МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи
«Створення системи розпізнавання зображень на основі регресійного класифікатору»

з курсу «Розпізнавання образів»

для студентів спеціальностей 122 «Комп’ютерні науки»

Затверджено
редакційно-видавничою
радою університету,
протокол № 2 від 27.06.2024 р.

Харків
НТУ «ХПІ»
2024

Методичні вказівки до лабораторної роботи «Створення системи розпізнавання зображень на основі регресійного класифікатору» з курсу «Розпізнавання образів» для студентів спеціальностей 122 «Комп’ютерні науки» / уклад. В.О. Колбасін. – Харків: НТУ «ХПІ», 2024. – 24 с.

Укладач В. О. Колбасін

Рецензент О. В. Костюк

Кафедра системного аналізу та інформаційно-аналітичних технологій

ВСТУП

Сучасні технології дають можливість відчутно автоматизувати сфери діяльності, які раніше були притаманні виключно людині. Однією з таких сфер є розпізнавання образів, зокрема зображень. Технології розпізнавання знаходять своє місце як для вирішення промислових задач у випадку систем технічного комп'ютерного зору, так і в побутових задачах, таких як оптичне розпізнавання тексту та розпізнавання природньої мови.

Більшість новітніх технологій розпізнавання використовують штучні нейронні мережі, за рахунок чого отримують реальний прорив в якості розпізнавання. Але вони є складними та достатньо вимогливими до ресурсів, що відбивається на економічній ефективності систем. Тому класичні алгоритми класифікації і досі знаходять своє використання в багатьох практичних застосунках.

Також класичні технології розпізнавання є дуже зручними для того, щоб приділити увагу не стільки самому алгоритму розпізнавання, скільки тому, як система розпізнавання будується, яким обов'язковим елементам системи та етапам її побудови має бути приділена увага.

Лабораторна робота «Створення системи розпізнавання зображень на основі регресійного класифікатора» присвячена опануванню стандартних етапів побудови системи розпізнавання двох класів зображень на основі класичного алгоритму класифікації – логістичної регресії. В процесі виконання лабораторної роботи пропонується створити процедуру виділення вектору ознак, виконати навчання моделі та оцінити якість створеної системи розпізнавання. Практичні навички та наробки, отримані в ході виконання лабораторної роботи будуть корисними у подальшому вивченні дисципліни «Розпізнавання образів» та можуть стати при нагоді у подальшій практичній діяльності інженера в ролі інженера машинного навчання або інженера роботи з даними.

Мета: отримати практичні навички з побудови та оцінки якості систем розпізнавання образів на прикладі використання методу логістичної регресії для вирішення задачі класифікації у середовищі Jupyter Notebook.

1. ВСТАНОВЛЕННЯ ТА РОБОТА З JUPYTER NOTEBOOK

Для виконання лабораторної роботи пропонується використати програмне забезпечення Jupyter Notebook [1], яке є одним з найбільш популярних середовищ розробки застосунків машинного навчання. Встановити його можна як з офіційного сайту, так і використати архів зі встановленим ПЗ з OneDrive ресурсу кафедри. Також за бажанням можна

використати аналогічні продукти, що пропонуються основними хмарними платформами – AWS, Google Cloud, Azure.

В даних методичних вказівках виконання роботи розглядається з використанням веб-версії Jupyter Notebook, але так само її можна виконувати у середовищі розробки PyCharm Professional.

Але оскільки виконання лабораторної роботи з використанням хмарних продуктів та середовища розробки PyCharm Professional суттєво не відрізняється від використання локально встановленої версії, то в цих методичних вказівках воно не розглядається.

1.1. Встановлення та робота з Jupyter Notebook

1.1.1. Встановлення з архіву

Для початку роботи треба завантажити з OneDrive ресурсу кафедри архів *jupyter-image-rec.zip* та розгорнути його в каталозі **c:\xampp**. Важливо розгортати саме в цьому каталозі, бо налаштування шляхів всередині віртуального оточення інтерпретатору Python було виконане саме для нього.

Далі треба запустити застосунок за шляхом:

```
C:\xampp\Python3.13\image_rec\Scripts\jupyter-lab.exe
```

Після його запуску в браузері за замовчанням буде відкрито сторінку наступного виду:

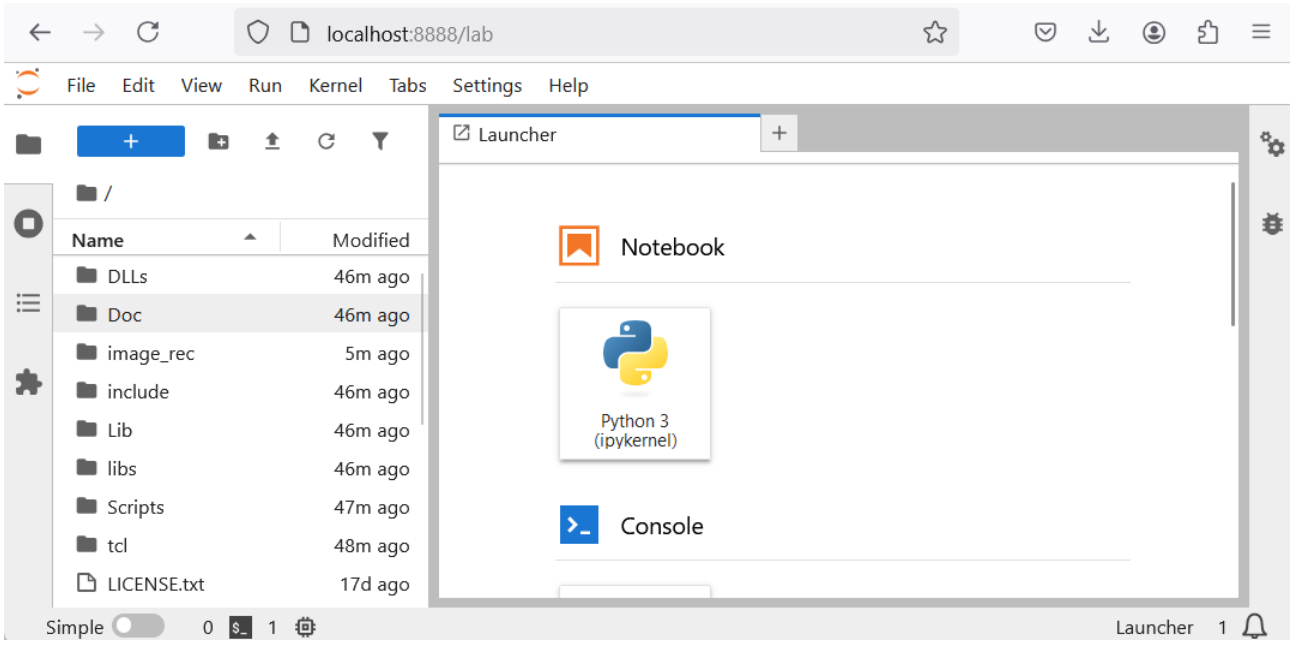

Рисунок 1.1 – Головна сторінка Jupyter Notebook

1.1.2. Самостійне встановлення

Спочатку треба встановити інтерпретатор мови програмування Python версії не нижче 3.7. Зробити це можна відповідно до посилань та інструкцій на сайті: https://www.python.org/downloads/.

Для того, щоб запобігти змішуванню бібліотек та утиліт, які потрібні для вирішення різних задач, в Python передбачено використання віртуальних оточень (virtual environments, venv`s). Рекомендовано використовувати його і при виконанні лабораторних робіт даного курсу.

Щоб використати віртуальні оточення, спочатку треба встановити бібліотеку їх підтримки за допомогою наступної команди:

```
pip install virtualenv
```

Після цього треба створити віртуальне оточення командою:

```
virtualenv image_rec
```

яка створить в поточному каталозі підкаталог ***image_rec***, що буде містити файли віртуального оточення.

Далі потрібно активувати віртуальне оточення командою:

```
image_rec\Scripts\activate
```

Та встановити необхідні бібліотеки за допомогою команди:

```
pip install numpy scikit-learn scikit-image pandas opencv-python
jupyterlab matplotlib
```

Після успішного встановлення всіх бібліотек запустити Jupyter Notebook можна командою:

```
jupyter-lab
```

яка запустить застосунок та відкриє у браузері головну сторінку застосунку, яка наведена на рис. 1.1.

1.1.3. Робота з Jupyter Notebook

Jupyter Notebook має модульну архітектуру, що складається з серверної частини, ядра (kernel) та інтерфейсу користувача. Сервер забезпечує виконання ядер та взаємодію з інтерфейсом користувача. Ядро відповідає за виконання коду. Воно може підтримувати різні мови програмування (Python, R) та фактично є віддаленим обчислювальним рушієм, що зберігає бібліотеки, функції та змінні. Інтерфейс користувача приймає команди, передає їх для виконання ядру та відображає результати користувачеві. Він підтримує декілька варіантів взаємодії з користувачем,

основним з яких є блокнот – документ, що складаються з комірок, які можуть містити код або текст, причому результат виконання коду відображається одразу під відповідною коміркою.

Для створення нового блокноту треба на головні сторінці застосунку обрати пункт **Notebook -> Python 3 (ipykernel)**. Після цього відкриється вікно з документом, як це показано на рис. 1.2.

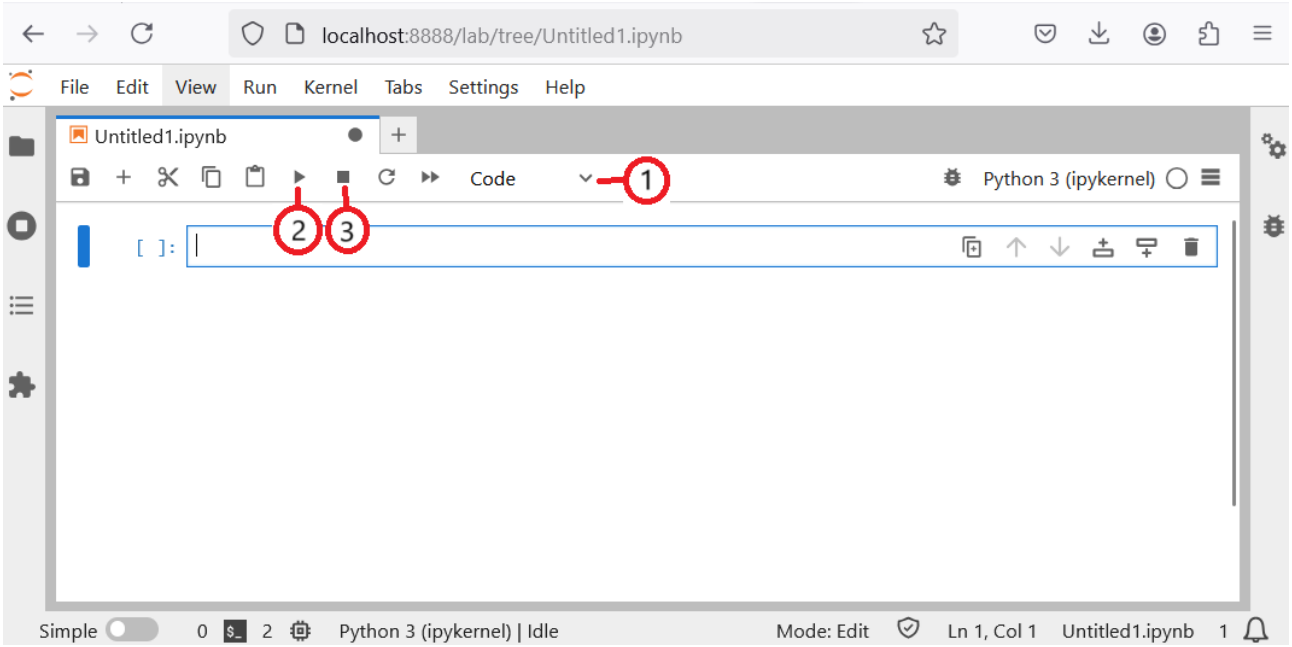

Рисунок 1.2 – Новий блокнот Jupyter Notebook

Блокнот складається з окремих комірок (cell) різних типів, в яких можна розмістити код (Code) або допоміжний текст (Markdown). Тип комірки можна змінити в списку з позначкою 1.

Текстові комірки мають містити текст, оформлений згідно з правилами мови розмітки документів Markdown [2]. При їх виконанні блокнот відображає гіпертекстові дані замість коду комірки. Повторне подвійне натискання на комірку призводить до переходу в режим редагування.

Комірки з кодом виконуються ядром і мають містити одну чи декілька цілісних синтаксичних конструкцій. Неприпустимо починати функцію або цикл в одній комірці та закінчувати в іншій.

Результати виконання комірок з кодом відображаються одразу під коміркою та містять текст, який було виведено в стандартний потік виводу, і зображення, якщо їх було згенеровано кодом.

Для виконання змісту комірки можна використати піктограму запуску (позначка 2) або комбінацію клавіш **Shift-Enter**. Якщо виконання комірки займає занадто багато часу – його можна зупинити, натиснувши на піктограму зупинки (позначка 3) або двічі натиснувши на клавішу «**i**».

Ліворуч від комірки з кодом в прямокутних дужках відображається стан виконання цієї комірки. Якщо в дужках немає нічого – комірка ще не відправлялася на виконання. Число в дужках означає, що комірка вже була виконана, та є порядковим номером її виконання. Зірочка ж означає, що комірка зараз виконується.

Виконання простого блокноту представлене на рис. 1.3.

Рисунок 1.3 – Виконання простого блокноту

Як можна бачити на наведеному прикладі, виконувати комірки можливо у довільному порядку. Також слід зазначити що за потреби одну комірку можна виконати декілька разів поспіль.

Блокнот зберігається на стороні сервера у вигляді файлу з розширенням **.ipynb.** Файли блокнотів за замовчанням зберігаються в каталозі, де розташовано Jupyter Notebook. Зручним підходом до впорядкування роботи з блокнотами є створення відповідного підкаталогу для зберігання виключно блокнотів.

У випадку локально запущеного застосунку Jupyter Notebook файл блокноту можна скопіювати безпосередньо з каталогу віртуального оточення. Але при використанні віддаленої інсталяції це неможливо і тоді зберегти файл на локальному комп'ютері можна за допомогою пункту меню **File -> Download**.

Для завантаження файлу блокноту в застосунок, його можна просто перетягнути до лівої панелі зі списком файлів. Також є можливість завантажити файл за його URL адресою.

Змінити назву блокноту можливо за допомогою пункту меню **File -> Save Notebook as…**, або натиснувши праву кнопку миші на назві вкладки та обравши пункт контекстного меню **Rename Notebook…**. Система запропонує задати нову назву блокнота, після чого виконає його збереження або перейменування.

2. ПОБУДОВА СИСТЕМИ РОЗПІЗНАВАННЯ ОБРАЗІВ

2.1. Загальна схема системи розпізнавання

Системи розпізнавання образів є типовими зразками систем машинного навчання і робота з ними відповідає життєвому циклу, зображеному на рис. 2.1.

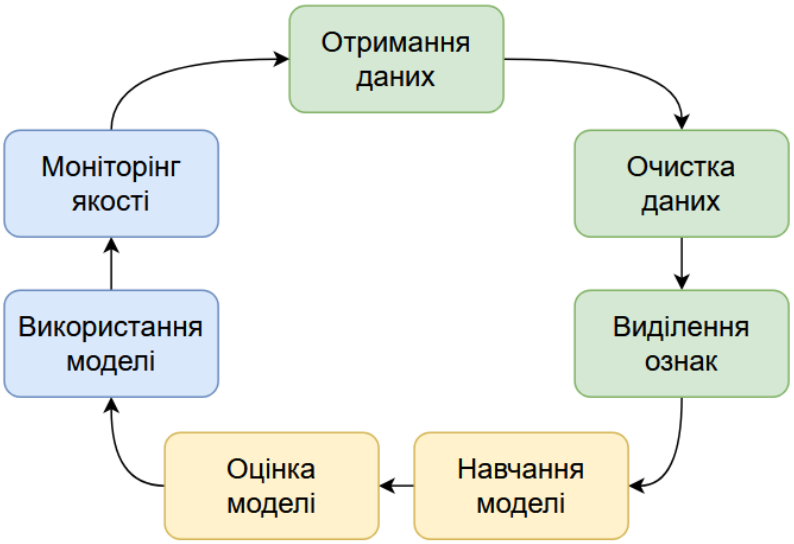

Рисунок 2.1 – Життєвий цикл системи машинного навчання

На етапі **отримання даних** необхідно отримати набір даних (датасет) для подальшої роботи. Набір має бути достатньо великим, щоб на його основі можна було навчити модель. Також при використанні керованого навчання (навчання з вчителем) для кожного елементу датасету має бути задана мітка, що позначатиме клас образу.

Етап **очистки даних** передбачає видалення некоректних елементів із датасету та його балансування. Балансування потрібно для того, щоб під час навчання зразки всіх класів впливали на побудову моделі однаково.

Метою етапу **виділення ознак** є зменшення розмірності даних, що будуть використовуватися при навчанні моделі. Виділення ознак є одним з найважливіших етапів розробки системи, бо від нього напряму залежить її якість. Виділення ознак можливо як на основі наявних знань про образ, так і з використанням математичного апарату зниження розмірності.

Навчання моделі передбачає вибір алгоритму машинного навчання та власне навчання моделі на його основі.

Етап **оцінки моделі** виконується, щоб зрозуміти, наскільки створена модель відповідає поставленим до неї вимогам. Обчислення оцінки якості виконується на датасеті, що не використовувався для навчання моделі.

Використання моделі передбачає застосування моделі для отримання практичного результату.

І **моніторінг моделі** потрібен для того, щоб вчасно помітити невідповідність навченої моделі даним реального світу.

Поставлена в даній лабораторній роботі задача буде відображатися на цю модель життєвого циклу наступним чином.

1. Отримання даних – підготувати по 25-40 характерних зображень двох різних класів для формування датасету.
2. Очистка даних – видалити зображення, які не підходять за якістю, або є повторами одного й того ж самого зображення.
3. Виділення ознак – розробити просте правило на основі загального розуміння відмінності між класами. В якості ознак може бути колір та яскравість частин зображеня, частота змін кольору та яскравості тощо.
4. Оцінка моделі – обчислити матрицю невідповідностей, повноту та влучність моделі.
5. Практичне використання – показати результати використання моделі на декількох прикладах різних класів.
6. Моніторінг моделі – не передбачається.

В подальших розділах буде розглянуто, як можна реалізувати ці етапи в середовищі Jupyter Notebook.

2.2. Зчитування та візуальний аналіз даних

2.2.1. Зчитування файлів зображень

Для роботи з зображеннями використовується бібліотека ***scikit-image***[3]. Вона містить засоби для зчитування та запису файлів зображень, перетворення кольорових просторів та інших операцій з зображеннями.

Бібліотека підключається до блокнота за допомогою наступних команд (псевдонім **ski** є рекомендованим розробниками бібліотеки):

```
import skimage as ski
from skimage import io
```

Для зчитування файлів використовується функція imread, яка має наступний формат виклику:

```
io.imread(fname, as_gray=False, plugin=None, **plugin_args)
```

де **fname** – шлях до файлу, це може бути відносний або абсолютний шлях, також функція підтримує завантаження файлів з мережі Інтернет за URL адресою;
as_gray – чи треба перетворити зображення у формат градацій сірого;
plugin та **plugin_args** – дозволяють вказати назву плагіну, що має бути використаний для завантаження зображення, та його параметри.

Функція повертає масив типу **ndarray**, оголошений в бібліотеці ***NumPy*** [4]. Розмірність масиву для чорно-білих зображень буде дорівнювати MxN, для кольорових у форматі RGB – MxNx3, та для кольорових у форматі RGBA з підтримкою прозорості – MxNx4.

Цю особливість треба враховувати при поєднання зображень різного формату в один датасет, бо однією з частих проблем є помилка обробки зображення через наявність зайвого кольорового каналу – функція обробки очікує отримати три кольорові канали, але отримує чотири. В цьому випадку треба перевірити скільки елементів має масив за останнім третім виміром, та якщо їх більше 3 – видалити зайвий кольоровий канал.

Кількість елементів третього виміру масиву можна отримати через властивість масиву **shape**. А для видалення зайвої площини масиву можна використати функцію **delete** бібліотеки ***NumPy***, вказавши індекс відповідної кольорової площини (3) у третьому вимірі масиву (індекс 2).

Загалом функція зчитування зображення з видаленням каналу прозорості буде виглядати наступним чином:

```
def read_image(file_path):
    pic = io.imread(file_path)
    if pic.shape[2] > 3:
        pic = np.delete(pic, 3, 2)
    return pic
```

Значення елементів масиву будуть відповідати інтенсивності відповідного кольорового каналу в діапазоні 0..255. За необхідністю, значення можна перевести в діапазон 0..1 поділивши масив на 256.

2.2.2. Відображення зображень у блокноті

Для відображення зображень у блокноті використовується бібліотека ***matplotlib***. Детально ознайомитися з її можливостями можна в методичних вказівках [5] або за документацією [6]. В цьому розділі будуть розглянуті тільки засоби, необхідні для виконання лабораторної роботи.

Бібліотека підключається наступним чином:

```
import matplotlib.pyplot as plt
```

Для виведення зображення використовується функція **imshow**, яка у самому простому варіанті використання приймає єдиний параметр – масив точок зображення. На рис. 2.2 наведено фрагмент коду, який друкує розмірність зчитаного зображення та відображає його в блокноті.

```
[4]: pic = read_image("jupyter.png")
     print(pic.shape)
     plt.imshow(pic)

     (57, 207, 3)

[4]: <matplotlib.image.AxesImage at 0x20fbe088830>
```

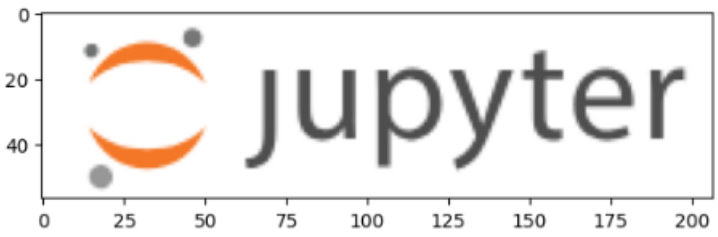

Рисунок 2.2 – Вивід зображення в блокноті

Як можна побачити з рисунку, зображення масштабується при виводі і до нього додаються горизонтальні та вертикальні осі з мітками.

Для виконання візуального аналізу датасету зручно виводити зображення групами у зменшеному вигляді. Для цього можна скористатися функцією **subplot**, яка створює область рисування та розміщує її у комірці таблиці. Функція має такий вигляд:

```
plt.subplot(nrows, ncols, index, **kwargs)
```

де **nrows, ncols** – кількість рядків та колонок областей рисування;
index – індекс комірки, в якій буде розташовано область рисування;
kwargs – іменовані параметри, серед котрих можна виділити **title**, який дозволяє задати назву області рисування.

Детальний опис інших параметрів даної функції можна знайти у документації бібліотеки [6].

Після створення області рисування, вона стає активною та наступні операції відображення будуть відбуватися відносно неї. Тобто щоб вивести зображення в цій області достатньо буде виконати **imshow**.

Код для зчитування зображень різних класів та виведення двох рядків по 5 зображень в кожному буде мати наступний вигляд:

```
[5]: indices = [1, 2, 4, 5, 7, 31, 32, 33, 34, 35]
images = [read_image(f"samples/{file_idx}.jpg") for file_idx in indices]
for i in range(10):
    plt.subplot(2,5, i+1, title=f'{indices[i]}.jpg')
    plt.imshow(images[i])
```

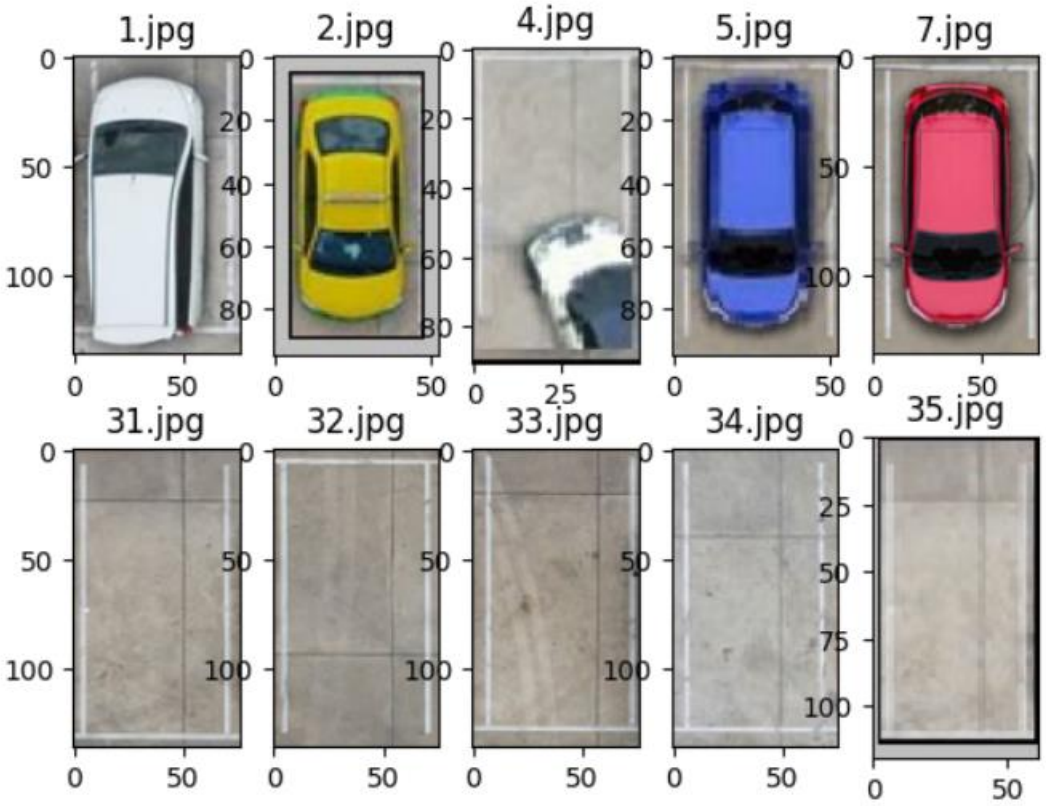

Рисунок 2.3 – Вивід прикладів зображень різних класів

В процесі аналізу та очищення датасету буде корисним оцінити розміри зображень для цілей подальшої уніфікації зображень датасету. Це можна зробити шляхом аналізу властивості **shape**, як показано на рис 2.4:

```
[6]: x_min = min(image.shape[1] for image in images)
x_max = max(image.shape[1] for image in images)
y_min = min(image.shape[0] for image in images)
y_max = max(image.shape[0] for image in images)
print(f"Width: minimum = {x_min}, maximum = {x_max}")
print(f"Height: minimum = {y_min}, maximum = {y_max}")
```

```
Width: minimum = 48, maximum = 76
Height: minimum = 91, maximum = 136
```

Рисунок 2.4 – Знаходження мінімальних та максимальних ширини та висоти зображення

Для зміни розміру зображення використовується функція **resize** бібліотеки ***OpenCV*** [7]. Підключення бібліотеки виконується командою:

```
import cv2
```

А сама функція має наступний формат виклику:

```
cv2.resize(image, new_dimension, interpolation)
```

де **image** – вихідне зображення;
new_dimension – новий розмір зображення;
interpolations – не обов'язковий параметр, алгоритм інтерполяції.

Залежно від алгоритму виділення ознак та розпізнавання приведення зображень до одного розміру може бути виконане зі збереженнями пропорцій співвідношення сторін, або без нього.

2.3. Виділення ознак

Виділення ознак зображення, значущих для вирішення задачі класифікації є доволі складною та нетривіальною задачею, яка потребує розуміння як предметної області, так і алгоритмів машинного навчання. Це складна задача, яку не завжди вдається вирішити емпіричним шляхом.

В цій роботі пропонується зробити спробу виділити ознаки на основі здорового глузду та людського розуміння відмінностей між зображеннями різних класів. Скоріше за все ця процедура виділення ознак буде працювати не найкращим чином, але для виконання лабораторної роботи цього буде достатньо.

Розглянемо корисні ідеї для побудови процедури виділення ознак.

Якщо треба визначити, чи присутній об'єкт на зображенні, та ми знаємо, що він зазвичай знаходиться в певних місцях, можна розбити зображення на області та порівняти яскравість або колір цих областей з яскравістю та кольором інших областей зображення. У випадку присутності об'єкту різниця має бути більша ніж при його відсутності.

Якщо один з класів має більш складний рельєф зображення, можна оцінити різницю між яскравістю точок зображення, та середньою яскравістю, очікуючи, що для першого класу різниця буде більше ніж для другого.

Можна робити аналогічні оцінки по одній або іншій осі зображення у випадку наявності характерних полос на зображенні.

Також зручним методом зменшення розмірності та нормалізації розміру зображення є використання гістограм зображення. Гістограма дозволяє отримати кількісний розподіл точок на зображенні за їх

яскравістю і це можна використати як показник приналежності зображення тому чи іншому класу.

Розглянемо побудову процедури виділення ознак на прикладі задачі розпізнавання пустих та заповнених місць для паркування. Приклади відповідних зображень можна знайти на рис 2.3.

Для пустого місця паркування можна побачити, що більшу частину зображення займає асфальт. Також на зображенні є лінії розмітки та сліди від шин, але вони займають відносно малу частину зображення. Тобто можна зробити припущення, що у даному випадку більшість зображення буде мати приблизно однакову яскравість та колір.

Більшу частину зайнятого місця для паркування буде займати зображення автомобіля. Скоріше за все воно буде кольоровим, але може бути і сірим. Хоча навіть у цьому випадку вікна автомобіля будуть відрізнятися за кольором та яскравістю від його кузова. Тобто таке зображення буде менш однорідним як за кольором, так і за яскравістю.

Формально відмінність даних класів зображень можна представити як те, що для пустого паркомісця гістограма зображення буде зосереджена в області піка, а для заповненого – розтягнута по краях.

В рамках даного прикладу обмежимося використанням тільки гістограми яскравості і не будемо аналізувати гістограми кольорових каналів. Це можна зробити самостійно, за показаним тут принципом.

Для побудови гістограми яскравості треба перетворити зображення з кольорового представлення на представлення у градаціях сірого. Одним з найкращих варіантів такого перетворення є трансформація зображення до кольорового простору YUV або Lab.

Перетворення зображення у кольорові простори Lab та YUV можна виконати за допомогою функцій **rgb2lab** та **rgb2yuv** модуля **color** бібліотеки ***scikit-image***. Канали для передачі кольору (U, V та a, b) можуть містити негативні значення, що треба враховувати при використання даних цих каналів. Також слід зазначити, що канал L кольорового простору Lab приймає значення від 0 до 100.

Перед тим, як починати створювати функцію виділення ознак, є сенс перевірити зроблене припущення на наявних даних. Для цього можна побудувати гістограми яскравості для датасету з використанням функції **hist** бібліотеки ***Matplotlib***, яка має наступний формат виклику:

```
plt.hist(array, bins, range, edgecolor)
```

де **array** – одновимірний масив, з елементів якого будується гістограма;
bins – кількість стовпчиків гістограми;
range – діапазон значень, по якому будується гістограма;
edgecolor – колір границь стовпчиків гістограми.

Код побудови гістограм для зображень зайнятих та пустих місць паркування та результат його виконання наведено на рис. 2.5:

```
[7]: for i in range(10):
         lab_pic = ski.color.rgb2lab(images[i])
         plt.subplot(2,5, i+1, title=f'{indices[i]}.jpg')
         plt.hist(np.concatenate(lab_pic[:,:,0]), bins=11, range=(0,100), edgecolor='black')
```

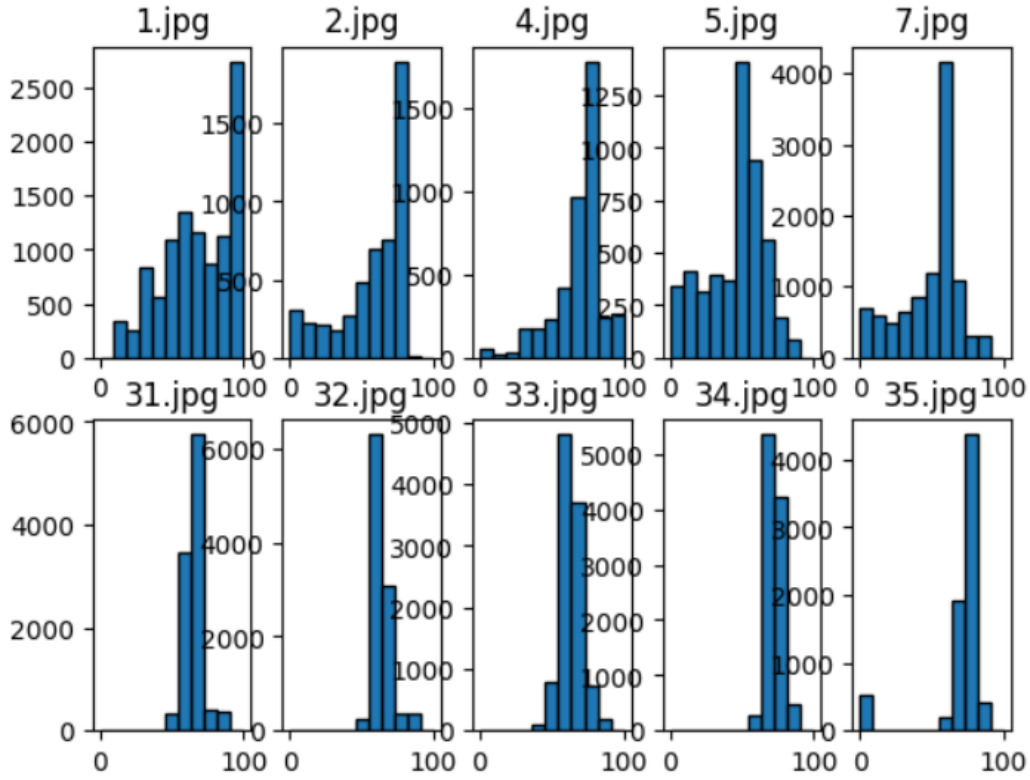

Рисунок 2.5 – Гістограми яскравості для заповнених та пустих паркомісць

Як можна побачити на рисунку, для пустих паркомісць характерне зосередження яскравості точок в двох сусідніх стовпчиках гістограми, а для зайнятих вона розподілена більш рівномірно. Тому в якості ознаки будемо використовувати відношення кількості точок в найбільшому стовпчику гістограми та сусідніх до нього стовпчиках до загальної кількості точок на зображенні (що є сумою всіх стовпчиків гістограми).

Для побудови гістограми будемо використовувати функцію **histogram** бібліотеки ***NumPy***, параметрами якої є одновимірний масив, з елементів якого будується гістограма, та кількість стовпчиків гістограми. Функція повертає двовимірний масив даних: перший рядок містить кількість елементів, які попали у відповідні стовпчики, а другий – частку від загальної кількості елементів. Для наших цілей підходить будь-який рядок, тому візьмемо перший.

Функція виділення ознак буде працювати наступним чином. Спершу вона буде перетворювати зображення у простір Lab і виділяти канал

яскравості. Далі за даними каналу яскравості буде будуватися гістограма та знаходитися сума значень максимального та сусідніх з ним стовпчиків. Результат ділення отриманої суми на загальну суму елементів гістограми і буде значенням ознаки.

Код функції виділення ознак буде мати такий вид:

```
def get_feature(image):
    lab_pic = ski.color.rgb2lab(image)
    L = lab_pic[:,:,0]
    hist = np.histogram(np.concatenate(L), bins=11, range=[0,100])
    raw_hist = hist[0]
    max_idx = np.argmax(raw_hist)
    peak = raw_hist[max_idx]
    peak = peak + (raw_hist[max_idx-1] if max_idx>0 else 0)
    peak = peak + (raw_hist[max_idx+1] if max_idx<10 else 0)
    return peak / np.sum(raw_hist)
```

2.4. Навчання та оцінка якості моделі

2.4.1. Підготовка датасету та навчання моделі

Для навчання та оцінки якості моделі потрібно створити маркований датасет, кожне зображення якого буде мати мітку класу. Зазвичай при вирішенні задачі бінарної класифікації в якості міток використовуються значення 0 та 1.

Оцінка якості моделі має бути побудованою з використанням даних, які не були раніше використані при навчанні моделі. Нехтування цим правилом призводить до перенавчання – ситуації, коли модель демонструє гарні результати на навчальній вибірці, «запам'ятовуючи» її, але при використанні інших даних якість розпізнавання суттєво знижується.

Таким чином створений маркований датасет треба розділити на навчальну та тестову вибірки у випадковому порядку, забезпечивши пропорційне представництво зображень кожного з класів.

Для розбиття датасету на навчальну та тестову вибірки можна використати функцію **train_test_split** бібліотеки ***scikit-learn*** [8], яка має наступний формат виклику:

```
x_train, x_test, y_train, y_test =
        train_test_split(arrays, train_size, test_size, stratify)
```

де **arrays** – датасет у вигляді списку масивів з однаковим числом рядків;
test_size – кількість або частка елементів у тестовій вибірці;
train_size – кількість або частка елементів у навчальній вибірці;
stratify – масив, який містить мітки класів, для пропорційного розподілу елементів по вибіркам.

Параметри **test_size** та **train_size** є необов'язковими, вони можуть містити як ціле число, так і число з плаваючою точкою. У випадку завдання цілого числа параметр буде означати точну кількість елементів у відповідній вибірці. Якщо значення параметра є числом з плаваючою точкою в діапазоні 0..1 – параметр буде означати частку вибірки від загального набору елементів. Якщо значення параметрів не задавати – функція виділить 25% датасету на тестову вибірку а залишок – на навчальну.

Якщо вектори ознак різних класів завантажені у змінні **x_0** та **x_1**, то створити вектор міток та розбити датасет на навчальну та тестову вибірки можна за допомогою наступного коду:

```
x = x_0 + x_1
y = [0 for i in x_0] + [1 for i in x_1]
x_train,x_test,y_train,y_test = train_test_split(x, y, stratify=y)
```

В даному прикладі будемо позначати міткою 1 зайняте, а міткою 0 – вільне паркомісце.

Найбільш простою моделлю для бінарної класифікації є логістична регресія [9]. Реалізація цієї моделі є в бібліотеці ***scikit-learn*** у вигляді класу **LogisticRegression**. Для використання моделі треба створити екземпляр класу задавши відповідні налаштування та навчити модель. Це можна зробити за допомогою наступного коду:

```
x = np.reshape(x_train, (-1,1))
model = LogisticRegression(solver='liblinear').fit(x, y_train)
```

Метод навчання моделі **fit** приймає на вхід двовимірний масив, рядки якого містять вектори ознак, та вектор або список міток. Кількість рядків вектору ознак має співпадати з кількістю міток у списку або векторі.

В даному прикладі ознаки зберігаються в простому одновимірному списку, який треба перетворити у матрицю з N рядків та однієї колонки. Для цього використано функцію **reshape** бібліотеки ***NumPy***, параметр якої вказує, що матриця має містити одну колонку та довільне число рядків (значення -1). У випадку, коли вектор ознак має більше значень, цей етап може не знадобитися.

Після переформатування вектору ознак створюється об'єкт класу **LogisticRegression**, який буде використовувати алгоритм **liblinear**, рекомендований для вирішення задач бінарної класифікації у випадку невеликого датасету. Далі викликом методу **fit** виконується навчання моделі та навчена модель записується у змінну **model**.

Навчену модель можна використати для розпізнавання зображень за вектором ознак використовуючи метод **predict**:

```
y = model.predict(feature_vector)
```

В якості параметра метод приймає двовимірний масив, рядки якого містять вектори ознак, а як результат повертає вектор результатів розпізнавання.

2.4.2. Метрики оцінки якості моделі

Найбільш розповсюдженим підходом до оцінки якості системи класифікації є побудова матриці невідповідностей (confusion matrix), яка представляє собою таблицю спеціального компонування, шо містить кількість вірно та помилково класифікованих об’єктів у відповідності до їх реального та прогнозованого класів.

Колонки матриці невідповідностей представляють собою реальні класи, а рядки – результати прогнозування. Комірками матриці є кількість об’єктів, що відповідають комбінації реального та прогнозованого класу. Наприклад, якщо на вхід системі надати об’єкт з міткою класу 1 та система розпізнає його як екземпляр класу 0, треба буде додати одиницю до комірки в рядку 0 та колонці 1 (містить +1 в дужках), як це показано на рис. 2.6:

		Реальний клас	
		0	**1**
Прогнозований клас	**0**	10	1 *(+1)*
	1	2	5

Рисунок 2.6 – Матриця невідповідностей, загальний вигляд.

У випадку, коли система бінарної класифікації відповідає на питання так чи ні (перевірка на спам, підтвердження або спростування чогось), матрицю невідповідностей стає можливим представити як:

		Реальний клас	
		Позитивні	**Негативні**
Прогнозований клас	**Позитивні**	Істинно позитивні True Positive (TP)	Хибно позитивні False Positive (FP)
	Негативні	Хибно негативні False Negative (FN)	Істинно негативні True Negative (TN)

Рисунок 2.7 – Матриця невідповідностей для випадку підтвердження або спростування.

Матриця дає можливість наявно побачити, які невідповідності між класами допускає система розпізнавання. Також на її основі можна налаштувати систему в залежності від того, які помилки розпізнавання є допустимими при вирішенні практичної задачі. Наприклад, в системі розпізнавання спаму збільшення частки хибно негативних результатів відносно частки хибно позитивних є прийнятним, бо користувач просто проігнорує небажаний електронний лист. А от для системи детектування шахрайства такий варіант є неприйнятним, бо може призвести до втрати грошей. Збільшення ж частки хибно позитивних результатів навпаки буде допустимим, бо це збереже гроші клієнта, хоча і за рахунок певних незручностей.

Також на основі матриці невідповідностей обчислюється цілий ряд показників якості систем машинного навчання, такий як влучність, повнота, точність тощо.

Точність розпізнавання характеризує здатність системи робити точну класифікацію та обчислюється як відношення вірно розпізнаних до всіх поданих на вхід системи образів:

$$accuracy = \frac{TP+TN}{TP+FP+FN+TN}. \quad (2.1)$$

Точність є доволі оманливою характеристикою у випадках незбалансованого датасету, бо якщо тестовий датасет буде містити 90 зайнятих паркомісць і 10 пустих, то якщо алгоритм вірно розпізнає всі зайняті та невірно розпізнає всі 10 пустих місць – його точність буде 0.9 або 90%. Це є відносно високим показником і система може бути прийнята до експлуатації. Але при іншому співвідношенні пустих та заповнених місць значення точності може легко вийти за рамки допустимого – достатньо підрахувати точність при збалансованому розбитті – це буде 0.5 або 50%.

Влучність (precision) визначає частку насправді позитивних об’єктів серед всіх об’єктів, розпізнаних системою як позитивні. Обчислюється як:

$$precision = \frac{TP}{TP+FP}. \quad (2.2)$$

Повнота (recall) визначає частку успішно розпізнаних серед всіх насправді позитивних об’єктів. Обчислюється за формулою:

$$recall = \frac{TP}{TP+FN}. \quad (2.3)$$

Останні дві метрики є сенс розглядати тільки у сукупності, бо по-одинці ними можна доволі легко маніпулювати. Наприклад, якщо система ніколи не повертає негативний результат розпізнавання – вона буде мати повноту 1. Також система, що класифікує більшість позитивних об’єктів як негативні буде мати влучність, близьку до 1.

2.4.3. Обчислення оцінок якості моделі

Щоб побудувати матрицю невідповідностей необхідно виконати розпізнавання даних з тестової вибірки та порівняти її результати з вірними відповідями – тобто з розставленими людиною мітками.

Обчислити матрицю невідповідності можна з використанням функції **confusion_matrix**, на вхід якої слід надати вектор міток (дійсні правдиві класи зображень, **y_true**) та вектор результатів розпізнавання (**y_predict**). Обидва вектори мають містити мітки у вигляді чисел 0 та 1.

Формат виклику функції є наступним:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_true, y_predict)
```

де **y_true** – достовірні мітки;
y_predict – результат розпізнавання.

Результатом виконання функції є двовимірний масив, рядками якого є дійсні класи зображень, а колонками – результати їх розпізнавання. Елементи масиву можна представити через позначки матриці невідповідностей наступним чином:

```
TP = cm[1, 1]
FP = cm[0, 1]
FN = cm[1, 0]
TN = cm[0, 0]
```

Для графічного відображення матриці невідповідностей бібліотека ***scikit-learn*** надає метод **from_predictions** класу **ConfusionMatrixDisplay,** який має наступний формат виклику:

```
ConfusionMatrixDisplay.from_predictions(y_true, y_predict, cmap,
                                        display_labels)
```

де **y_true** – достовірні мітки;
y_predict – результат розпізнавання;
cmap – не обов'язковий параметр, кольорова мапа відображення;
display_labels – не обов'язковий параметр, масив текстових назв класів, в порядку збільшення значень міток класів.

Код відображення таблиці невідповідностей та результат його виконання наведено на рис. 2.8:

```
[85]: y_predict = model.predict(np.reshape(x_test, (-1,1)))
      class_names = ["empty", "occupied"]
      ConfusionMatrixDisplay.from_predictions(y_test, y_predict, display_labels=class_names, cmap=plt.cm.Blues)

[85]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1fcb70b9e50>
```

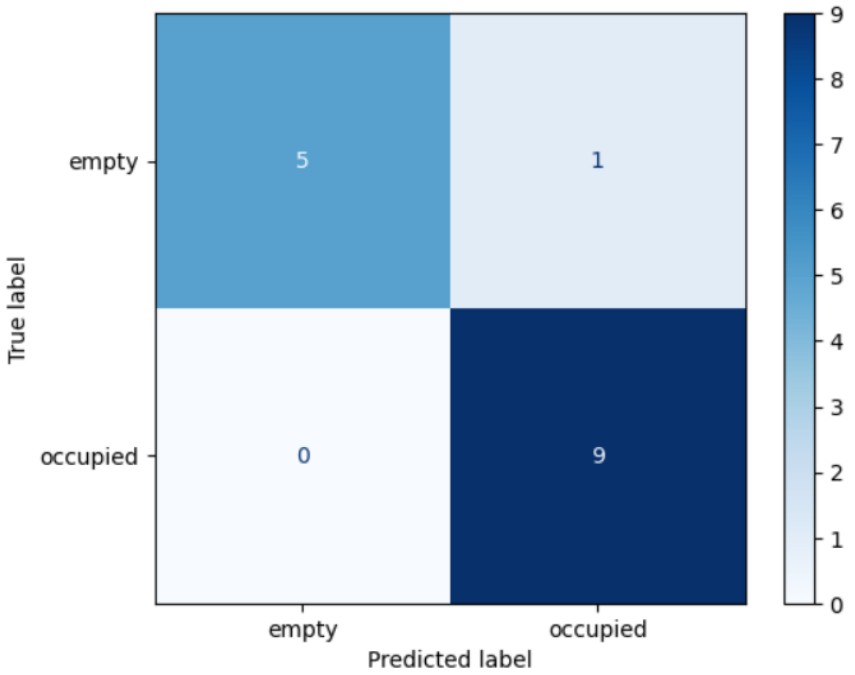

Рисунок 2.8 – Матриця невідповідностей для розпізнавання зайнятості паркомісця

Для обчислення влучності та повноти в бібліотеці ***scikit-learn*** реалізовано функції **precision_score** та **recall_score**, Обидві приймають першим параметром вектор міток, а другим – вектор результатів розпізнавання. Для обчислення даних показників якості можна використати наступний код:

```
from sklearn.metrics import precision_score, recall_score
precision = precision_score(y_test, y_predict)
recall = recall_score(y_test, y_predict)
```

Також значення влучності і повноти, як і інші метрики якості системи розпізнавання можна обчислити на основі матриці невідповідностей.

3. ВАРІАНТИ ЗАВДАНЬ

Для виконання лабораторної роботи необхідно обрати два класи зображень для розпізнавання. Рекомендовано обирати не дуже складні для розпізнавання класи. У якості прикладів подібних класів можна використати наступні варіанти:

1. Яблука та банани
2. Пусті та заповнені коробки
3. Хмарне та ясне небо
4. Літній та зимній пейзажі
5. Цегляна та бетонна поверхні
6. Квіти чи трава
7. Небо чи поле
8. Дорожній знак зупинки або пішохідного переходу
9. Ялинка чи сніговик
10. Дерева за снігом чи без нього

Контрольні запитання

1. Як запустити Jupyter Notebook та виконати комірку з кодом?
2. Які проблеми виникають при обробці та розпізнаванні зображень з каналом прозорості?
3. Які дії передбачають етапи отримання та очистки даних?
4. Чому визначення процедури перетворення образу на вектор ознак є таким важливим для успішності системи розпізнавання?
5. Для чого потрібен етап моніторингу у життєвому циклі системи машинного навчання?
6. Як можна відобразити зменшені зображення з датасету у вигляді таблиці?
7. Для чого перетворювати зображення з формату RGB у формати YUV та Lab?
8. Як побудувати гістограму яскравості точок зображення?
9. Для чого треба виконувати розбиття датасету на тренувальну та тестову вибірки?
10. Якими засобами бібліотеки scikit-learn можна зробити випадкове розбиття датасету на тренувальну та тестову вибірки?
11. Що таке перенавчання?
12. Чому точність не є ідеальним показником якості системи розпізнавання?

СПИСОК ЛІТЕРАТУРИ

1. Project Jupyter. URL: https://jupyter.org/.
2. Markdown Guide. Basic syntax. URL: https://www.markdownguide.org/basic-syntax/.
3. Scikit-image documentation. URL: https://scikit-image.org/docs/stable/.
4. NumPy Documentation. URL: https://numpy.org/doc/stable/.
5. Методичні вказівки до лабораторної роботи «Основи роботи з бібліотекою Matplotlib» з курсу «Обробка даних Python» для студентів спеціальностей 121 Інженерія програмного забезпечення, 122 Комп'ютерні науки, 124 Системний аналіз, 126 Інформаційні системи і технології / уклад. : С. М. Коваленко, С.В. Коваленко, О. В. Шматко. – Харків : НТУ «ХПІ», 2021. 28 с.
6. Matplotlib documentation. URL: https://matplotlib.org/stable/index.html.
7. Open CV documentation. URL: https://docs.opencv.org/4.10.0/index.html.
8. Scikit-learn documentation. URL: https://scikit-learn.org/stable/.
9. Харченко В.О. Основи машинного навчання. Суми: Сумський державний університет, 2023. 264 с.

Навчальне видання

Методичні вказівки до лабораторної роботи
«Створення системи розпізнавання зображень на основі регресійного класифікатору»

для студентів спеціальності 122 «Комп’ютерні науки»

Укладач:
КОЛБАСІН Вячеслав Олександрович

Відповідальний за випуск Ю. І. Дорофєєв
Роботу до видання рекомендував М. І. Безменов
Комп’ютерна верстка В. О. Колбасін

У авторській редакції

План 2024 р., поз. 582

Підп. до друку 15.12.2024 р. Гарнітура Таймс. Ум. друк. арк. 1.
Електронне видання

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2