C_2 розглядаються як окремі підграфи. Інакше підграфи слід об'єднати.

Алгоритм сегментації наступний. Дано вхідний граф G=(V, E) з n вершинами та m ребрами. Результат алгоритму є розбиття V на подграфи – сегментація S = (C_1, ..., C_r).

1. Ребра сортуються за вагою у порядку зростання та позначаються як e_1, e_2, …, e_m.

2. Сегментація агломеративна, тому спочатку кожному пікселю відповідає окремий підграф, тоді маємо n підграфів.

3. Повторюємо для k= k=1, …, m.

3.1. Сегментацію на кроці k позначимо як S_k.

3.2. Беремо k-е ребро по порядку, e_k=(v_i , v_j). Якщо v_i , v_j належать одному й тому підграфу, нічого не робимо і, тоді S_k=S_{k-1}. Якщо v_i , v_j належать двом різним підграфам C_i і C_j при сегментації S_{k-1}, об'єднуємо їх в один підграф, якщо $w(v_i,v_j) \le$ MInt(C_i, C_j); інакше нічого не робимо.

4. Повертається результат сегментації на кроці m – це S_m.

Приклад 5.2. Сегментація зображення методом Фельценшвальба та Хаттенлохера.

```
from skimage.data import camera
from skimage.util import img_as_float
#import function for marking boundaries
from skimage.segmentation import mark_boundaries
from skimage.segmentation import felzenszwalb
import numpy as np
import matplotlib.pyplot as plt # to plot any graph
img = img_as_float(camera()[::2, ::2])
res3 = felzenszwalb(img, scale=100)
res4 = felzenszwalb(img, scale=500)
fig = plt.figure(figsize=(12, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.imshow(mark_boundaries(img, res3)); ax1.set_xlabel("With k=100")
ax2.imshow(mark_boundaries(img, res4)); ax2.set_xlabel("With k=500")
fig.suptitle("Plotting the boundaries")
print(f'Felzenszwalb number of segments:', len(np.unique(res3)))
print(f'Felzenszwalb number of segments:', len(np.unique(res4)))
plt.show()
```

Завдання для виконання роботи

1. Отримати оригінальне кольорове зображення – свою фотографію, постаратися забезпечити рівномірне освітлення. Завантажити отримане зображення та ще два інші портретні зображення.

2. Проаналізувати, як впливають на якість сегментації зображень значення параметрів методу `slic()`. Помістити у звіт описи та діапазони зміни параметрів методу `slic()`.

3. Виконати сегментацію трьох завантажених зображень методом SLIC, налаштувавши параметри методу так, щоб виділити область обличчя на портретах (приклад 5.1). Привести початкові зображення та зображення, отримані в результаті сегментації.

4. Проаналізувати, як впливають на якість сегментації зображень значення параметрів методу `felzenszwalb()`. Помістити у звіт описи та діапазони зміни параметрів методу `felzenszwalb()`.

5. Виконати сегментацію трьох завантажених зображень методом Фельценшвальба та Хаттенлохера, настроївши параметри методу так, щоб виділити контур силуету людини на портретах (приклад 5.2). Привести початкові зображення та зображення, отримані в результаті сегментації.

Контрольні питання

1. За яким принципом виконується сегментація методом SLIC?
2. За яким принципом виконується сегментація методом Фельценшвальба та Хаттенлохера?
3. Як при вирішенні практичних завдань налаштувати параметри методів сегментації SLIC і Фельценшвальба та Хаттенлохера?

Лабораторне заняття 6

Опис границь об'єктів на півтонових зображеннях за допомогою сигнатури

Мета заняття: ознайомитися з методами опису границь об'єктів на напівтонових зображеннях за допомогою сигнатури з бібліотеки scikit-image.

Зміст заняття: вивчення функцій бібліотеки scikit-image опису границь об'єктів на напівтонових зображеннях.

Теоретичні основи

Для виділення ознак контурів зображень використовуються сигнатури контурів плоских фігур. Сигнатура контуру фігури є одновимірною функцією – це залежність відстані від центру ваги фігури до контуру від кута у радіанах.

Щоб уникнути залежності сигнатури від лінійного зміщення, початковою точкою вибирається центр ваги, так як при зміщенні об'єкта він буде зміщуватися. Для забезпечення незалежності сигнатури від масштабу здійснюється нормування на довжину периметра контуру, оскільки периметр лінійно залежить від розміру вихідного зображення. Інваріантність до повороту об'єкта забезпечується шляхом циклічного усунення значень сигнатури так, щоб вони починалися з мінімального чи максимального значення. Для забезпечення стійкості методу побудови сигнатури до розривів контуру та множинних границь проводиться апроксимація тих значень сигнатури, які потрапляють на розрив контуру та не були визначені, а також вибираються середні значення множинних границь.

Таким чином, побудова сигнатури плоскої фігури включає:

– визначення центру ваги фігури;

– визначення контуру фігури;

– побудова прямих ліній з центру ваги до границі об'єкта з фіксованим кроком по куту;

– апроксимацію невизначених значень;

– нормування отриманих значень поділом їх на довжину периметра;

– циклічне зрушення значень так, щоб вони починалися з мінімального (максимального) значення.

Для побудови сигнатур, що враховують особливості об'єктів зображень, що аналізуються, в бібліотеці scikit-image реалізовані функції для визначення центру ваги фігури та визначення контуру фігури.

Приклад 6.1. Вхідне зображення читається із зовнішнього файлу і перетворюється на

напівтонове. Визначення центру ваги фігури виконується функцією center_of_mass().

```
import numpy as np # numeric python lib
import matplotlib.image as mpimg # reading images to numpy arrays
import matplotlib.pyplot as plt # to plot any graph
import matplotlib.patches as mpatches # draw a circle at the mean
#contour
from skimage import measure # to find shape contour
import scipy.ndimage as ndi # to determine shape centrality

# matplotlib setup
from pylab import rcParams
rcParams['figure.figsize'] = (6, 6) # setting default size of plots

# reading an image file using matplotlib в numpy array
image = mpimg.imread('C:/Users/Desktop/1.bmp')
from skimage.color import rgb2gray
img = rgb2gray(image)

# find the center of the leaf
cy, cx = ndi.center_of_mass(img)

plt.imshow(img, cmap='Set1') # show the leaf
plt.scatter(cx, cy) # show its center
plt.show()
```

Приклад 6.2. Визначення контуру фігури виконується функцією find_contours(). Параметри цієї функції – вхідне напівтонове зображення та граничне значення. Для виділення контуру об'єкта знаходиться найдовший контур, передбачається, що помилкові контури, що не належать до об'єкта, мають невелику довжину. Використовується функція max(), параметри якої – множина контурів та вказівка на відбір контуру за довжиною.

```
# scikit-learn imaging contour finding, returns a list of found edges
contours = measure.find_contours(img, .4)

# from which we choose the longest one
contour = max (contours, key = len)

# let us see the contour that we hopefully found
plt.plot(contour[:,1], contour[:,0], linewidth=1)
plt.show()
```

Приклад 6.3. Визначення точок контуру фігури у полярних координатах. Побудова сигнатури фігури за допомогою прямих ліній із центру ваги до границі об'єкта з фіксованим кроком по кутку. Функція cart2pol() отримує на вхід декартові координати (x, y) точки та переводить їх у полярні координати за формулами:

полярний радіус: $\rho = \sqrt{x^2 + y^2}$;
полярний кут: .$\varphi = arctg(y/x)$.

```
# numpy is smart and assumes the same about us
# if we substract a number from array of numbers,
# it assumes that we wanted to substract from all members
contour[:,1] -= cx # demean X
contour[:,0] -= cy # demean Y
```

```
# checking if we succeeded to move the center to (0,0)
plt.plot(-contour[::,1], -contour[::,0], linewidth=0.5)
plt.grid()
plt.scatter(0, 0)
plt.show()

# just calling the transformation on all pairs in the set
polar_contour = np.array([cart2pol(x, y) for x, y in contour])

# and plotting the result
rcParams['figure.figsize'] = (12, 6)
plt.subplot(121)
plt.scatter(polar_contour[::,1], polar_contour[::,0], linewidth=1, s=.5,
c=polar_contour[::,1])
plt.title('in Polar Coordinates')
plt.grid()
plt.subplot(122)
plt.scatter(contour[::,1], # x axis is radians
contour[::,0], # y axis is distance from center
linewidth=1, s=2, # small points, w/o borders
c=range(len(contour))) # continuous coloring (so that plots match)
plt.scatter(0, 0)
plt.title('in Cartesian Coordinates')
plt.grid()
plt.show()
```

Завдання для виконання роботи

1. Створити файл із зображенням листа з дерева згідно з варіантом (табл. 6.1 та рис. 6.1). Прочитати вхідне зображення з файлу та перевести його у напівтонове зображення. Привести вхідне зображення листа та напівтонове.
2. Визначити центр ваги листа функцією center_of_mass(). Відобразити знайдену точку на напівтоновому зображенні листа.
3. Виділити контур листа функцією find_contours(), підібравши граничне значення. Позбавитися хибних контурів функцією max(). Вивести зображення отриманого контуру листа.
4. Створити функцію cart2pol, вхідні аргументи якої – декартові координати точки. Ця функція має повертати полярні координати точки.
5. Перетворити декартові координати точок контуру листа на полярні координати. Вивести графік сигнатури листа залежно від кута повороту відрізка із центру ваги листа до точки контуру в радіанах.

Контрольні питання

1. Як створити множину зображень рукописних цифр для класифікації?
2. За яким принципом виконується класифікація рукописних цифр за допомогою згорткової нейронної мережі?
3. Як розбити вибірку на навчальну та тестову?
4. Як під час вирішення практичних завдань оцінити якість класифікації?

Лабораторне заняття 7

Розпізнавання цифр методом дискримінантного аналізу

Мета заняття: вивчення можливостей розпізнавання цифр методом дискримінантного аналізу.

Зміст заняття: Побудова програми, яка моделює розпізнавання цифр методом дискримінантного аналізу.

Теоретичні основи

Робота включає три етапи:

1. Підготовка еталонних (навчальних) образів друкованих цифр як масивів даних.
2. Формування множини невідомих образів друкованих цифр.
3. Класифікація невідомого способу методом дискримінантного аналізу.

Для виконання цієї роботи на Python необхідно підключити бібліотеку scikit-image. Для цього в командному рядку Windows набрати:

```
python -m pip install -U scikit-image
```

Документацію цієї бібліотеки можна переглянути за посиланням:
API Reference for skimage 0.18.0 - skimage v0.18.0 docs (scikit-image.org)

Підготовка еталонних образів

Набіром еталонних образів є послідовність із десяти цифр від 0 до 9. У цьому прикладі число образів M=10. У разі, коли кожен клас образів характеризується лише своїм зразком, маємо число класів, також рівне M. Ширина зразка N_1 пікселів і висота зразка N_2 пікселів; N_1, N_2 = 8 ... 20 задаються варіантом завдання. На рис. 7.1 наведено приклади графічних символів цифр, якщо N_1=10, N_2=12 пікселів.

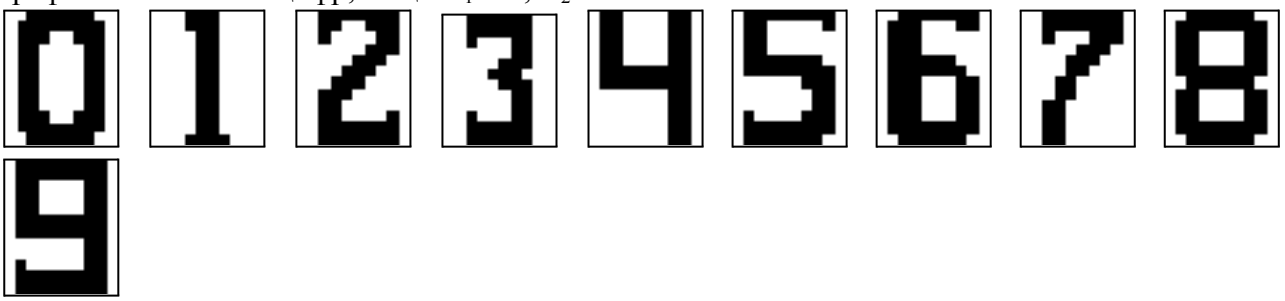


Рисунок 7.1 - Приклади графічних символів цифр.

Наприклад, програму можна розпочати так:

```
import numpy as np
import skimage.io as io
x=np.zeros((5,3))
x[0,1]=1
x[1,0:2]=1
x[2,1]=1
x[3,1]=1
x[4,:]=1
```

```
print('Initial data: ', x)

# відображення цифри на екран

io.imshow(x)

io.show()
```

Формування множини невідомих образів друкованих цифр

Формування невідомих образів проводиться шляхом незначного спотворення еталонних образів шляхом додавання до нього рівномірного (за площею зображення) шуму типу «сіль і перець», що є випадковим спотворенням окремих пікселів зображення:

```
from skimage.util import random_noise

x_noise=random_noise(x,mode='s&p', amount=0.5)
```

Ступінь спотворення характеризується числом $p \in [0;1]$, що визначає частку спотворених пікселів. Цьому числу відповідає параметр amount.

Такий підхід при формуванні образів дозволяє регулювати шляхом зміни значення p ступінь розкиду множини образів у межах одного класу.

Класифікація зображень цифр

При порівнянні бінарних зображень цифр насамперед виконується перетворення матриці еталонного образу у вектор ознак. Еталонний образ кожного символу представляється як вектор-стовпець $[N,1]$, число елементів N якого дорівнює числу ознак (іншими словами, N – розмірність простору ознак). Такий вектор-стовпець формується із двовимірного масиву-зображення A розміру $[N_1, N_2]$ за допомогою команди:

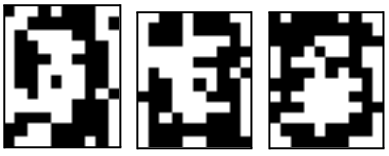
```
x = A.reshape (1, n1 * n2)
```

Ця команда перетворює двовимірний масив $A[N_1, N_2]$ у одновимірний вектор-стовпець $x[N,1]$, де $N=N_1*N_2$.

Далі для порівняння бінарних зображень цифр використовуємо міру близькості – коефіцієнт Хеммінгу $\mu_{ij}^{H} = \rho_{ij} / N$, де ρ_{ij} – загальна кількість значень ознак, що збігаються (нульових і одиничних), N – число ознак об'єкта. Приналежність невідомого об'єкта конкретному класу визначається за максимальним значенням коефіцієнта Хеммінга.

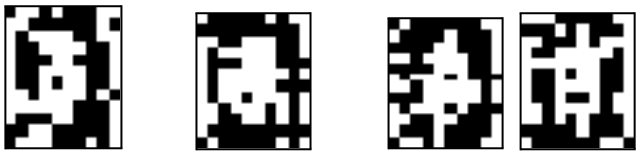
```
від scipy.spatial import distance

print(1-distance.hamming(x.reshape(n1*n2,1), x_noise.reshape(n1*n2,1)))
```

На рис. 7.2 – 7.5 наведено деякі приклади розпізнавання символів, зображених на рис. 7.1 методом дискримінантного аналізу. Навчання проводилося за числі навчальних образів M=10 кожного виду символу і параметрі спотворення символів p=0,1...0,2.



Результат розпізнавання: «2»**, «3», «5»**

Рисунок 7.2 - Невірні розпізнавання символу «0», спотвореного шумом "сіль і перець", p =0,2.



Результати розпізнавання: «**0**», «**0», «0», «0»**

Рисунок 7.3 - Правильні розпізнавання символу «0», спотвореного шумом "сіль і перець", p = 0,2.

Результат розпізнавання: «3», «5», «6»

Рисунок 7.4 - Неправильні розпізнавання символу «4», спотвореного шумом "сіль і перець", p = 0,2.

Результати розпізнавання:«4», **«4», «4», «4», «4»**

Рисунок 7.5 - Правильні розпізнавання символу «4», спотвореного шумом "сіль і перець", $p = 0{,}2$.

Якість класифікації характеризується ймовірностями правильної класифікації $P(i)$ образу i-го класу, i=1, …, M , які оцінюються за формулою: $P(i)= N_i/ N$, де N_i – число правильних розпізнавань образу i-го класу; N – загальна кількість образів i-го класу. Число N_i визначається експериментально при значеннях N_0 =10…100.

Завдання для виконання роботи

1. Створити еталонні зображення цифр розміру $N_1 \times N_2$, де N_1, N_2 задаються варіантом завдання.
2. На зображення цифр, заданих варіантом у 4-му стовпці табл. 7.1, додати заваду типу «сіль і перець», параметр завади p задається варіантом (два значення в табл. 7.1, в 5-му стовпці). Для кожного значення параметра завади p отримати 10 зашумлених зображень кожної цифри.
3. Для кожного значення параметра завади виконати класифікацію 10 зашумлених зображень кожної цифри, отриманих на етапі 2. Використати коефіцієнт Хеммінга для порівняння бінарних зображень.
4. Для кожного значення параметра завади p обчислити імовірність правильної класифікації кожної цифри.

Таблиця 7.1– Варіанти завдань

Номер варіанту	N_1	N_2	Цифри для класифікації	Параметр шуму
1	8	13	1, 2	0,1; 0,3
2	9	14	2, 3	0,25; 0,45
3	10	15	3, 4	0,5; 0,7
4	11	16	4, 5	0,4; 0,6
5	12	17	5, 6	0,3; 0,5

6	13	18	6, 7	0,35; 0,55
7	14	19	7, 8	0,55; 0,75
8	15	20	8, 9	0,2; 0,35
9	8	14	9, 0	0,2; 0,65
10	9	15	0, 1	0,3; 0,4
11	10	16	1, 3	0,15; 0,55
12	11	17	2, 4	0,25; 0,6
13	12	18	5, 7	0,1; 0,5
14	13	19	6, 8	0,4; 0,55
15	14	20	7, 9	0,15; 0,3
16	12	15	1, 4	0,1; 0,4
17	11	18	2, 5	0,2; 0,3

Контрольні питання

1. Як сформувати вектори ознак із матриць зображень?
2. Яка міра близькості застосовується для порівняння бінарних зображень?
3. Яка відстань застосовується для порівняння напівтонових зображень?
4. Як оцінити якість класифікації?

Рекомендована література

1. Кутковецький В. Я. Розпізнавання образів : навчальний посібник / В. Я. Кутковецький. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2017. – 420 с.
2. Машинне навчання та обробка сигналів в біомедичних електронних системах: навч. / КПІ ім. Ігоря Сікорського; уклад.: К.О. Іванько, А.О. Попов, Н.Г. Іванушкіна. – Київ: КПІ ім. Ігоря Сікорського, 2020. – 97 с.
3. Кононова К. Ю. Машинне навчання: методи та моделі: підручник / К. Ю. Кононова. – Харків: ХНУ імені В. Н. Каразіна, 2020. – 301 с.
4. Т. М. Басюк, В. В. Литвин, Л. М. Захарія, Н. Е. Кунанець. Машинне навчання: Навчальний посібник. - Львів: Видавництво «Новий Світ - 2000», 2021. - 315 с.
5. Мосіюк О. О. Штучний інтелект: вступ до машинного навчання: навчально-методичний посібник. – Житомир: Вид-во ЖДУ ім. Івана Франка, 2019. – 76 с.

Навчальне видання

Методичні рекомендації до лабораторних занять з дисципліни «Теорія розпізнавання образів» для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 124 – Системний аналіз та за спеціальністю 113 – Прикладна математика.

Укладач: Марина Вячеславівна Полякова