МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

С.Б. Могильний

МАШИННЕ НАВЧАННЯ В РАДІОТЕХНІЧНИХ КОМП'ЮТЕРИЗОВАНИХ СИСТЕМАХ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра за освітніми програмами
«Радіотехнічні комп'ютеризовані системи», «Інформаційна та комунікаційна
радіоінженерія», «Інтелектуальні технології радіоелектронної техніки»
спеціальності 172 «Електронні комунікації та радіотехніка»*

Київ
КПІ ім. Ігоря Сікорського
2023

Рецензент *Мосійчук Віталій Сергійович*, канд. техн. наук, доц. кафедри прикладної радіоелектроніки радіотехнічного факультету, Національний технічний університет КПІ ім. Ігоря Сікорського

Відповідальний редактор *Жук Сергій Якович*, д-р техн. наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 8 від 02.06.2023 р.)
за поданням Вченої ради радіотехнічного факультету (протокол № 05/2023 від 28.04.2023 р.)

Електронне мережне навчальне видання

Автор: *Могильний Сергій Борисович*, канд. техн. наук, доц.

МАШИННЕ НАВЧАННЯ В РАДІОТЕХНІЧНИХ КОМП’ЮТЕРИЗОВАНИХ СИСТЕМАХ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

Машинне навчання в радіотехнічних комп’ютеризованих системах: Лабораторний практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 172 «Електронні комунікації та радіотехніка» / КПІ ім. Ігоря Сікорського; автор: С.Б.Могильний. – Електронні текстові дані (1 файл: 3,26 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2023. – 175 с.

В навчальному посібнику наводяться рекомендації до виконання лабораторних робіт з кредитного модуля «Машинне навчання в радіотехнічних комп’ютеризованих системах», який викладається студентам радіотехнічного факультету, що навчаються на спеціальності 172 «Електронні комунікації та радіотехніка».

Реєстр. № НП 22/23-724. Обсяг 7,28 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© С.Б.Могильний
© КПІ ім. Ігоря Сікорського, 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ………… 6
ВСТУП………………………………………………………. 7
ЛР 1. ВСТАНОВЛЕННЯ TENSORFLOW НА WINDOWS ТА ВИКОРИСТАННЯ БЛОКНОТА JUPYTER…………….……. 11
1.1. Загальні відомості..…………………….……...……………. 11
1.2. Встановлення Anaconda і TensorFlow..……………..……… 12
1.3. Використання Jupyter Notebook……………………..…...…… 20
1.4. Завдання…………………………………….………………… 26
1.5. Зміст звіту……………………………………………………. 27
1.6. Контрольні питання……………………………………………. 27
2. ЛР 2. ВИКОРИСТАННЯ БІБЛІОТЕКИ PANDAS PYTHON ДЛЯ РОБОТИ З ДАНИМИ…………………………………………. 28
2.1. Теоретичні відомості……………………………………... 28
2.2. Завдання………………..………………………………. 40
2.3. Зміст звіту…………………………………………………… 40
2.4. Контрольні питання……………………………………….…… 40
3. ЛР 3. БІБЛІОТЕКА NUMPY…………………………….…... 41
3.1. Основи NumPy…………………………………………..… 41
3.2. Завдання…………………………………………………… 57
3.3. Зміст звіту……………………………………………….… 57
3.4. Контрольні питання…………………………………….…. 57
4. ЛР 4. РЕАЛІЗАЦІЯ ЛІНІЙНОЇ РЕГРЕСІЇ НА PYTHON..…….. 58
4.1. Теретичні відомості ………………………………………… 58
4.2. Виконання роботи……………………………………….…. 65
4.3. Завдання……………………….………...………………. 70
4.4. Зміст звіту…………………………………………….…. 71
4.5. Контрольні питання…………………………………….…… 71

5. ЛР 5 ПОБУДОВА ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ З TENSORFLOW……..……..……………………….………..….. 72
5.1. Загальні відомості …………..……….………………………… 72
5.2. Початок роботи з TensorFlow………………………………… 75
5.3. Робота з реальними даними……………………………….. 78
5.4. Моделювання нейронної мережі…………………………….. 91
5.5. Завдання……………………………………………..……… 98
5.6. Зміст звіту………………………………………………………… 98
5.7. Контрольні питання…………………………………………… 98
6. ЛР 6. ПОЧАТОК РОБОТИ З KERAS, DEEP LEARNING І PYTHON……………………………………………………………….. 99
6.1. Теоретичні відомості..………………………………………… 99
6.2. Завдання…………………………………………………………... 119
6.3. Зміст звіту………………………………………………………… 119
6.4. Контрольні питання…………………………………………… 119
7. ЛР 7. НАВЧАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ З KERAS..………………………………………………………………… 120
7.1. Загальні відомості………………………………….……...… 120
7.2. Реалізація SmallVGGNet…………………….……..……….. 121
7.3. Завдання…………………………………………………….…... 133
7.4. Зміст звіту………………………………………………………. 133
7.5. Контрольні питання……………………………………….…… 133
8. ЛР 8. КЛАСИФІКАТОР ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ АЛГОРИТМУ K-NN..…………………………………………………. 134
8.1. Загальні відомості…………………….…....……….……….. 134
8.2. Початкові налаштування для глибокого навчання…………. 136
8.3. Реалізація k-NN………………………………………………… 146
8.4. Завдання………………………………………………………… 153
8.5. Зміст звіту……………………………………………….……… 153
8.6. Контрольні питання…………………………………….……… 153

9. ЛР 9. ВИКОРИСТАННЯ МІКРОКОМП'ЮТЕРА RASPBERRY PI ДЛЯ МАШИННОГО НАВЧАННЯ…..………….. 154
9.1. Загальні відомості………………………………………….….… 154
9.2. Встановлення OpenVINO OpenCV на Raspberry Pi…………… 158
9.3. Виявлення об'єктів у реальному часі за допомогою RPi та OpenVINO………………………………………………………. 166
9.4. Завдання……………………………………………...….. 172
9.5. Зміст звіту…………………………………………….…..… 172
9.6. Контрольні питання…………………………………..… 173
РЕКОМЕНДОВАНА ЛІТЕРАТУРА………………..……………… 174

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ЄСКД	єдина система конструкторської документації
ІС	інтегральна схема
ЛР	лабораторна робота
ООП	об'єктно-орієнтоване програмування
ПЗ	програмне забезпечення
ПОП	подійно орієнтоване програмування
CIFAR	Canadian Institute For Advanced Research
CNN	Convolutional Neural Network
DL	Deep Learning
DMA	Direct Memory Access
FPGA	Field-Programmable Gate Array
FPS	Frames Per Second
IDE	Integrated Development Environment
MNIST	Modified National Institute of Standards and Technology database
MSE	Mean Squared Error
NCS	Neural Compute Stick
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RPi	Raspberry Pi
SHAVE	Streaming Hybrid Architecture Vector Engine
SGD	Stochastic Gradient Descent
SPI	Serial Peripheral Interface Bus
GPU	Graphics Processing Unit
CPU	Central Processing Unit
VPU	Vision Processing Unit

ВСТУП

Даний навчальний посібник призначений для підготовки бакалаврів за освітніми програмами «Радіотехнічні комп'ютеризовані системи», «Інформаційна та комунікаційна радіоінженерія», «Інтелектуальні технології радіоелектронної техніки» спеціальності 172 «Електронні комунікації та радіотехніка» і може бути використаний для інших освітніх програм.

Підготовка спеціалістів за дисципліною «Машинне навчання в радіотехнічних комп'ютеризованих системах» передбачає 36 годин лекцій, 18 годин лабораторних занять, модульну контрольну роботу та розрахункову роботу, які орієнтовані на отримання знань і навичок побудови моделі машинного навчання для розпізнавання образів та оптимізації задач управління в радіотехнічних системах.

Основна мета посібника – сформувати у студентів в процесі виконання лабораторних робіт навички реалізації алгоритмів машинного навчання для оптимізації інженерних рішень. Такі задачі вимагають знань системного проєктування, програмування, алгоритмів роботи різних існуючих бібліотек програмного забезпечення. Виконання лабораторних робіт дозволяє закріпити теоретичні знання, отримані на лекціях.

Навчальний посібник підготовлено відповідно до робочої навчальної програми (силабусу) дисципліни «Машинне навчання в радіотехнічних комп'ютеризованих системах». Він містить необхідний теоретичний матеріал, який дозволяє застосовувати набуті знання для реалізації алгоритмів машинного навчання в радіотехнічних системах.

Змістом даного видання є методичні рекомендації до виконання та опис лабораторних робіт кредитного модуля «Машинне навчання в радіотехнічних комп'ютеризованих системах». У вказівках до кожної лабораторної роботи наведені теоретичні питання в необхідному для виконання лабораторних робіт об'ємі.

Виконання лабораторних робіт

Перед виконанням лабораторних робіт студенти повинні самостійно вивчити відповідні розділи дисципліни за рекомендованою літературою і конспектом лекцій, зміст лабораторної роботи, порядок її виконання. В режимі ofline перед початком виконання кожної роботи студенти проходять контроль знань, на якому повинні показати розуміння мети і змісту роботи. Студенти, які отримали незадовільну оцінку, до лабораторної роботи не допускаються. Приступати до виконання лабораторної роботи можна тільки з дозволу викладача. Результати виконання лабораторної роботи у вигляді працюючих схем, сценаріїв, результати виконання при симуляції реалізованого проєкта тощо надаються викладачу.

Зміст та оформлення звіту

У звіті необхідно відобразити:

- мету роботи;
- стислі теоретичні відомості;
- алгоритм та коди на мові сценаріїв Python;
- аналіз отриманих результатів та висновки.

Звіти з лабораторних робіт виконуються кожним студентом індивідуально на окремих аркушах формату А4. На координатних осях графіків повинні бути позначенням масштабу та розмірності вимірюваних величин. У формулах та електричних схемах необхідно використовувати умовні позначення, які відповідають стандартам ЄСКД або використаної САПР. Особливу увагу необхідно приділити аналізу отриманих результатів та формулюванню висновків за результатами роботи.

Титульний лист звіту оформляється наступним чином:

Національній технічних університет України

Київський політехнічних інститут імені Ігоря Сікорського

Радіотехний факультет

Кафедра радіотехнічних систем

ЗВІТ

про лабораторну роботу ___

з дисципліни «Машинне навчання в радіотехнічних комп'ютеризованих системах»

____

(назва роботи)

студента першого курсу, групи_________________

____

(прізвище, ім'я та по-батькові)

Викладач Могильний С.Б.:

Дата___________

202_ р.

Для забезпечення дистанційного (online) навчання використовуються платформи Zoom та Moodle, які дозволяють проводити дистанційно лекційні та вступні заняття перед роботою (Zoom). Також при дистанційному навчанні студенти отримують цілодобовий доступ до мікрокомп'ютерів з Інтернету. Роз'яснення (консультації) зі складних та незрозумілих питань проводяться на онлайн зустрічах в Zoom.

Контроль виконання робіт та їх захист здійснюється на платформі дистанційного навчання Moodle. Протоколи виконання робіт завантажуються у

відповідний розділ дисципліни в Moodle. Треба звернути увагу при оформлені звітів робіт на дотримання вимог ДСТУ 3008:2015. Посилання на дистанційний курс – http://iot.kpi.ua/lms/course/view.php?id=6.

Зв'язок зі студентами підтримується через Телеграм–групу «Машинне навчання» та електронну пошту старост груп, що забезпечує своєчасну видачу та контроль індивідуальних завдань у відповідності до графіку навчання.

Отримані бали відповідно до рейтингової системи оцінювання накопичуються в журналі оцінок Moodle та в розділі «Поточна інформація» Кампуса КПІ (https://ecampus.kpi.ua). В Кампусі також викладені всі навчальні матеріали курсу.

Лабораторна робота 1

ВСТАНОВЛЕННЯ TENSORFLOW НА WINDOWS ТА ВИКОРИСТАННЯ БЛОКНОТА JUPYTER

Мета роботи: Ознайомитися із встановленням та початковими налаштуваннями програмного забезпечення, який дозволяє роботу фреймворку TensorFlow.

Зміст. В роботі вивчаються встановлення віртуального середовища Anaconda та використання його для роботи з фреймворком TensorFlow. Детально розглянуто використання блокнотам Jupyter для створення та роботи із сценаріями Python.

1.1. Загальні відомості

Надалі в курсі будемо використовувати фреймворк TF, тому пояснимо, як встановити TF за допомогою Anaconda та як користуватися TF з Jupyter – переглядачем у вигляді блокнота.

TF підтримує одночасне обчислення на декількох звичайних і графічних процесорах. Це означає, що обчислення можуть бути розподілені між пристроями для збільшення швидкості навчання. За допомогою паралелізму нам не треба чекати тижнями, щоб отримати результати алгоритмів тренувань.

Для користувача Windows TF пропонує дві версії:

• TF лише з підтримкою CPU: якщо наш комп’ютер працює не на NVIDIA GPU, то можемо встановити лише дану версію.

• TF з підтримкою GPU: для більш швидкого обчислення можемо використовувати дану версію TF. Вона має сенс лише тоді, коли потрібні потужні обчислювальні можливості.

В нашому посібнику достатньо базової версії TF з підтримкою CPU.

Примітка. TF не забезпечує підтримку GPU на MacOS і мікрокомп’ютері Raspberry Pi.

Щоб запустити TF з блокнотом Jupyter, треба створити середовище в Anaconda. Це означає, що треба встановити Ipython, Jupyter та TF у відповідну папку на комп'ютері. Крім цього, треба додати одну важливу бібліотеку для наукових даних: Pandas. Бібліотека Pandas допомагає маніпулювати кадром даних.

1.2. Встановлення Anaconda

Пакет Anaconda включає в себе інтерпретатор мови Python (є версії 2 і 3), набір бібліотек, які найчастіше використовуються, і зручне середовище розробки та виконання, яке запускається в браузері.

Для встановлення пакета попередньо треба завантажити дистрибутив.

Завантажуємо Anaconda (https://www.anaconda.com/download/) версії 2022.05 (Python 3.9) для відповідної системи.

Anaconda допоможе керувати всіма бібліотеками, необхідними для Python або R. Пропонуються варіанти для Windows, Linux і MacOS.

Встановлення Anaconda на Windows

1. Запустимо завантажений інсталятор. В першому вікні треба натиснути "Next" (рис.1.1):

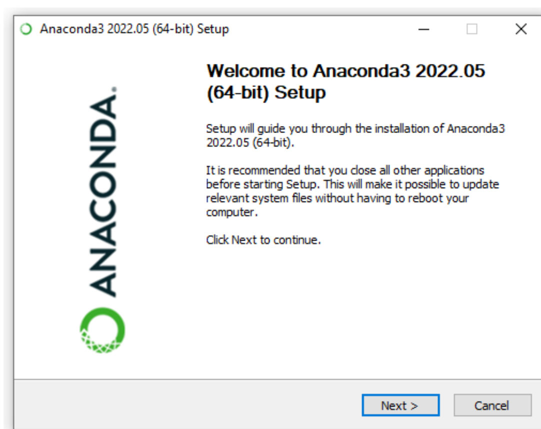

Рисунок 1.1 – Запуск інсталятора

2. Приймаємо ліцензійну угоду (рис.1.2):

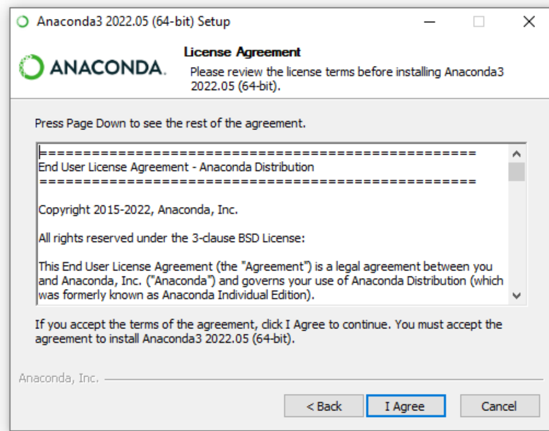

Рисунок 1.2 – Ліцензійна угода

3. Обираємо одну з опцій встановлення (рис.1.3):

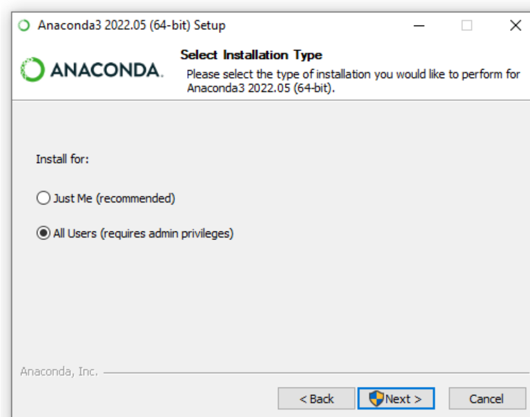

Рисунок 1.3 – Вибір опцій встановлення

- Just Me – лише для користувача, який запустив встановлення.
- All Users – для всіх користувачів.

4. Вказуємо шлях, за яким буде встановлена Anaconda (рис.1.4):

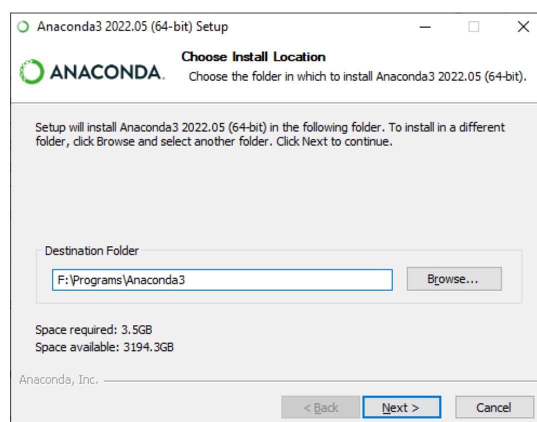

Рисунок 1.4 – Шлях до папки встановлення

5. Задаємо додаткові опції (рис.1.5):

- Add Anaconda to the system PATH environment variable – додати Anaconda до системної змінної PATH.
- Register Anaconda as the system Python 3.9 – використовувати Anaconda, як інтерпретатор Python 3.9 за замовчуванням.

Для початку встановлення натискаємо кнопку “Install”.

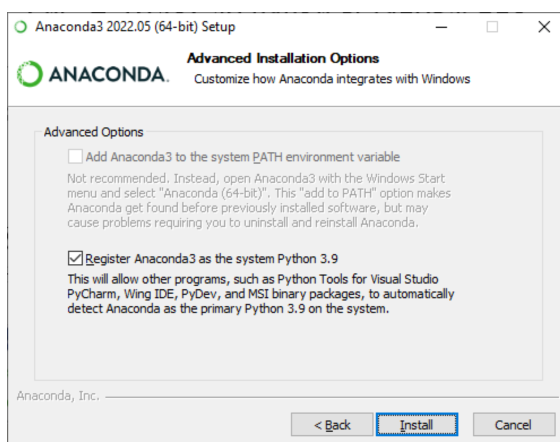

Рисунок 1.5 – Додаткові опції встановлення

6. Після цього буде виконане встановлення Anaconda (рис.1.6):

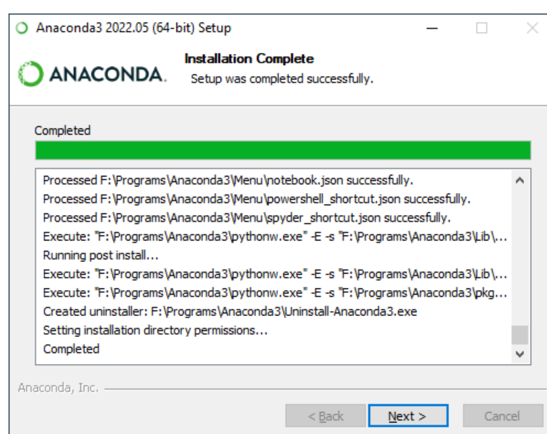

Рисунок 1.6 – Завершення встановлення Anaconda

Створення .yml-файла для встановлення TensorFlow та залежностей

Необхідно виконати ще кілька кроків, щоб завершити встановлення TF.

Крок 1. Знаходження Anaconda

Перший крок, який треба зробити, – це знайти шлях до Anaconda. Потім створимо нове середовище conda, яке включає необхідні бібліотеки, які будемо використовувати під час вивчення TF.

В ОС Windows можемо запустити Anaconda Prompt з меню та набрати (рис.1.7):

```
C:\Users\PC>where anaconda
```

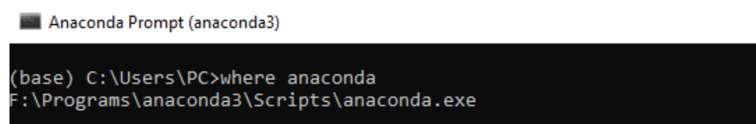

Рисунок 1.7 – Знаходження Anaconda

В користувача цей шлях може бути іншим, але варто знати назву папки, в якій встановлена Anaconda, оскільки нове середовище створюємо всередині цієї папки. Наприклад, на рис.1.7 Anaconda встановлена в папці PC. У нас це може бути, наприклад, ім'я користувача.

Далі встановимо робочий каталог.

Треба створити нову папку всередині Anaconda, в якій будуть розміщені Ipython, Jupyter та TF. Швидкий спосіб встановити бібліотеки та програмне забезпечення – написати файл .yml.

Крок 2. Встановлення робочого каталогу

Треба вказати робочий каталог, в якому створимо файл .yml. Як було сказано раніше, він буде розташований всередині Anaconda.

Для цього в ОС Windows запускаємо Anaconda Prompt з правами адміністратора і вводимо двома рядками (спочатку подивимося шлях до папки з Anaconda3) (рис.1.8):

```
F:
cd \Programs\Anaconda3
```

Безпосередній перехід за допомогою команди `cd` в папку на іншому диску в Windows 10 не працює, а права адміністратора необхідні, бо ми обрали параметр `All Users` (рис.1.3).

```
Administrator: Anaconda Prompt (Anaconda3)

(base) C:\Windows\system32>f:

(base) F:\>cd \Programs\Anaconda3
```

Рисунок 1.8 – Робочий каталог Anaconda

Крок 3. Створення файла .yml

Можемо створити файл .yml всередині нового робочого каталогу. У файлі встановлюються залежності, необхідні для запуску TF. Скопіюємо та вставимо в Terminal наведений нижче код:

```
echo.>hello-tf.yml
```

З'явиться новий файл з назвою `hello-tf.yml` (рис.1.9):

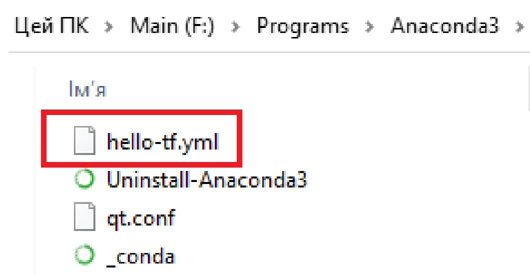

Рисунок 1. 9 – Файл `hello-tf.yml` в робочому каталозі

Крок 4. Редагування файла .yml

Скористаємося Блокнотом для виконання наступного кроку:

```
notepad hello-tf.yml
```

Додаємо у файл зміст:

```
name: hello-tf
dependencies:
- python=3.9
- jupyter
- ipython
- pandas
```

Пояснення коду:

`name: hello-tf`: ім'я файла `.yml`,

`dependencies`: залежності:

```
python = 3,9
jupyter
ipython
```

`pandas`: встановлює бібліотеки Python версії 3.9, бібліотеки Jupyter, Ipython та pandas

Відкриваємо блокнот в Anaconda і можемо редагувати файл в ньому (рис.1.10):

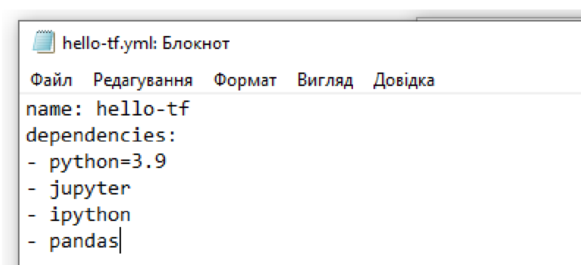

Рисунок 1.10 – Редагування файла `hello-tf.yml`

Примітка. Користувачі Windows встановлюють TF на наступному кроці. На даному кроці ми лише підготували conda-середовище.

Крок 5. Компілюємо файл .yml

Файл.yml можемо компілювати за допомогою такого коду (рис.1.11):

```
conda env create -f hello-tf.yml
```

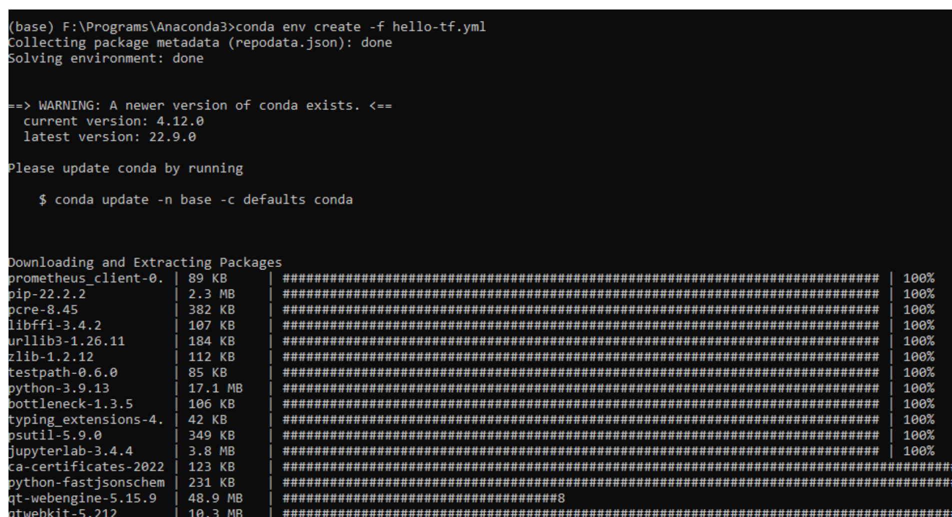

Рисунок 1.11 – Компілювання файлу `.yml`

Примітка. Нове середовище створюється всередині поточного каталогу користувачів.

Це потребує часу і займе близько 1,1 ГБ місця на жорсткому диску.

Крок 6. Активізація середовища conda

Зараз маємо дві conda (рис.1.12). Ми створили ізольоване середовище conda з бібліотеками, якими будемо користуватися під час навчання. Це рекомендована практика, оскільки кожен проєкт машинного навчання потребує різних бібліотек. Коли проєкт закінчимо, можемо це середовище видалити або залишити.

```
conda env list
```

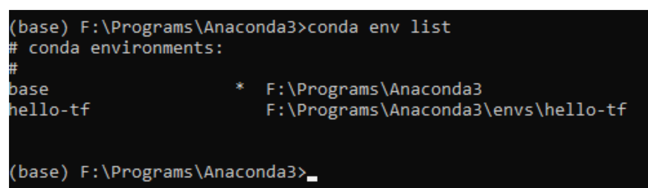

Рисунок 1.12 – Два середовища conda

Бачимо два середовища conda. Основне і новостворене в `\envs\hello-tf`. Основне середовище conda ще не має TF, встановлено лише `hello-tf`. Зірочка вказує на замовчування. Нам потрібно переключитися на `hello-tf`, щоб активувати середовище (рис.1.13):

```
activate hello-tf
```

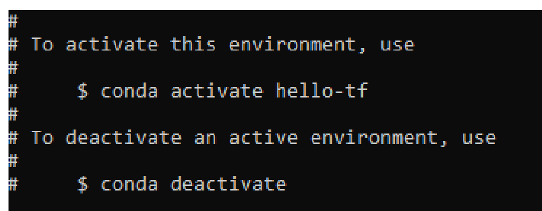

Рисунок 1.13 – Активація/деактивація середовища

Тепер можемо перевірити, чи всі залежності знаходяться в одному середовищі (рис.1.14). Це важливо, оскільки дозволяє Python використовувати Jupyter та TF з одного середовища. Якщо не побачимо всіх трьох в одній папці, то треба буде почати все заново.

Крок 7. Встановлення TensorFlow

```
where python
where jupyter
where ipython
```

На рис. 1.14 видно, що Python, Jupyter та Ipython встановлені в одному середовищі. Це означає, що можна використовувати TF із блокнотом Jupyter.

```
(base) F:\Programs\Anaconda3>activate hello-tf

(hello-tf) F:\Programs\Anaconda3>where python
F:\Programs\Anaconda3\python.exe
F:\Programs\Anaconda3\envs\hello-tf\python.exe
C:\Users\PC\AppData\Local\Microsoft\WindowsApps\python.exe

(hello-tf) F:\Programs\Anaconda3>where jupyter
F:\Programs\Anaconda3\envs\hello-tf\Scripts\jupyter.exe
F:\Programs\anaconda3\Scripts\jupyter.exe

(hello-tf) F:\Programs\Anaconda3>where ipython
F:\Programs\Anaconda3\envs\hello-tf\Scripts\ipython.exe
F:\Programs\anaconda3\Scripts\ipython.exe

(hello-tf) F:\Programs\Anaconda3>
```

Рисунок 1.14 – Встановлені `Python`, `Jupyter` та `Ipython`

Тепер можна встановити TF за допомогою наступної команди (рис.1.15):

```
pip install tensorflow
```

```
Collecting oauthlib>=3.0.0
  Downloading oauthlib-3.2.1-py3-none-any.whl (151 kB)
     ---------------------------------------- 151.7/151.7 kB 9.4 MB/s eta 0:00:00
Installing collected packages: tensorboard-plugin-wit, pyasn1, libclang, keras, flatbuffers, wrapt, werkzeug, termcolor, tensorflow-io-gcs-filesystem, tensorflow-estimator, tensorboard-data-server, rsa, pyasn1-modules, protobuf, opt-einsum, oauthlib, keras-preprocessing, h5py, grpcio, google-pasta, gast, cachetools, astunparse, absl-py, requests-oauthlib, markdown, google-auth, google-auth-oauthlib, tensorboard, tensorflow
Successfully installed absl-py-1.2.0 astunparse-1.6.3 cachetools-5.2.0 flatbuffers-22.9.24 gast-0.4.0 google-auth-2.12.0 google-auth-oauthlib-0.4.6 google-pasta-0.2.0 grpcio-1.49.1 h5py-3.7.0 keras-2.10.0 keras-preprocessing-1.1.2 libclang-14.0.6 markdown-3.4.1 oauthlib-3.2.1 opt-einsum-3.3.0 protobuf-3.19.6 pyasn1-0.4.8 pyasn1-modules-0.2.8 requests-oauthlib-1.3.1 rsa-4.9 tensorboard-2.10.1 tensorboard-data-server-0.6.1 tensorboard-plugin-wit-1.8.1 tensorflow-2.10.0 tensorflow-estimator-2.10.0 tensorflow-io-gcs-filesystem-0.27.0 termcolor-2.0.1 werkzeug-2.2.2 wrapt-1.14.1

(hello-tf) F:\Programs\Anaconda3>
```

Рисунок 1.15 – Встановлення TensorFlow

На цьому можна вважати встановлення закінченим.

Для запуску `jupyter` в Anaconda Promt можна використати послідовність двох команд:

```
activate hello-tf
jupyter notebook
```

Примітка. Якщо працюємо з дистрибутивом `anaconda`, то можемо зробити наступне, щоб використовувати, наприклад, версію python 3.5 у новому середовищі "`tensorflow`":

```
conda create --name tensorflow python=3.5
activate tensorflow
conda install jupyter
```

```
conda install scipy
pip install tensorflow
# or
# pip install tensorflow-gpu
```

Важливо додати `python=3.5` в кінці першого рядка, щоб встановити Python 3.5. Це іноді необхідно, якщо версії TF «не встигають» за версіями Python.

1.3. Використання Jupyter Notebook

Розглянемо детальніше деякі особливості використання Jupyter Notebook. Для цього напишемо рядок простого коду, щоб ознайомитися з оточенням Jupyter.

Крок 1. Додаємо папку всередині робочого каталогу, яка буде містити всі блокноти, які створимо під час вивчення TF.

Відкриємо в меню команд термінал Anaconda Promt і вводимо:

```
mkdir jupyter_tf
jupyter notebook
```

Пояснення коду:

- `mkdir jupyter_tf`: створюємо папку з назвою `jupyter_tf`
- `jupyter notebook`: відкриваємо веб-додаток Jupyter

Крок 2. Можемо побачити нову папку всередині середовища (рис.1.16). Клацнемо папку `jupyter_tf`.

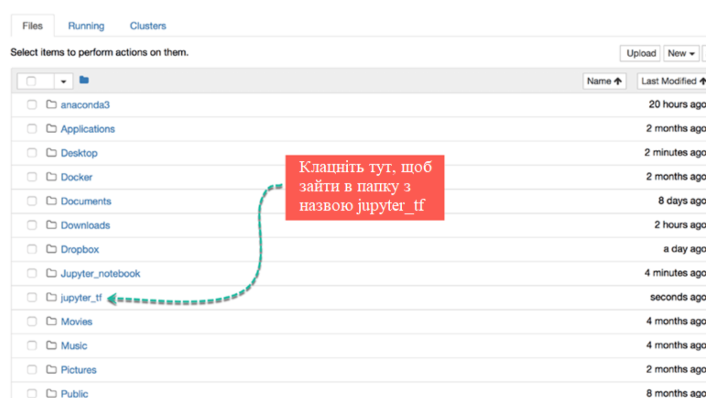

Рисунок 1.16 – Нова папка всередині середовища Jupyter

Крок 3. Всередині цієї папки створимо свій перший блокнот. Клацнемо на кнопці **New**, а потім на **Python 3** (рис.1.17).

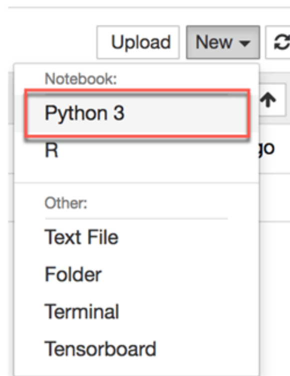

Рисунок 1.17 – Створення блокнота

Крок 4. Ми знаходимось у середовищі Jupyter. Поки що, наш блокнот називається Untitled.ipynb. Це ім'я за замовчуванням, яке дав Jupyter. Давайте перейменуємо його, натиснувши на **File** і вибравши **Rename** (рис.1.18):

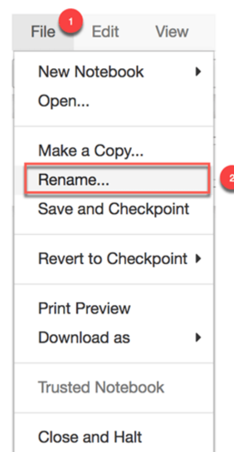

Рисунок 1.18 – Перейменування блокнота

Можемо перейменувати його в Introduction_jupyter (рис.1.19):

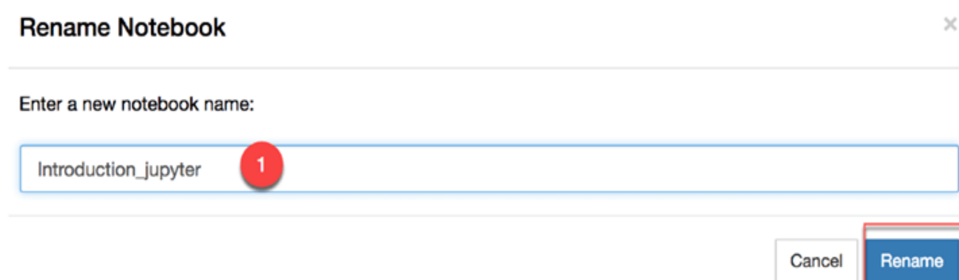

Рисунок 1.19 – Перейменування блокнота в Introduction_jupyter

У блокноті Jupyter пишемо коди, примітки або текст всередині комірок (рис.1.20).

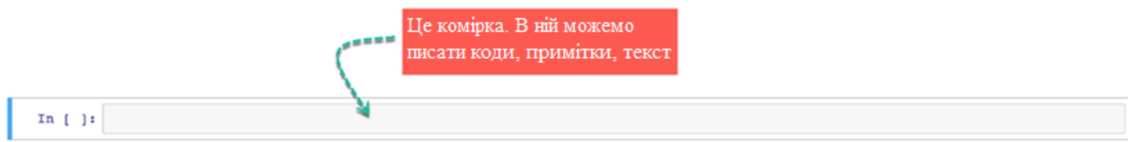

Рисунок 1.20 – Комірка блокнота

Всередині комірки можна написати один рядок коду (рис.1.21):

```
In [4]: import matplotlib.pyplot as plt
```

Рисунок 1.21 – Рядок коду в комірці

або кілька рядків (рис.1.22). Jupyter читає код послідовно рядок за рядком:

```
In [1]: import numpy as np
        import pandas as pd
        from scipy import stats, integrate
```

Рисунок 1.22 – Кілька рядків коду в комірці

Наприклад, напишемо далі наступний код всередині комірки (рис.1.23):

```
In [6]: %matplotlib inline
        import seaborn as sns
        sns.set(color_codes=True)
        np.random.seed(sum(map(ord, "distributions")))
        x = np.random.normal(size=100)
        sns.displot(x)
```

Рисунок 1.23 – Код всередині комірки

В результаті виконання даного коду буде виведено (рис.1.24):

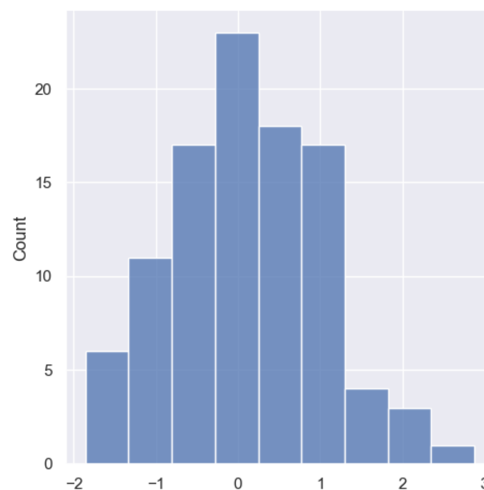

Рисунок 1.24 – Результат виконання коду

Крок 5. Тепер готові написати свій перший рядок коду. Звернемо увагу, що клітина має два кольори. Зелений колір означає, що ми перебуваємо в **editing mode** (режимі редагування) (рис.1.25).

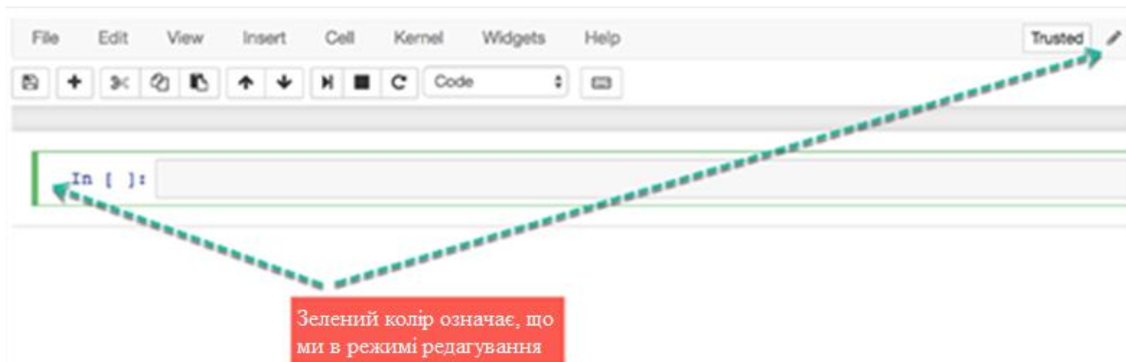

Рисунок 1.25 – Режим редагування

А синій колір вказує на перебування в режимі виконання (**executing mode**) (рис.1.26).

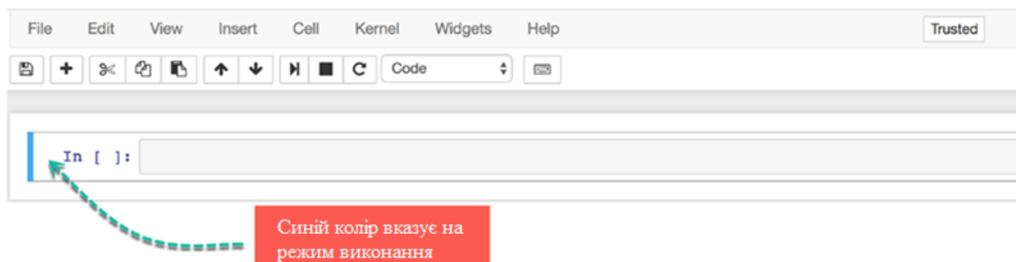

Рисунок 1.26 – Режим виконання

Перший рядок коду буде виводити Guru99!. Всередині клітинки можна написати:

```
print("Guru99!")
```

Є два способи запуску коду в Jupyter:

• Клацнути та запустити (**Run**).

• Скористатися гарячими клавішами.

Щоб запустити код, можемо натиснути на **Cell**, а потім **Run Cells and Select Below** (рис.1.27):

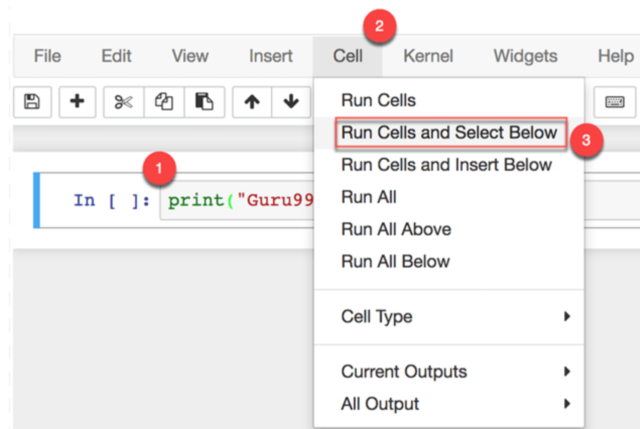

Рисунок 1.27 – Запуск коду з меню

Можемо бачити, що код виводиться під коміркою, а нова комірка з'явилася відразу після виводу (рис.1.28):

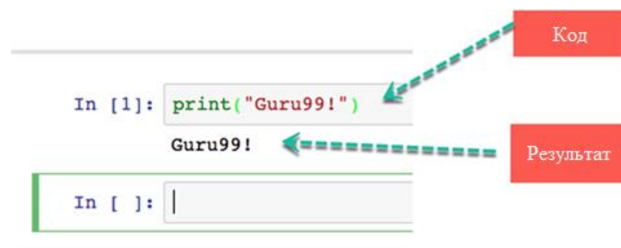

Рисунок 1.28 – Виконання коду

Більш швидкий спосіб запуску коду – це використання комбінацій клавіш. Щоб отримати доступ до комбінацій клавіш, перейдіть до **Help** та **Keyboard Shortcuts** (рис.1.29):

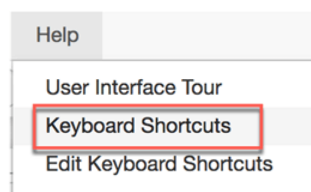

Рисунок 1.29 – Доступ до комбінації клавіш

Комбінації клавіш для Windows (рис.1.30):

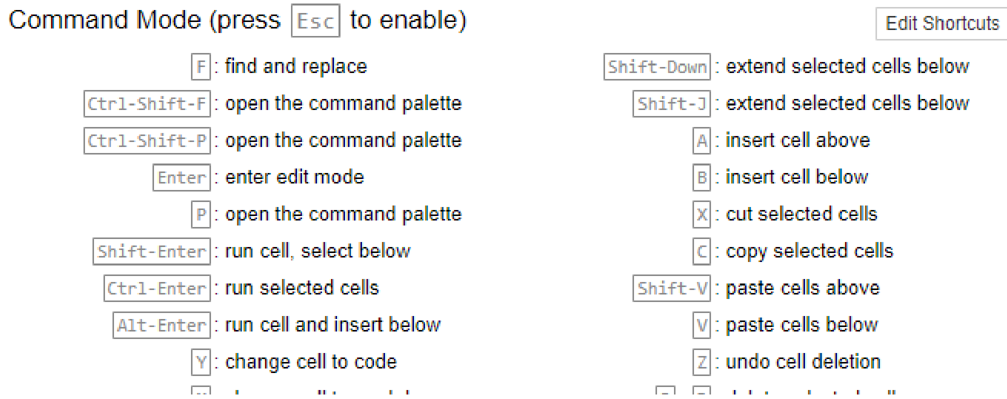

Рисунок 1.30 – Комбінації клавіш для Windows

Напишемо рядок:

```
print("Hello world!")
```

і спробуємо використати клавіші швидкого доступу для запуску коду. Натиснемо alt+Enter: буде виконаний код в комірці і вставлена нова порожня комірка внизу, як і раніше (рис.1.31).

Рисунок 1.31 – Використання клавіш швидкого доступу

Крок 6. Настав час закрити блокнот. Перейдемо у **File** і натиснемо кнопку **Close and Halt** (рис.1.32):

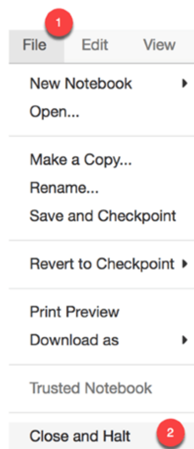

Рисунок 1.32 – Закривання блокнота

Примітка. Jupyter автоматично зберігає блокнот з контрольною точкою. Може з'явитися таке повідомлення (рис.1.33):

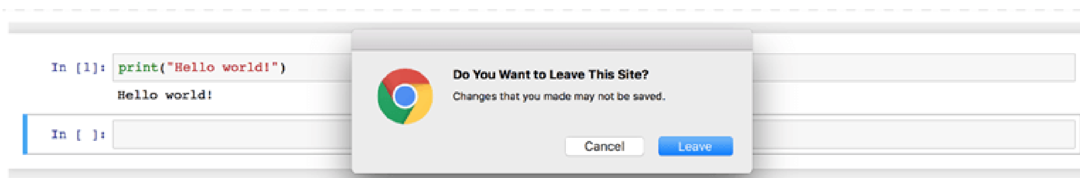

Рисунок 1.33 – Повідомлення при незбереженні файла

Це означає, що Jupyter не зберіг файл з останньої контрольної точки. Можемо вручну зберегти блокнот (рис.1.34):

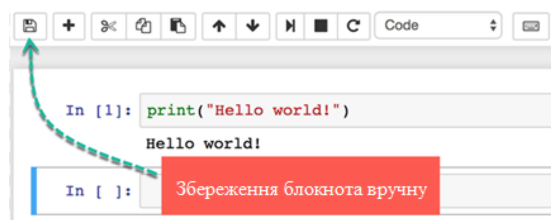

Рисунок 1.34 – Збереження блокнота

Нас перенаправлять на головну панель. Можна побачити, що блокнот був збережений хвилину тому (рис.1.35). Тепер можна безпечно вийти.

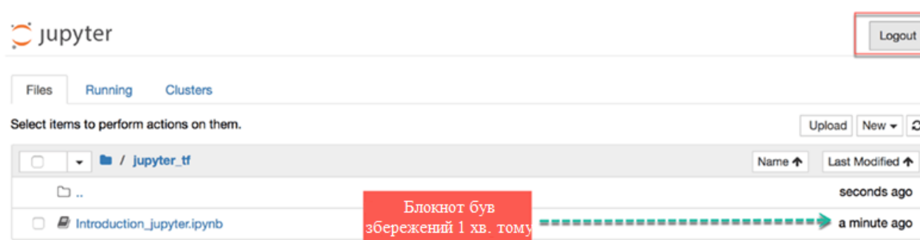

Рисунок 1.35 – Блокнот, збережений хвилину тому

1.4. Завдання

1. Повторити вправи, наведені в тексті модуля.
2. Пояснити призначення команд кожного з рядків коду, який виводить графік (рис.1.29).

1.5. Зміст звіту

1. Мета роботи.

2. Скріншоти результатів виконання завдання.

3. Відповіді на контрольні питання до роботи.

4. Короткі висновки.

1.6. Контрольні питання

1. Для чого треба встановлювати Anaconda на Windows?

2. Для чого треба встановлювати Anaconda на Linux?

3. Чим відрізняється Jupyter від звичних нам IDE для Python?

4. Для чого всі встановлені залежності повинні знаходитися в одному середовищі?

Лабораторна робота 2

ВИКОРИСТАННЯ БІБЛІОТЕКИ PANDAS PYTHON ДЛЯ РОБОТИ З ДАНИМИ

Мета роботи: Вивчення можливостей та набуття навиків створення, маніпуляції та зміни даних за допомогою відкритої бібліотеки Pandas.

Зміст. Вивчається створення кадру даних, перевірка їх та фрагментація, використання бібліотеки для роботи з часовими рядами, об'єднання даних, видалення повторів тощо.

2.1. Теоретичні відомості

Pandas – це бібліотека з відкритим джерелом, яка дозволяє здійснювати маніпуляції з даними в Python. Бібліотека надає дуже зручні з точки зору використання інструменти для зберігання даних і роботи з ними. При аналізі даних або використанні машинного навчання з Python ми просто зобов'язані вміти нею користуватися.

Бібліотека Pandas – високорівнева і побудована зверху низькорівневого Numpy, тобто для роботи Pandas треба Numpy. Pandas не лише надає простий спосіб, щоб створювати, маніпулювати та змінювати дані – це також елегантне рішення для даних часових рядів.

Можливості бібліотеки:

- Об'єкт DataFrame із вбудованим індексуванням для маніпулювання даними.
- Інструменти для зчитування та запису даних між структурами даних у пам'яті та різними форматами файлів.
- Вирівнювання даних та вбудована підтримка пропущених даних.
- Переформатовування для отримання зведених наборів даних.
- Отримання зрізів за мітками, індексування з розширеними можливостями та отримання піднаборів з великих наборів даних.
- Вставляння та вилучення стовпчиків в структурах даних.

- Руйнівник групування, який дозволяє робити з наборами даних операції розділення-зміни-об'єднання (англ. *split-apply-combine*).
- Злиття та з'єднання наборів даних.
- Ієрархічне індексування осей для роботи з даними високої розмірності в структурі даних нижчої розмірності.
- Функціональність для часових рядів: створення діапазонів дат та перетворення частоти, статистики рухливого вікна, лінійні регресії рухливого вікна, зсування дат та запізнювання.

Якщо коротко, Pandas – це корисна бібліотека для аналізу даних. З її допомогою можна виконувати маніпуляції та аналіз даних.

Встановлення Pandas

При встановленні TensorFlow Pandas встановлюється за замовчуванням. Якщо ж чомусь Pandas не встановився, то можемо встановити Pandas за допомогою:

• Anaconda: `conda install -c anaconda pandas`

• У блокноті Jupyter:

```
import sys
!conda install --yes --prefix {sys.prefix} pandas
```

Для запуску Pandas не забуваємо вставити в сценарій рядок:

```
import pandas as pd
```

Кадр даних та серія

Кадр даних – це двовимірний масив з міченими осями (рядки та стовпці). Кадр даних – це стандартний спосіб зберігання даних.

Кадр даних добре відомий статистикам та іншим практикам з даних. Кадр даних – це табличні дані, з рядками для зберігання інформації та стовпцями для іменування інформації. Наприклад, назвою стовпця може бути ціна, 2,3,4 – значення ціни.

Нижче (рис.2.1) зображення кадру даних Pandas:

	Item	Price
0	A	2
1	B	3

Рисунок 2.1 – Приклад кадру даних

Серія – це одновимірна структура даних. Вона може мати будь-який тип даних: цілі, з плаваючою комою та рядкові. Це корисно, коли потрібно виконати обчислення або повернути одновимірний масив. Серія, за визначенням, не може мати більше одного стовпця. Для багатьох стовпців використовуємо структуру кадру даних.

У серії є один параметр:

• Дані: може бути список, словник або скалярне значення:

```
pd.Series([1., 2., 3.])
0    1.0
1    2.0
2    3.0
dtype: float64
```

Можна додати індекс з індексом. Це допомагає назвати рядки. Довжина повинна дорівнювати розміру стовпчика:

```
pd.Series([1., 2., 3.], index=['a', 'b', 'c'])
```

Нижче створимо серію Pandas з відсутнім значенням для третього рядка. Звертаємо увагу, що відсутні значення в Python позначаються "`NaN`". Можемо використати numpy, щоб штучно створити відсутнє значення: `np.nan`

```
pd.Series([1,2,np.nan])
```

Результат виконання:

```
0    1.0
1    2.0
2    NaN
dtype: float64
```

Створення кадру даних

Можна перетворити масив numpy у кадр даних pandas за допомогою `pd.DataFrame()`. Можна зробити і протилежне. Для перетворення кадру даних pandas (`DataFrame`) в масив можна використати `np.array()`:

```
## з Numpy в Pandas
import numpy as np
h = [[1,2],[3,4]]
df_h = pd.DataFrame(h)
print('Data Frame:', df_h)

## з Pandas в Numpy
df_h_n = np.array(df_h)
print('Numpy array:', df_h_n)

Data Frame:    0  1
0  1  2
1  3  4
Numpy array: [[1 2]
[3 4]]
```

Також можна скористатися словником для створення кадру даних Pandas:

```
dic = {'Name': ["John", "Smith"], 'Age': [30, 40]}
pd.DataFrame(data=dic)
```

Результат виконання:

	Age	**Name**
0	30	John
1	40	Smith

Діапазон дат

Pandas має зручний API для створення діапазону дат:

```
pd.data_range(date,period,frequency)
```

• Перший параметр – дата початку

• Другий параметр – кількість періодів (необов'язково, якщо вказана кінцева дата)

• Останній параметр – частота: день: 'D', місяць: 'M' та рік: 'Y'.

```
## Створення дати
# Дні
dates_d = pd.date_range('20300101', periods=6, freq='D')
print('Day:', dates_d)
```

Результат виконання:

```
Day: DatetimeIndex(['2030-01-01', '2030-01-02', '2030-01-03', '2030-01-04', '2030-01-05', '2030-01-06'], dtype='datetime64[ns]', freq='D')
```

```
# Місяці
dates_m = pd.date_range('20300101', periods=6, freq='M')
print('Month:', dates_m)
```

Результат виконання:

```
Month: DatetimeIndex(['2030-01-31', '2030-02-28', '2030-03-31', '2030-04-30','2030-05-31', '2030-06-30'], dtype='datetime64[ns]', freq='M')
```

Перевірка даних

Можна перевірити заголовок або хвіст набору даних за допомогою `head()` або `tail()`, які йдуть попереду назви кадру даних pandas.

Крок 1. Створимо випадкову послідовність з numpy. Послідовність має 4 стовпчики та 6 рядків:

```
random = np.random.randn(6,4)
```

Крок 2. Створимо кадр даних, використовуючи pandas.

Використаємо `dates_m` як індекс для кадру даних. Це означає, що кожен рядок буде мати "name" або індекс, у відповідності з датою.

Нарешті, дамо назви 4-м стовпцям із стовпцями аргументів:

```
# Створення даних з датою
df = pd.DataFrame(random,
                  index=dates_m,
                  columns=list('ABCD'))
```

Крок 3. Використаємо функцію заголовка

```
df.head(3)
```

Результат виконання:

	A	**B**	**C**	**D**
2030-01-31	1.139433	1.318510	-0.181334	1.615822
2030-02-28	-0.081995	-0.063582	0.857751	-0.527374
2030-03-31	-0.519179	0.080984	-1.454334	1.314947

Крок 4. Використаємо функцію хвоста

```
df.tail(3)
```

Результат виконання:

	A	**B**	**C**	**D**
2030-04-30	-0.685448	-0.011736	0.622172	0.104993
2030-05-31	-0.935888	-0.731787	-0.558729	0.768774
2030-06-30	1.096981	0.949180	-0.196901	-0.471556

Крок 5. Чудовою практикою для отримання поняття про дані є використання `describe()`. Команда забезпечує підрахунок, середнє значення, std, min, max та процентиль набору даних.

```
df.describe()
```

Результат виконання:

	A	**B**	**C**	**D**
count	6.000000	6.000000	6.000000	6.000000
mean	0.002317	0.256928	-0.151896	0.467601
std	0.908145	0.746939	0.834664	0.908910
min	-0.935888	-0.731787	-1.454334	-0.527374
25%	-0.643880	-0.050621	-0.468272	-0.327419

50%	-0.300587	0.034624	-0.189118	0.436883
75%	0.802237	0.732131	0.421296	1.178404
max	1.139433	1.318510	0.857751	1.615822

Фрагментація даних

Розглянемо, як фрагментувати кадр даних pandas.

Можемо використати назву стовпця для збереження даних у певному стовпці:

```
## Фрагмент. Використання імені
df['A']
```

Результат виконання:

```
2030-01-31   -0.168655
2030-02-28    0.689585
2030-03-31    0.767534
2030-04-30    0.557299
2030-05-31   -1.547836
2030-06-30    0.511551
Freq: M, Name: A, dtype: float64
```

Для вибору декількох стовпців треба двічі скористатись дужкою, `[[.., ..]]`. Перша пара дужок означає, що хочемо вибрати стовпці, а друга – вказує, які стовпці хочемо повернути.

```
df[['A', 'B']].
```

Результат виконання:

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

Можемо фрагментувати рядки за допомогою коду, який повертає перші три рядки:

```
### Використання фрагмента для рядка
    df[0:3]
```

Результат виконання:

	A	**B**	**C**	**D**
2030-01-31	-0.168655	0.587590	0.572301	-0.031827
2030-02-28	0.689585	0.998266	1.164690	0.475975
2030-03-31	0.767534	-0.940617	0.227255	-0.341532

Функція `loc` використовується для вибору стовпців за назвами. Як завжди, значення перед комою відносяться до рядків, а після неї – до стовпця. Треба користуватися дужками, щоб вибрати більше одного стовпця.

```
## Багато стовпців
df.loc[:,['A','B']]
```

Результат виконання:

	A	**B**
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

Існує ще один спосіб вибору декількох рядків і стовпців у Pandas. Можна використати `iloc[]`. Цей метод використовує індекс замість назви стовпців. Код нижче повертає той самий кадр даних, що і вище:

```
df.iloc[:, :2]
```

Результат виконання:

	A	**B**
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266

2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

Видалення стовпця

Можемо видаляти стовпці, використовуючи `pd.drop()`:

```
df.drop(columns=['A', 'C'])
```

Результат виконання:

	B	**D**
2030-01-31	0.587590	-0.031827
2030-02-28	0.998266	0.475975
2030-03-31	-0.940617	-0.341532
2030-04-30	0.507350	-0.296035
2030-05-31	1.276558	0.523017
2030-06-30	1.572085	-0.594772

Конкатенція

Можемо об'єднати два DataFrame в Pandas. Для цього можна використати `pd.concat()`.

Перш за все, треба створити два DataFrame. Тому добре, що ми вже знайомі зі створенням кадрів даних

```
import numpy as np
df1 = pd.DataFrame({'name': ['John', 'Smith','Paul'],
                     'Age': ['25', '30', '50']},
                    index=[0, 1, 2])
df2 = pd.DataFrame({'name': ['Adam', 'Smith' ],
                     'Age': ['26', '11']},
                    index=[3, 4])
```

Тепер можемо об’єднати два DataFrame

```
df_concat = pd.concat([df1,df2])
df_concat
```

Результат виконання:

	Age	name
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam
4	11	Smith

Видалення дуплікатів

Якщо набір даних містить дублікати інформації, то можна скористатися `drop_duplicates`, щоб легко виключити рядки, які повторюються. Можемо побачити в прикладі вище, що у `df_concat` є повторюване спостереження: `Smith` відображається двічі у колонці `name`.

```
df_concat.drop_duplicates('name')
```

Результат виконання:

	Age	name
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam

Сортування значень

Значення можна сортувати за допомогою `sort_values`:

```
df_concat.sort_values('Age')
```

Результат виконання:

	Age	name
4	11	Smith
0	25	John
3	26	Adam

1	30	Smith
2	50	Paul

Перейменування: зміна індексу

Для перейменування стовпця в Pandas можна скористатися `rename`. Перше значення – це поточна назва стовпця, а друге – нова його назва.

```
df_concat.rename(columns={"name": "Surname", "Age": "Age_ppl"})
```

Результат виконання:

	Age_ppl	**Surname**
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam
4	11	Smith

Імпорт даних CSV за допомогою `Pandas.read_csv()`

Вивчаючи TF, будемо використовувати набір даних про доросле населення. Його часто використовують із завданням класифікації.

Дані зберігаються у форматі CSV. Цей набір даних включає вісім категорій змінних:

- workclass
- education
- marital
- occupation
- relationship
- race
- sex
- native_country

Крім того, є шість безперервних змінних:

- age
- fnlwgt
- education_num

- capital_gain
- capital_loss
- hours_week

Щоб імпортувати набір даних CSV, можемо використати об’єкт `pd.read_csv()`. Основний аргумент всередині. Синтаксис:

```
pandas.read_csv(filepath_or_buffer,sep=',
',`names=None`,`index_col=None`,`skipinitialspace=False`)
```

• `filepath_or_buffer`: Шлях або URL-адреса даних.

• `sep=', '`: Визначення роздільника, який слід використовувати.

• `` `names = None' ``: Назви стовпців. Якщо в наборі даних є десять стовпців, нам треба передати десять назв.

• `` `index_col = None' ``: Якщо таке визначення, то перший стовпець використовується як індекс рядків.

• `` `skipinitialspace=False' ``: Пропуск пробілів після роздільника.

Для отримання додаткової інформації про `readcsv()`, звертайтеся до офіційної документації.

Нижче зібрані найбільш корисні методи вивчення даних з Pandas.

Імпорт даних	`read_csv`
Створення серій	`Series`
Створення Dataframe	`DataFrame`
Створення діпазону дат	`date_range`
Повернення заголовка	`head`
Повернення хвоста	`tail`
Опис	`describe`
Фрагментація з використанням назви	`dataname['columnname']`

Фрагментація з використанням рядків	`data_name[0:5]`

2.2. Завдання

1. Повторити вправи, наведені в даній лабораторній роботі, зробити скріншоти виконання.

2. Оформити і завантажити протокол на сайт дистанційного навчання.

2.3. Зміст звіту

1. Лістинги програм з коментарями.

2. Відповіді на контрольні питання до роботи.

3. Короткі висновки.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

2.4. Контрольні питання

1. Як працюють методи «повернення заголовка» і «повернення хвоста»?

2. Які параметри має `Series`?

3. Які параметри має `DataFrame`?

Лабораторна робота 3
БІБЛІОТЕКА NUMPY

Мета роботи: Вивчити основи використання бібліотеки NumPy як безкоштовної альтернативи MATLAB.

Зміст. Розглядається створення масивів, базові функції, доступ до елементів, створення спеціальних масивів, математичні операції, записк даних з диску тощо.

3.1. Основи NumPy

Оскільки Python – мова сценаріїв, математичні алгоритми на ній, часто працюють набагато повільніше, ніж у таких компільованих мовах, як C або навіть Java. NumPy намагається вирішити цю проблему для великої кількості обчислювальних алгоритмів забезпечуючи підтримку багатовимірних масивів і безліч функцій і операторів для роботи з ними.

NumPy можна розглядати як хорошу безкоштовну альтернативу MATLAB, оскільки мова програмування MATLAB зовні нагадує NumPy: обидві вони інтерпретовані, і обидві дозволяють користувачам писати швидкі програми, коли більшість операцій проводяться над масивами або матрицями, а не над скалярами. Внутрішньо як MATLAB, так і NumPy базується на бібліотеці LAPACK, призначеної для вирішення основних задач лінійної алгебри.

NumPy найчастіше використовують при аналізі даних та навчанні нейронних мереж – у кожній з цих областей потрібно проводити багато обчислень з масивами. NumPy працює з багатовимірними масивами швидко. NumPy зроблено так, щоб ефективно працювати в Python з наборами чисел будь-якого розміру. Бібліотека частково написана на Python, а частково на C та C++ – у тих місцях, де необхідна швидкість. Крім того, код NumPy оптимізовано під більшість сучасних процесорів.

Для NumPy існують пакети, що розширюють її функціональність, наприклад бібліотека SciPy або Matplotlib. SciPy (Scientific Python) розширює функціонал NumPy величезною колекцією корисних алгоритмів, таких як

мінімізація, перетворення Фур'є, регресія та інші прикладні математичні техніки. Matplotlib – пакет для створення графіки в стилі MATLAB.

Масиви в NumPy відрізняються від звичайних списків та кортежів у Python тим, що вони повинні складатися лише з елементів одного типу. Таке обмеження дозволяє збільшити швидкість обчислень у 50 разів, а також уникнути непотрібних помилок із приведенням та обробкою типів. Перед використанням NumPy потрібно підключити бібліотеку:

```
import numpy as np
```

`np` – це звичне скорочення для NumPy у Python-спільноті. Воно дозволяє швидко звертатися до методів бібліотеки. Хоча можна використовувати NumPy і без надання окремого скорочення або назвати її іншими літерами та символами. Але рекомендуємо дотримуватися традицій спільноти, щоб уникнути непорозуміння з боку інших програмістів.

Створення масивів

Головний об’єкт бібліотеки NumPy – масив. Створюється він просто:

```
a = np.array([1, 2, 3])
```

Ми оголосили змінну `a` та використали вбудовану функцію `array`. В неї треба вкласти сам Python-список, який хочемо створити. Він може бути будь-якої форми: одновимірний, двовимірний, тривимірний тощо. Вище ми створили одновимірний масив. Давайте створимо інші:

```
a1 = np.array([[1, 2, 3], [4, 5, 6]])
a2 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

Вийшла велика кількість дужок. Щоб зрозуміти, як все це виглядає, скористаємося функцією `print`:

```
print(a)
[1 2 3]
```

```
print(a1)
[[1 2 3]
 [4 5 6]]

print(a2)
[[[1 2 3]
  [4 5 6]]

 [[7 8 9]
  [10 11 12]]]
```

У першому випадку вийшов двовимірний масив, що складається з двох одновимірних. А в другому – тривимірний, що складається з двовимірних. Якщо рухатися вгору за вимірами, то вони мають схожу логіку:

• чотиривимірний масив складається з тривимірних;

• п'ятивимірний масив – це кілька чотиривимірних;

• n-вимірний масив – це кілька (n-1)-вимірних.

У NumPy-масиви можна передавати не лише цілі числа, а й дробові:

```
b = np.array([1.4, 2.5, 3.7])

print(b)
[1.4 2.5 3.7]
```

Щоб вказати конкретний тип, це можна зробити за допомогою додаткового параметра `dtype`:

```
c = np.array([1, 2, 3], dtype='float32')

print(c)
[1. 2. 3.]
```

Цілі числа відразу були приведені до числа з плаваючою точкою. Подібні перетворення необхідні, щоб зекономити пам'ять. Наприклад, числа `int32` займають 32 біти, або 4 байти, а `int16` – 16 біт, або 2 байти. Програмісти

використовують параметр `dtype`, коли точно знають, що їх змінні будуть в діапазоні від -32768 до 32767.

Якщо реальні значення елементів масиву вийдуть за рамки явно зазначеного типу, то їх значення в якийсь момент просто обнуляються і почнуть відлік заново:

```
# Допустимо, що a — це змінна типу int16, у якої максимальне значення — 32 767 a = 32 000 b = 768 print(a + b) -32768
```

Як ми бачимо, результат обнулився до свого мінімального значення – – 32 768.

Базові функції

Тепер, розглянемо, які є у масивів вбудовані функції.

Допустимо, що у нас є такий об’єкт:

```
b = np.array([1,2,3], dtype='int32')

print(b)
[1 2 3]
```

Щоб дізнатися, скільки у нього вимірів, скористаємося функцією `ndim`:

```
print(b.ndim)
1
```

Все правильно, адже у нас одновимірний масив, або вектор. Якби він був двовимірним, то й результат виявився б іншим:

```
b = np.array([[1, 2, 3], [4, 5, 6]])

print(b.ndim)
2
```

Для підрахунку кількість рядків і стовпців є функція `shape`:

```
print(a.shape)
(3, )
```

Вийшло трохи незрозуміло, тому пояснимо. Справа в тому, що вектор – це лише одновимірний масив. Вектори в бібліотеці NumPy мають лише рядки або елементи. Тому функція `shape` видала число 3.

З двовимірними масивами ситуація зрозуміліша:

```
print(b.shape)
(2, 3)
```

В `b` – 2 рядки і 3 стовпці.

Для тривимірних і n-вимірних масивів функція `shape` додаватиме додаткові цифри в кортежі через кому:

```
c = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(c.shape)
(2, 2, 3)
```

Читається це так: в об'єкті `c` два тривимірні масиви з двома рядками і трьома стовпцями. Крім розмірності, можна також дізнатися тип елементів – у цьому допоможе функція `dtype` (не плутаємо її з однойменним параметром):

```
a = np.array([1,2,3], dtype='int32')

print(a.dtype)
dtype('int32')
```

Якщо не присвоювати тип елементам вручну, за замовчуванням буде встановлено `int32`.

Ще можна дізнатися кількість елементів за допомогою функції `size`:

```
print(a.size)
3
```

А через функції `itemsize` і `nbytes` можна дізнатися, скільки байт у пам'яті займає один елемент і скільки байт займає весь масив:

```
b = np.array([[1, 2, 3], [4, 5, 6]], dtype='int16')

print(b.itemsize)
```

```
2
```

```
print(b.nbytes)
12
```

Один елемент займає 2 байта, а весь об’єкт `b` із 6 елементів – 2 × 6 = 12 байтів.

Доступ до елементів

У NumPy можна звертатися до окремих елементів, рядків чи стовпців, а також окремо вибирати послідовність потрібних елементів. Розглянемо все докладніше. Допустимо, у нас є двовимірний масив:

```
a = np.array([[1, 2, 3, 4, 5, 6, 7], [8, 9, 10, 11, 12, 13, 14]])

print(a)
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]]
```

І ми хочемо дістати з нього елемент, який знаходиться в першому рядку на п'ятому місці. Зробити це можна за допомогою спеціального оператора `[]`:

```
print(a[0, 4])
5
```

0 і 4 тому, що нумерація елементів у Python починається з нуля, а значить, перший рядок буде нульовим, а п'ятий стовпець – четвертим.

Якби у нас був тривимірний масив, звернення до його елементів було б схожим:

```
c = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(c)
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]

print(c[1, 1, 1])
11
```

Тут ми спочатку звернулися до другого двовимірного масиву, а потім вибрали в ньому другий рядок та другий стовпець. Там і було число 11.

Крім окремих елементів, у бібліотеці NumPy можна звернутися до цілого рядка або стовпця за допомогою оператора «:». Він дозволяє вибрати всі елементи зазначеного рядка або стовпця:

```
print(a[0, :])
[1 2 3 4 5 6 7]

print(a[:, 0])
[1 8]
```

У першому випадку ми вибрали весь перший рядок, а в другому перший стовпець. Також можна використовувати просунуті методи виділення необхідних нам елементів. До речі, оператор «:» насправді є скороченою формою конструкції `початковий_індекс: кінцевий_індекс: крок`.

Давайте зупинимося на ній детальніше:

```
b = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(b[0, 1:4:2])
[2 4]
```

Ми вказали, що хочемо вибрати перший рядок, а потім уточнили, які саме стовпці потрібні нам: 1:4:2.

- Перше число означає, що ми починаємо брати елементи з першого індексу другого стовпця.
- Друге число – що ми закінчуємо ітерацію на четвертому індексі, тобто проходимо весь рядок.
- Третє число вказує, з яким кроком ми йдемо рядком. У нашому прикладі – з кроком у два елементи. Тобто ми пройдемо елементами 1, 3 і 5.

Подивимося на інший приклад:

```
a = np.array([[1, 2, 3, 4, 5, 6, 7], [8, 9, 10, 11, 12, 13, 14]])

print(a)
[[ 1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14]]
```

```
print(a[1, 0:-1:3])
[8, 11]
```

Тут ми використовували негативні індекси – вони дозволяють рахувати індекси елементів справа наліво. -1 означає, що останній індекс – це останній стовпець другого рядка. Зауважимо, що 6-й стовпець не виведений. Це означає, що NumPy не досягає цього індексу, а закінчує обхід на один індекс раніше.

Також можемо змінювати значення в NumPy-масиві за допомогою тієї ж операції доступу до елементів. Наприклад:

```
c = np.array([[0, 2], [4, 6]])
c[0, 0] = 1

print(c)
[[1 2]
 [4 6]]
```

Ми замінили елемент із першого рядка та першого стовпця (елемент 0) на одиницю. І наш масив вдало змінився.

Крім окремих елементів, можна замінювати будь-які послідовності елементів за допомогою конструкції `початковий_індекс: кінцевий_індекс: крок` та її спрощену версію – «`:`».

```
c = np.array([[0, 2], [4, 6]])
c[0, :] = [3, 3]

print(c)
[[3 3]
 [4 6]]
```

Тепер весь перший рядок замінився на трійки. Головне за такої заміни – враховувати розмір рядка, щоб не виникало помилок. Наприклад, якщо присвоїти першому рядку вектор із трьох елементів, інтерпретатор видає помилку, що не можна присвоїти одновимірному масиву з розміром 2 масив розміром 3.

Створення спеціальних масивів

Ми навчилися створювати масиви будь-якої розмірності. Але іноді бажано створити їх із вже заповненими значеннями.

Наприклад, заповнити всі комірки нулями чи одиницями. NumPy може це.

Масив із нулів. Для створення його використовуємо функцію `zeros`:

```
a = np.zeros((2, 2))

print(a)
[[0. 0.]
 [0. 0.]]
```

Перше, що треба пам'ятати, як задавати розмір. Він задається кортежем (2, 2). Якщо вказати розмір без дужок, знову отримаємо помилку.

А все тому, що без дужок NumPy розцінює другий елемент (число 2) як тип даних, який має бути вказаний для параметра `dtype`.

Якщо вказати лише одне число, створиться вектор розміром в два елементи. В цьому випадку вже не треба ставити додаткові дужки:

```
a = np.zeros(2)

print(a)
[0. 0.]
```

Варто відзначити, що елементам за умовчанням надається тип `float64`. Це дробові числа, які займають у пам'яті 64 біти, або 8 байт. Якщо потрібні лише цілі числа, то ми за старою схемою вказуємо це в параметрі `dtype` через кому:

```
a = np.zeros(2, dtype='int32')

print(a)
[0 0]
```

Масив з одиниць. Він створюється так само, як і з нулів, але використовується функція `ones`:

```
b = np.ones((4, 2, 2), dtype='int32')
print(b)
[[[1 1]
  [1 1]]

 [[1 1]
  [1 1]]

 [[1 1]
  [1 1]]

 [[1 1]
  [1 1]]]
```

Тут ми створили тривимірний масив із чотирьох двовимірних, кожен з яких має два рядки та два стовпці. Також вказали, що елементи повинні мати тип `int32`.

Масив з довільних чисел. Іноді буває потрібно заповнити масив якимись відмінними від нуля та одиниці числами. У цьому допоможе функція `full`:

```
c = np.full((2, 2), 5)

print(c)
[[5 5]
 [5 5]]
```

Тут ми спочатку вказали, який розмір має бути у масиву через кортеж, – (2,2), а потім число, яким ми хочемо заповнити всі його елементи – 5. Як і у функцій `zeros` і `ones`, елементи за промовчанням матимуть тип `float64`. А розмір повинен передаватися у вигляді кортежу (2, 2).

Масив випадкових чисел. Він створюється за допомогою функції `random.rand`:

```
d = np.random.rand(3, 2)

print(d)
[[0.76088962 0.14281283]
 [0.32124888 0.34894434]
 [0.66903093 0.72899792]]
```

Вийшло щось незрозуміле. Але так і має бути – адже NumPy генерує випадкові числа в діапазоні від 0 до 1 із вісьмома знаками після коми.

Ще одне цікаве те, як задається розмір. Це потрібно робити не через кортеж (3, 2), а просто вказуючи розмірності через кому. Все тому, що у функції `random.rand` немає параметра `dtype`. Щоб створити масив випадкових чисел, треба скористатися функцією `random.randint`:

```
e = np.random.randint(-5, 10, size=(4, 4))

print(e)
[[ 3  1 -4  3]
 [ 0 -2  5  3]
 [ 5 -1  9  2]
 [ 0 -4  9 -2]]
```

У нас вийшов масив розміром чотири на чотири – `size = (4, 4)` – з цілими числами з діапазону від -5 до 10. Як і в попередньому випадку, створюється такий масив трохи незвично, але таке буває в NumPy.

Одинична матриця. Вона потрібна тим, хто займається лінійною алгеброю. Діагональні елементи такої матриці рівні одиниці, а решта елементів – нулі. Створюється вона за допомогою функції `identity` або `eye`:

```
i = np.identity(4)

print(i)
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Тут задається лише кількість рядків матриці, тому що одинична матриця має бути симетричною – мати однакову кількість рядків і стовпців. Також можна вказати тип елементів за допомогою `dtype`:

```
i = np.identity(4, dtype='int32')

print(i)
[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]
 [0 0 0 1]]
```

Математичні операції

Масиви з NumPy підтримують усі стандартні арифметичні операції – наприклад, додавання, ділення, віднімання. Працює це поелементно:

```
a = np.array([1, 2, 3, 4])

print(a)
[1 2 3 4]

print(a + 3)
[4 5 6 7]
```

До кожного елемента додалося число 3, а розмірність не змінилася. Всі інші операції працюють так само:

```
print(a - 2)
[-1 0 1 2]
```

```
print(a * 2)
[2 4 6 8]

print(a / 2)
[0.5 1. 1.5 2.] #Тут тип елементів приведений до 'float64'

print(a ** 2)
[1 4 9 16]
```

Можна також проводити будь-які математичні операції з масивами однакового розміру:

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[2, 2], [2, 2]])

print(a + b)
[[3 4]
 [5 6]]

print(a * b)
[[2 4]
 [6 8]]

print(a ** b)
[[ 1  4]
 [ 9 16]]
```

Тут кожен елемент `a` додається, множиться і підноситься в ступінь на елемент із такої ж позиції масиву `b`. Крім примітивних операцій, у NumPy можна проводити і складніші – наприклад, обчислити косинус:

```
a = np.array([[1, 2], [3, 4]])

print(np.cos(a))
[[ 0.54030231 -0.41614684]
 [-0.9899925  -0.65364362]]
```

Всі математичні функції викликаються за схожим шаблоном: спочатку пишемо `np.назва_математичної_функції`, а потім передаємо всередину масив – як ми і зробили вище.

Ще до масивів можна застосовувати різні операції з лінійної алгебри, математичної статистики тощо. Давайте, для прикладу перемножимо матриці за правилами лінійної алгебри:

```
a = np.ones((2, 3))
```

```
print(a)
[[1. 1. 1.]
 [1. 1. 1.]]

b = np.full((3, 2), 2)
print(b)
[[2 2]
 [2 2]
 [2 2]]

 print(np.matmul(a, b))
[[6. 6.]
 [6. 6.]]
```

Повний список всіх операцій, що підтримуються в бібліотеці, можна знайти в офіційній документації.

Копіювання та організація

Якщо в NumPy надамо масив іншій змінній, то просто створимо посилання на нього. Розберемо на прикладі:

```
a = np.array([1, 2, 3])
b = a
b[0] = 5

print(b)
[5 2 3]

print(a)
[5 2 3]
```

Як бачимо, при зміні `b` змінюється також і `a`. Справа в тому, що масиви в NumPy – це лише посилання на області в пам'яті. Тому, коли ми присвоїли `a` змінній `b`, насправді ми просто надали їй посилання на перший елемент `a` в пам'яті.

Щоб створити незалежну копію `a`, користуємося функцією `copy`:

```
a = np.array([1, 2, 3])
b = a.copy()
b[0] = 5

print(b)
[5 2 3]

print(a)
[1 2 3]
```

Також можемо змінювати розмір масиву за допомогою функції `reshape`:

```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(a)
[[1 2 3 4]
 [5 6 7 8]]

b = a.reshape((4, 2))

print(b)
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

Спочатку розмір `a` був 2 на 4. Ми переробили його під 4 на 2. Звернемо увагу: нові розміри повинні відповідати кількості елементів. В іншому випадку Python поверне помилку.

Також NumPy дозволяє нам "нашаровувати" масиви один на одного і з'єднувати їх за допомогою функцій `vstack` і `hstack`:

```
v1 = np.array([1, 2, 3])
v2 = np.array([9, 8, 7])
v3 = np.vstack([v1, v2])

print(v3)
[[1 2 3]
 [9 8 7]]
```

Тут ми створили двовимірний масив із двох векторів однакового розміру, які "поставили" один на одного. Те ж саме можна зробити і горизонтально:

```
h1 = np.ones((2, 4))
h2 = np.zeros((2, 2))
h3 = np.hstack((h1, h2))

print(h3)
[[1. 1. 1. 1. 0. 0.]
 [1. 1. 1. 1. 0. 0.]]
```

До масиву з одиниць праворуч приєднався масив із нулів. Головне, щоб кількість рядків в обох була однаковою, інакше буде помилка.

Додаткові можливості

NumPy підтримує інші корисні функції, які використовуються рідше, але знати про які корисно. Наприклад, одна з таких функцій – читання файлів із жорсткого диска.

Читання даних із файлу. Допустимо, у нас є файл `data.txt` з таким вмістом:

```
1,2,3,5,6,7
8,1,4,2,6,4
9,0,1,7,3,4
```

Можемо записати числа в NumPy-масив за допомого функції `genfromtxt`:

```
filedata = np.genfromtxt('data.txt', delimiter=',')
filedata = filedata.astype('int32')

print(filedata)
[[1  2  3  5  6  7]
[8  1  4  2  6  4]
[9  0  1  7  3  4]]
```

Спочатку ми дістали дані із файлу `data.txt` через функцію `genfromtxt`. У ній потрібно вказати файл, що зчитується, а потім роздільник – щоб NumPy розумів, де починаються і закінчуються числа. У нас роздільником буде ,.

Потім нам треба привести числа до формату `int32` за допомогою функції `astype`, передавши в неї потрібний тип.

Булеві вирази. Ще одна з можливостей NumPy – булеві вирази для елементів масиву. Вони дозволяють дізнатися, які елементи відповідають певним умовам – наприклад, чи більше кожне число масиву 50.

Припустимо, у нас є масив `a` і ми хочемо перевірити, чи дійсно всі його елементи більші за 5:

```
a = np.array([[1, 5, 8], [3, 4, 2]])

print(a > 5)
[[False False  True]
 [False False False]]
```

На виході – масив з «відповіддю» для кожного елемента: чи він більший за число 5. Якщо менший чи рівний, то стоїть `False`, інакше – `True`. За допомогою булевих виразів можна складати і складніші конструкції – наприклад, створювати нові масиви з елементів, які відповідають певним умовам:

```
a = np.array([[1, 5, 8], [3, 4, 2], [2, 6, 7]])

print(a[a > 3])
[5 8 4 6 7]
```

Ми отримали вектор, що складається з елементів масиву `a`, які більше трьох.

Зробимо деякі висновки:

• NumPy – це бібліотека для ефективної роботи з масивами будь-якого розміру. Вона досягає високої продуктивності, тому що написана частково на C і C++ і в ній дотримується принцип локальності – вона зберігає всі елементи послідовно в одному місці.

• Перед використанням NumPy в сценарії Python його потрібно підключити за допомогою команди `import numpy as np`.

• Основа NumPy – масив. Щоб його створити, потрібно використовувати функцію `array` і передати туди список як перший аргумент. Другим аргументом через `dtype` можна вказати тип всіх елементів – наприклад, `int16` чи `float32`. За замовчуванням для цілих чисел обирається `int32`, а десяткових – `float64`.

• Функція `ndim` дозволяє дізнатися скільки вимірювань у масиву; `shape` – його структуру (скільки стовпців та рядків); `dtype` – який тип у елементів; `size` – кількість елементів; `itemsize` – скільки байтів займає один елемент; `nbytes` – скільки всього пам'яті займає масив.

• До елементів масиву можна звертатись за допомогою оператора `[]`, в якому вказуються індекси потрібного елемента. Важливо пам'ятати, що індексація починається з нуля. Також в NumPy можна вибирати одразу цілі рядки або стовпці за допомогою оператора `:` і його просунутої версії – `початковий_індекс: кінцевий_індекс: крок`.

• Функції `zeros`, `ones`, `full`, `random.rand`, `random.randint`, `identity` та `eye` допомагають швидко створити масиви будь-якого розміру із заповненими елементами.

• Усі арифметичні операції, доступні в Python, застосовні і до масивів NumPy. Головне – пам'ятати, що операції проводяться елемент за елементом. А для складних операцій, таких як обчислення похідної, також є свої функції.

• NumPy-масиви не можна просто присвоїти іншій змінній, щоб скопіювати. Для цього існує функція `copy`. Щоб змінити структуру даних, можна застосувати функції `reshape`, `vstack` і `hstack`.

• В NumPy є додаткові функції – наприклад, читання з файлу за допомогою `genfromtxt` та булеві вирази, які дозволяють вибирати елементи із набору даних за заданими умовами.

3.2. Завдання

1. Повторити наведені в ЛР приклади та пояснити отримані результати.

3.3. Зміст звіту

1. Скріншоти виконання завдання.

2. Коментарі до виконаних прикладів.

2. Відповіді на контрольні питання до роботи.

3. Короткі висновки.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

3.4. Контрольні питання

1. Що є основою NumPy?

2. Які булеві вирази є в NumPy?

3. Чому NumPy має високу швидкість роботи з масивами?

4. Які бібліотеки ви знаєте, що розширюють функціональність NumPy? Для чого вони призначені?

Лабораторна робота 4

РЕАЛІЗАЦІЯ ЛІНІЙНОЇ РЕГРЕСІЇ НА PYTHON

Мета роботи: Навчитися будувати алгоритми для реалізації лінійної регресії з допомогою сценаріїв Python.

Зміст. Розглядаються моделі лінійних нейронів з одним входом та кількома, враховуються помилки експерименту через використання середньоквадратичної помилки, виконується навчання моделі лінійного нейрона з одним входом за допомогою алгоритму градієнтного спуску. Запропоновані алгоритми реалізовані у вигляді сценаріїв Python.

4.1. Теоретичні відомості

Можемо абстрагувати модель нейрона в математичну структуру, як показано на рис.4.1(a). Вхідний вектор нейрона $x = [x_1, x_2, x_3, \ldots, x_n]^T$ відображається на y через функцію $f_\theta: x \rightarrow y$, де θ представляє параметри функції f. Розглянемо спрощений випадок, коли маємо лінійне перетворення: $f(x) = w^T x + b$. Розгорнута форма:

$$f(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + \ldots + w_n x_n + b$$

Попередню логіку обчислень інтуїтивно показано на рис.4.1 (b).

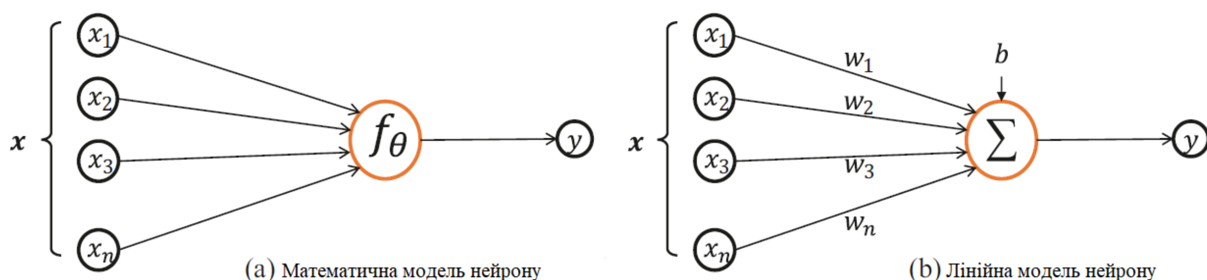

Рисунок 4.1 – Математична модель нейрону

Параметри $\theta = \{w_1, w_2, w_3, \ldots, w_n, b\}$ визначають стан нейрона, а логіку обробки цього нейрона можна визначити шляхом фіксації цих параметрів. Коли

кількість вхідних вузлів $n = 1$ (один вхід), модель нейрона може бути додатково спрощена як

$$y = wx + b$$

Тоді можемо побудувати y як функцію змінної x, як показано на рис.4.2. Із збільшенням вхідного сигналу x вихідний сигнал y також лінійно зростає. Тут параметр w визначає нахил прямої, а b – її зміщення.

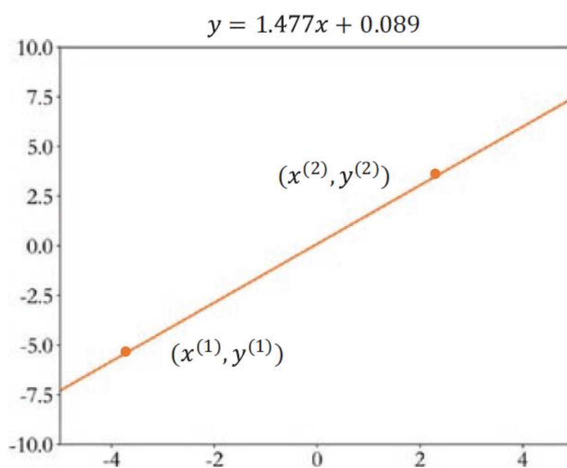

Рисунок 4.2 – Лінійна модель нейрону з одним входом

Для певного нейрона співвідношення $f_{w,b}$ між x і y є невідомим, але фіксованим. Дві точки можуть визначити пряму. Щоб оцінити значення w і b, нам треба лише відібрати будь-які дві точки даних $(x^{(1)}, y^{(1)})$ і $(x^{(2)}, y^{(2)})$ з прямої лінії на рис.4.2, де верхній індекс вказує номер точки даних:

$$y^{(1)} = wx^{(1)} + b$$

$$y^{(2)} = wx^{(2)} + b$$

Якщо $(x^{(1)}, y^{(1)}) \neq (x^{(2)}, y^{(2)})$, ми можемо розв’язати попередні рівняння, щоб отримати значення w і b. Розглянемо конкретний приклад: $x^{(1)} = 1$, $y^{(1)} = 1.567$, $x^{(2)} = 2$, $y^{(2)} = 3.043$. Підставляючи числа в попередні формули отримаємо:

$$1.567 = w \cdot 1 + b$$

$$3.043 = w \cdot 2 + b$$

Це система двох лінійних рівнянь, яку вивчають у школі. Аналітичне рішення можна легко розрахувати методом елімінації, отримаємо $w = 1.477$, $b = 0.089$.

Бачимо, що нам треба лише дві різні точки даних, щоб ідеально визначити параметри моделі лінійного нейрона з одним входом. Для моделей лінійних нейронів із N вхідними даними нам треба взяти лише N + 1 різних точок даних. Здається, лінійні моделі нейронів можна ідеально розв'язати. Отже, що не так з попереднім методом? Враховуючи, що можуть бути помилки спостереження для будь-якої точки відбору, припускаємо, що змінна помилки спостереження відповідає нормальному розподілу $N(\mu,\sigma^2)$ з μ як середнім і σ^2 як дисперсією. Далі наступні зразки:

$$y = wx + b + \varepsilon\ , \varepsilon \sim N(\mu,\sigma^2)$$

Після появи помилки спостереження, навіть якщо вона така ж проста, як лінійна модель, якщо взяти лише дві точки даних, то це може призвести до значного зміщення оцінки. Як показано на рис.4.3, усі точки даних мають помилки спостереження. Якщо оцінка базується на двох синіх прямокутних точках даних, оцінена синя пунктирна лінія матиме велике відхилення від справжньої помаранчевої прямої лінії. Щоб зменшити зміщення оцінки, викликане помилками спостереження, можемо взяти вибірку з кількох точок даних

$$\mathrm{D} = \{(x^{(1)}, y^{(1)}),(x^{(2)}, y^{(2)}),\dots, (x^{(n)}, y^{(n)})\}$$

а потім знайти «найкращу» пряму так, щоб вона мінімізувала суму помилок між усіма точками відбору проб і прямою лінією.

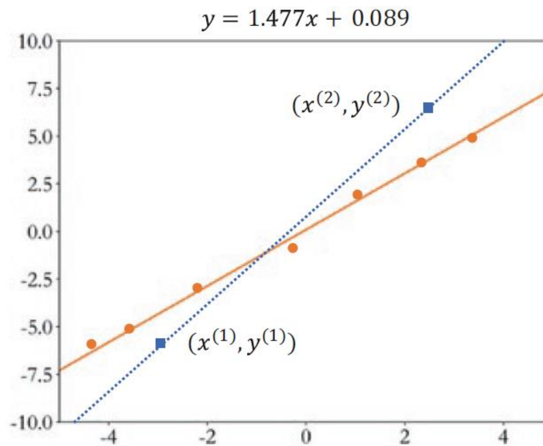

Рисунок 4.3 – Модель з помилками спостереження

Через наявність помилок спостереження пряма може не бути лінією, яка ідеально проходить через усі точки відбору проб D. Тому ми сподіваємося знайти «хорошу» пряму лінію поблизу всіх точок відбору. Як виміряти це «добре» і «погано»? Природною ідеєю є використання середньоквадратичної помилки (MSE) між прогнозованим значенням $wx^{(i)} + b$ та істинним значенням $y^{(i)}$ у всіх точках вибірки як загальну помилку, тобто

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\left(wx^{(i)} + b - y^{(i)}\right)^2$$

Далі шукаємо набір параметрів w* та b*, щоб мінімізувати загальну помилку $\mathcal{L}$. Пряма, що відповідає мінімальній сумарній похибці, є оптимальною прямою, яку ми шукаємо, тобто

$$w^*, b^* = \underset{w,b}{\arg\min} \frac{1}{n}\sum_{i=1}^{n}\left(wx^{(i)} + b - y^{(i)}\right)^2$$

Тут n означає кількість точок відбору проб.

Метод оптимізації

Тепер давайте зробимо висновки з попереднього розгляду: нам потрібно знайти оптимальні параметри w^* і b^*, щоб вхід і вихід відповідали лінійній залежності $y^{(i)} = wx^{(i)} + b$, i ∈ [1, n]. Однак, через наявність помилок спостереження, необхідно взяти вибірку з набору даних D = $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$, що складається з достатньої кількості вибірок даних, щоб

знайти оптимальний набір параметрів w^* та b^* для мінімізації середньої квадратичної помилки

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\left(wx^{(i)} + b - y^{(i)}\right)^2$$

Для моделі нейрона з одним входом потрібні лише дві вибірки, щоб отримати точне рішення рівнянь методом елімінації. Це точне рішення, отримане за строгою формулою, називається аналітичним рішенням. Однак, у випадку кількох точок даних ($n \gg 2$), ймовірно, немає аналітичного рішення. Ми можемо використовувати лише чисельні методи оптимізації для отримання наближеного чисельного рішення.

Чому це називається оптимізацією? Це тому, що швидкість обчислення комп'ютера дуже висока. Ми можемо використовувати потужну обчислювальну потужність для «пошуків» та «проб» кілька разів, тим самим зменшуючи помилку $\mathcal{L}$ крок за кроком.

Найпростішим методом оптимізації є пошук грубою силою або випадковий експеримент. Наприклад, щоб знайти найбільш відповідні w^* та b^*, ми можемо випадково вибрати будь-які w та b із простору реальних чисел та обчислити значення помилки $\mathcal{L}$ відповідної моделі. Виберіть найменшу похибку Λ∗ з усіх експериментів _Λ_, а відповідні їй w^* та b^* будуть оптимальними параметрами, які ми шукаємо. Цей алгоритм грубої сили простий і зрозумілий, але він надзвичайно неефективний для великомасштабних задач оптимізації великої розмірності.

Градієнтний спуск є найбільш часто використовуваною оптимізацією алгоритм навчання нейронної мережі. Завдяки можливостям паралельного прискорення потужних GPU він дуже підходить для оптимізації моделей нейронної мережі з масивними даними. Звичайно, він також підходить для оптимізації нашої простої лінійної моделі нейрона. Оскільки алгоритм градієнтного спуску є основним алгоритмом глибокого навчання, ми спочатку застосуємо алгоритм градієнтного спуску для вирішення простих моделей нейронів.

За допомогою концепції похідної, якщо хочемо знайти максимальне та мінімальне значення функції, можемо просто знайти функцію похідної рівною 0 і отримати відповідні значення незалежної змінної, тобто точку стагнації, а потім перевірити тип стагнації. Взявши, як приклад, функцію $f(x) = x^2 \cdot sin(x)$, можемо побудувати графік функції та її похідної в інтервалі $x \in [-10, 10]$, де синя суцільна лінія – $f(x)$, а жовта пунктирна лінія $\frac{df(x)}{dx}$, як показано на рис.4.4. Можна побачити, що точки, де похідна (пунктирна лінія) дорівнює 0, є точками стагнації, і як максимальне, так і мінімальне значення $f(x)$ з'являються в точках стагнації.

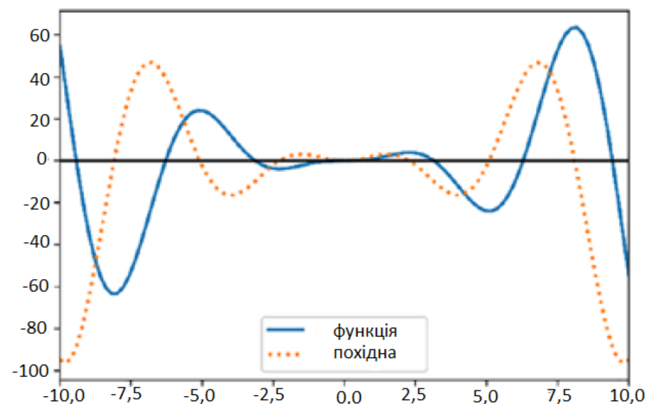

Рисунок 4.4 – Функція $f(x) = x^2 \cdot sin(x)$ та її похідна

Градієнт функції визначається як вектор частинних похідних функції за кожною незалежною змінною. Розглядаючи тривимірну функцію $z = f(x, y)$, часткова похідна функції за незалежною змінною x дорівнює $\frac{\partial z}{\partial x}$, часткова похідна функції за незалежною змінною y записується як градієнт ∇f і є вектором $\left(\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}\right)$.

Давайте розглянемо конкретну функцію $f(x,y) = -(\cos^2 x + \cos^2 y)^2$. Як показано на рис.4.5, довжина червоної стрілки в площині представляє модуль вектору градієнта, а напрямок стрілки – напрямок вектору градієнта. Видно, що напрямок стрілки завжди вказує на напрямок збільшення значення функції. Чим крутіша функціональна поверхня, тим більша довжина стрілки, і більший модуль градієнта.

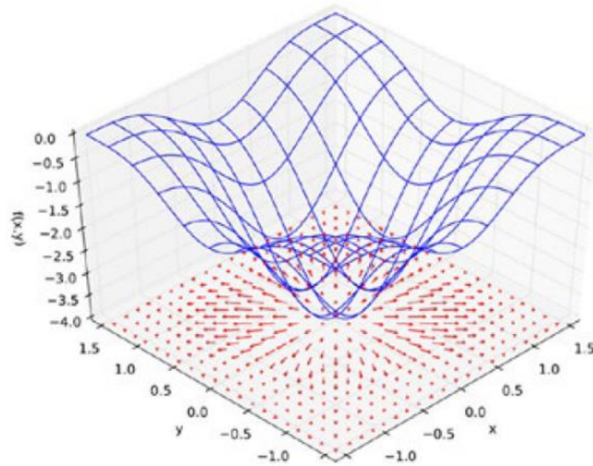

Рисунок 4.5 – Функція та її градієнт

Завдяки попередньому прикладу можемо інтуїтивно відчути, що напрямок градієнта функції завжди вказує на напрямок, у якому збільшується значення функції. Тоді протилежний напрямок градієнта має вказувати на напрямок, у якому значення функції зменшується.

$$\boldsymbol{x}' = \boldsymbol{x} - \eta \cdot \nabla f$$

Щоб скористатися цією властивістю, нам просто потрібно слідувати попередньому рівнянню, щоб ітеративно оновлювати x'. Тоді зможемо отримувати все менші значення функції. η використовується для масштабування вектору градієнта, який відомий як швидкість навчання та зазвичай має менше значення, наприклад 0,01 або 0,001. Зокрема, для одновимірних функцій попередню векторну форму можна записати у скалярній формі:

$$x' = x - \eta \cdot \frac{\mathrm{d}y}{\mathrm{d}x}$$

Повторюючи та оновлюючи x' кілька разів у попередній формулі, значення функції y' в точці x' завжди більш ймовірно буде меншим, ніж значення функції в точці x. Метод оптимізації параметрів за наведеною формулою називається алгоритмом градієнтного спуску. Він обчислює градієнт ∇f функції f та ітеративно оновлює параметри θ для отримання оптимального чисельного значення параметрів θ, коли функція f досягає свого мінімального значення. Слід зазначити, що вхідні дані моделі в глибокому навчанні зазвичай представлені як x, а параметри, які потрібно оптимізувати, зазвичай представлені як θ, w і b.

Тепер ми застосуємо алгоритм градієнтного спуску для розрахунку оптимальних параметрів w* і b*, наведених на початку цього розділу. Тут функція середньо квадратичної помилки мінімізована:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\left(wx^{(i)} + b - y^{(i)}\right)^2$$

Параметрами моделі, які потрібно оптимізувати, є w і b, тому оновлюємо їх ітеративно, використовуючи такі рівняння:

$$w' = w - \eta\frac{\partial\mathcal{L}}{\partial w}$$

$$b' = b - \eta\frac{\partial\mathcal{L}}{\partial b}$$

4.2. Виконання роботи

Нам треба фактично навчити модель лінійного нейрона з одним входом за допомогою алгоритму градієнтного спуску. По-перше, нам потрібно взяти вибірку з кількох точок даних. Для приклада, безпосередньо вибираємо зазначену реальну модель:

$$y = 1.477x + 0.089$$

Дані вибірки

Щоб змоделювати помилки спостереження, додаємо до моделі незалежну змінну помилки ϵ, де ϵ відповідає розподілу Гауса із середнім значенням 0 і стандартним відхиленням 0.01 (тобто дисперсія 0.01^2):

$$y = 1.477x + 0.089 + \varepsilon,\ \varepsilon \sim N(0, 0.01^2)$$

Шляхом випадкової вибірки $n = 100$ разів отримуємо навчальний набір даних D^{train}, використовуючи такий код:

```
import numpy as np
data = [] # Список для збереження зразків даних
for i in range(100): # повторити 100 разів
    # Вибірка x випадковим чином із рівномірного розподілу
    x = np.random.uniform(-10., 10.)
    # Випадкова вибірка з розподілу Гауса
    eps = np.random.normal(0., 0.01)
```

```
    # Розрахунок виходу моделі з випадковими помилками
    y = 1.477 * x + 0.089 + eps
    data.append([x, y]) # зберегти до списку даних
data = np.array(data) # перетворити на двовимірний масив Numpy
```

У попередньому коді ми виконали 100 вибірок у циклі, і кожного разу випадково відбирали одну точку даних x із рівномірного розподілу $U(-10, 10)$, а потім випадково відбирали шум ϵ з розподілу Гауса $\mathcal{N}(0,0.01^2)$. Нарешті, ми згенерували дані за допомогою правильної моделі та випадкового шуму ϵ і зберегли їх як масив Numpy.

Обчислення середньої квадратичної похибки

Тепер давайте обчислимо середню квадратичну помилку на навчальному наборі шляхом усереднення квадрата різниці між прогнозованим значенням і справжнім значенням у кожній точці даних. Можемо досягти цього за допомогою такої функції:

```
def mse(b, w, points):
    # Розрахувати MSE на основі поточних w і b
    TotalError = 0
    # Прогнати в циклі всі точки
    for i in range(0, len(points)):
        x = points[i, 0] # Отримати i-тий вхід
        y = points[i, 1] # Отримати i-тий вихід
        # Обчислюємо загальну квадратичну помилку
        totalError += (y - (w * x + b)) ** 2
    # Обчислюємо середнє значення повної квадратичної помилки
    return totalError / float(len(points))
```

Обчислення градієнта

Згідно з алгоритмом градієнтного спуску, нам потрібно обчислити градієнт у кожній точці даних $\left(\frac{\partial \mathcal{L}}{\partial w}, \frac{\partial \mathcal{L}}{\partial b}\right)$.

По-перше, розглянемо розширення функції середньоквадратичної помилки $\frac{\partial \mathcal{L}}{\partial w}$:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \frac{1}{n} \sum_{i=1}^{n} \left(wx^{(i)} + b - y^{(i)} \right)^2}{\partial w} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial \left(wx^{(i)} + b - y^{(i)} \right)^2}{\partial w}$$

Оскільки

$$\frac{\partial g^2}{\partial w} = 2 \cdot g \cdot \frac{\partial g}{\partial w}$$

ми маємо

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{1}{n} \sum_{i=1}^{n} 2 \left(wx^{(i)} + b - y^{(i)} \right) \cdot \frac{\partial \left(wx^{(i)} + b - y^{(i)} \right)}{\partial w}$$

$$= \frac{1}{n} \sum_{i=1}^{n} 2 \left(wx^{(i)} + b - y^{(i)} \right) \cdot x^{(i)}$$

$$= \frac{2}{n} \sum_{i=1}^{n} \left(wx^{(i)} + b - y^{(i)} \right) \cdot x^{(i)}$$

Якщо важко зрозуміти попередній висновок, можемо переглянути відповідний розділ з математики про похідну. Наразі можемо запам’ятати остаточний вираз $\frac{\partial \mathcal{L}}{\partial w}$. Таким же чином отримаємо вираз часткової похідної $\frac{\partial \mathcal{L}}{\partial b}$:

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \frac{1}{n} \sum_{i=1}^{n} \left(wx^{(i)} + b - y^{(i)} \right)^2}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial \left(wx^{(i)} + b - y^{(i)} \right)^2}{\partial b}$$

$$= \frac{1}{n} \sum_{i=1}^{n} 2 \left(wx^{(i)} + b - y^{(i)} \right) \cdot \frac{\partial \left(wx^{(i)} + b - y^{(i)} \right)}{\partial b}$$

$$= \frac{1}{n} \sum_{i=1}^{n} 2 \left(wx^{(i)} + b - y^{(i)} \right) \cdot 1$$

$$= \frac{2}{n} \sum_{i=1}^{n} \left(wx^{(i)} + b - y^{(i)} \right)$$

Відповідно до наведених вище виразів нам треба лише обчислити середнє значення $(wx^{(i)} + b - y^{(i)}) \cdot x^{(i)}$ і $(wx^{(i)} + b - y^{(i)})$ у кожній точці даних.

Реалізація на Python наступна:

```
def step_gradient(b_current, w_current, points, lr):
    # Розрахувати градієнт і оновити w і b
    b_gradient = 0
    w_gradient = 0
    M = float(len(points)) # загальна кількість зразків
    for i in range(0, len(points)):
        x = points[i, 0]
        y = points[i, 1]
        # dL/db:grad_b = 2(wx+b-y) з рівняння вище
        b_gradient += (2/M) * ((w_current * x + b_current) - y)
        # dL/dw:grad_w = 2(wx+b-y)*x з рівняння вище
        w_gradient += (2/M) * x * ((w_current * x + b_current) -
y)
    # Оновити w',b' відповідно до алгоритму градієнтного спуску
    # lr — швидкість навчання
    new_b = b_current - (lr * b_gradient)
    new_w = w_current - (lr * w_gradient)
    return [new_b, new_w]
```

Оновлення градієнта

Після обчислення градієнта функції помилки для w і b можемо оновити значення w і b відповідно до рівняння. Одноразове навчання всіх зразків набору даних називається однією епохою. Можемо повторити кілька епох, використовуючи попередньо визначені функції. Реалізація наступна:

```
def gradient_descent(points, starting_b, starting_w, lr,
num_iterations):
    # Оновити w, b кілька разів
    b = starting_b # початкове значення для b
    w = starting_w # початкове значення для w
    # Час повторення num_iterations
    for step in range(num_iterations):
        # Оновити w, b один раз
        b, w = step_gradient(b, w, np.array(points), lr)
```

```
        # Розрахувати поточні втрати
        loss = mse(b, w, points)
        if step%50 == 0:# виведення втрат та w, b
            print(f"iterations:{step}, loss:{loss}, w:{w}, b:{b}")
        return [b, w] # повернути кінцеве значення w і b
```

Основна функція навчання визначається наступним чином:

```
def main ():
    # Завантажити навчальний набір даних
    data = []
    for i in range(100):
        x = np.random.uniform(3., 12.)
        # середнє = 0, стандартне = 0.1
        eps = np.random.normal(0., 0.1)
        y = 1.477 * x + 0.089 + eps
        data.append([x, y])
    data = np.array(data)
    lr = 0.01 # швидкість навчання
    initial_b = 0 # ініціалізувати b
    initial_w = 0 # ініціалізувати w
    num_iterations = 1000
    # Тренуватися 1000 разів і повернути оптимальні w*,b* та
відповідні втрати
    [b, w]= gradient_descent(data, initial_b, initial_w, lr,
num_iterations)
    loss = mse(b, w, data) # Обчислити MSE
    print(f'Final loss:{loss}, w:{w}, b:{b}')
```

Після 1000 ітераційних оновлень отримані остаточні «оптимальні» рішення для *w* і *b*, які шукаємо.

Результат виконання:

```
iteration:0, loss:11.437586448749, w:0.88955725981925,
b:0.02661765516748428
iteration:50, loss:0.111323083882350, w:1.48132089048970,
b:0.58389075913875
```

```
iteration:100, loss:0.02436449474995, w:1.479296279074,
b:0.78524532356388

iteration:950, loss:0.01097700897880, w:1.478131231919,
b:0.901113267769968
Final loss:0.010977008978805611, w:1.4781312318924746,
b:0.901113270434582
```

Бачимо, що на 100-й ітерації значення w і b вже близькі до реальних значень моделі. w і b, отримані після 1000 оновлень, дуже близькі до реальної моделі. Середньоквадратична помилка процесу навчання показана на рис.4.6.

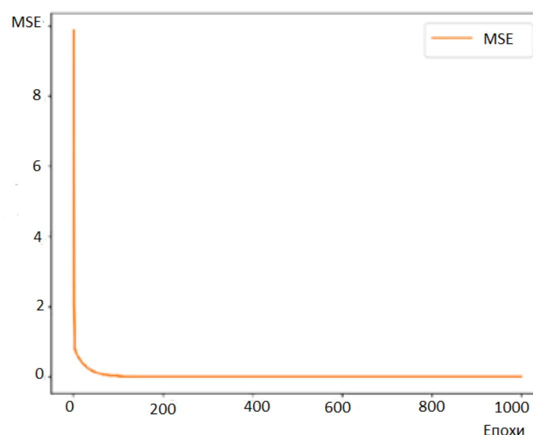

Рисунок 4.6 – Зміна MSE в процесі навчання

Приклад демонструє потужність алгоритму градієнтного спуску для визначення параметрів моделі. Варто зазначити, що для складних нелінійних моделей параметри, які розв'язуються алгоритмом градієнтного спуску, можуть бути локальним мінімальним рішенням замість глобального мінімального рішення, яке визначається невипуклістю функції. Проте на практиці ми виявили, що продуктивність чисельного рішення, отриманого за допомогою алгоритму градієнтного спуску, часто можна добре оптимізувати, а відповідне рішення можна безпосередньо використовувати для наближеного оптимального рішення.

4.3. Завдання

1. Дослідити модель лінійного нейрона з одним входом за допомогою алгоритму градієнтного спуску $y = ax + b + eps$ для виборок (табл.4.1).

2. Визначити мінімальне число ітерацій, коли помилка вже не зменшується.

3. Оформити звіт.

Таблиця 4.1 – Дані варіантів завдання

Варіант	***a***	***b***	***eps***	***n***
1	1.2	0.05	0 - 0.005	100
2	1.25	0.06	0 - 0.01	110
3	1.3	0.07	0 - 0.015	120
4	1.35	0.08	0 - 0.02	130
5	1.4	0.09	0 - 0.005	140
6	1.45	0.05	0 - 0.01	150
7	1.5	0.06	0 - 0.015	100
8	1.55	0.07	0 - 0.02	110
9	1.6	0.08	0 - 0.005	120
10	1.65	0.09	0 - 0.01	130
11	1.7	0.05	0 - 0.015	140
12	1.75	0.06	0 - 0.02	150
13	1.8	0.07	0 - 0.005	100
14	1.85	0.08	0 - 0.01	110
15	1.9	0.09	0 - 0.015	120
16	1.95	0.05	0 - 0.02	130
17	2.0	0.06	0 - 0.005	140
18	2.05	0.07	0 - 0.01	150
19	2.1	0.08	0 - 0.015	100
20	2.15	0.09	0 - 0.02	110

4.4. Зміст звіту

1. Скріншоти виконання завдання.

2. Відповіді на контрольні питання до роботи.

3. Короткі висновки.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

4.5. Контрольні питання

1. Які методи оптимізації, крім методу градієнтного спуску ви можете запропонувати?

2. Які переваги та недоліки методу градієнтного спуску?

3. Які методи, крім MSE, використовуються для обчислення помилки?

Лабораторна робота 5
ПОБУДОВА ШТУЧНОЇ НЕЙРОННОЇ МЕРЕЖІ З TENSORFLOW

Мета роботи: Навчитися використовувати фреймворк TensorFlow для навчання та оцінки штучної нейронної мережі.

Зміст. Розглядаються прості приклади використання TF, аналізується набір реальних даних та вирішується задача розпізнавання об’єктів за допомогою TF.

5.1. Загальні відомості

Ми можемо використовувати TF, щоб робити числові обчислення. Взагалі, це не здається специфічним, але ці обчислення проводяться за допомогою data-flow графів. У цих графах вершини представляють собою математичні операції, а ребра є даними, які зазвичай подаються у вигляді багатовимірних масивів або тензорів, які сполучаються між цими ребрами.

Назва TensorFlow походить від обчислень, які нейромережа робить з багатовимірними даними і тензорами. Буквально – потік тензорів. На даний момент це все, що потрібно знати про тензори, але ми повернемося до них трохи пізніше.

За допомогою TF ми познайомимось з глибоким навчанням в інтерактивній формі:

- Спочатку дізнаємося більше про тензори.
- Потім розглянемо основи TF: як зробити свої перші найпростіші обчислення.
- Наступний етап – справжнє завдання на реальних даних (Бельгійських дорожніх знаках) з розпізнаванням зображень.
- Навчимося розмічати дані таким чином, щоб "згодувати" їх нейромережі.
- Нарешті, розробимо свою модель нейронної мережі – шар за шаром.
- Як тільки архітектура буде готова, зможемо інтерактивно тренувати мережу, а також оцінити ефективність, використовуючи тестову вибірку.

• Останнє – сформуємо вказівки, як можна поліпшити свою модель і як можна далі працювати з TF.

Введення в тензори

Щоб добре зрозуміти тензори, слід мати деякі знання з лінійної алгебри та вміти робити обчислення з векторами. Ми вже знаємо, що тензори реалізовані в TF як багатовимірні масиви даних, але давайте згадаємо, що таке тензори і яка їх роль в машинному навчанні.

Плоскі вектори

Вектор – це особливий вид матриці, прямокутний масив з числами. Так як вектори – це упорядкований набір чисел, то вони часто представляються у вигляді стовпців матриць. Іншими словами, вектор – скалярна величина, якій задали напрямок.

Приклад скаляра – "5 метрів" або "60 м/с", тоді як вектор – "5 метрів на північ" або "60 м/с на схід".

Різниця між ними очевидна: вектор має напрямок (рис.5.1). Проте, приклади можуть бути дуже далекі від тих векторів, з якими ми зіткнемося, займаючись машинними навчанням. Довжина математичного вектору – величина абсолютна, в той час як напрямок – відносна. Довжина вимірюється щодо напрямку, а в якості одиниць виступають градуси або радіани. Зазвичай вважається, що напрямок позитивний і відраховується проти годинникової стрілки щодо початкової точки відліку.

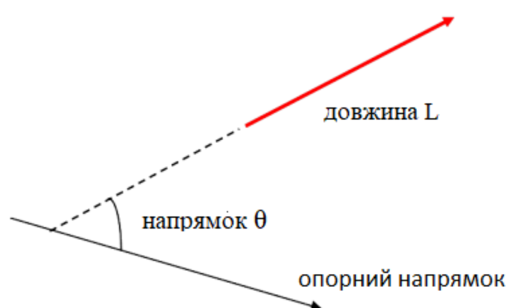

Рисунок 5.1 – Вектор

Візуально, звичайно, вектори є стрілками, як на картинці вище. Це означає, що ми можемо розглядати вектори, як стрілки певної довжини і напрямку.

Плоскі вектори – це найпростіші тензори. Вони дуже схожі на звичайні вектори, такі які бачили вище, з тією невеликою відмінністю, що вони можуть самі визначати себе в векторному просторі.

Щоб зрозуміти, що це означає, розглянемо приклад: нехай є вектор 2 X 1. Це означає, що вектор належить множині дійсних чисел, які об'єднані в пари. Або, іншими словами, елемент двовимірного простору. У таких випадках можемо задавати вектор на координатній площині, як стрілки або промені.

Працюючи на координатній площині, можемо дізнатися **x** координату кінця променю з початком в (0, 0), подивившись на перший рядок вектору, а **y** координату – на другий.

Примітка. Якщо ми розглядаємо вектор розміру 3 X 1, то ми працюємо в тривимірному просторі. Тоді можемо уявити вектор як стрілку в тривимірному просторі, яка зазвичай задається трьома осями x, y і z.

Чудово мати дані вектора і уявлення їх на координатній площині, але, власне, важливо лише те, які операції можемо робити над ними. У цьому нам допоможе вираз даних векторів через базисні або одиничні вектори.

Одиничні вектори – вектори довжини 1. Двовимірні або тривимірні вектори добре розкладаються в суму ортогональних одиничних векторів, таких як осі координат.

Тензори

Плоский вектор – це окремий випадок тензора. Пам'ятаємо, що вектор визначався раніше як скаляр, якому надали напрямок. Тензор же – це математичне уявлення фізичної сутності, яка може бути задана величиною і декількома напрямками. І так само, як ми представляли скаляр одним числом, а тривимірний вектор як трійку чисел, тензор представляється у вигляді масиву **3R** чисел в тривимірному просторі.

R в цьому записі відповідає за ранг тензора: в тривимірному просторі тензор рангу **2** може бути представлений дев'ятьма числами. В N-вимірному скаляр вимагає тільки одного числа, вектори вимагають N чисел, а тензори вимагають N^R чисел. Цим пояснюється, чому скаляри часто називають тензорами розміру 0: у них немає напрямку, і вони можуть бути представлені тільки одним числом.

Можна досить легко розрізняти вектори, скаляри і тензори: скаляри представляються одним числом, вектори – послідовністю чисел, тензори – масивом чисел.

Що робить тензори такими особливими, так це комбінація компонентів і базисних векторів: всі операції виконуються над тензорами і зберігаються відносини між базисними векторами і такими ж компонентами.

5.2. Початок роботи з TensorFlow

Зазвичай TF-програми запускаються блоками. На перший погляд це суперечить принципам програмування на Python. Однак, якщо ми хочемо, можна також використовувати інтерактивні сесії TF, в яких робота з бібліотекою йде більш тісно. Це особливо зручно, якщо ми раніше працювали з IPython. Зараз розглянемо другий варіант: це допоможе нам вивчити глибоке навчання в TF. Але, перш ніж перейти до складних завдань, давайте спочатку спробуємо вирішити кілька примітивних завдань.

По-перше, імпортуємо бібліотеку `tensorflow` під назвою `tf`, як було показано вище. Потім ініціалізуємо дві змінні-константи. Подаємо масив з чотирьох чисел в функцію `constant()`.

Звернемо увагу, що можна також подати ціле число, але частіше за все ми будемо працювати з масивами. Як ми бачили у введенні, тензори – це і є масиви. Так що переконаємося, що передаємо масив. Потім можна використовувати `multiply()` для перемноження двох змінних. Зберігаємо результат у змінній `result`. Нарешті, виведемо результат за допомогою функції `print()`.

```
# Імпорт `tensorflow`
```

```
import tensorflow as tf

# Ініціалізація двох констант
x1 = tf.constant([1,2,3,4])
x2 = tf.constant([5,6,7,8])

# Множення
result = tf.multiply(x1, x2)

# Виведення результату
print(result)
```

Зауважимо, що ми визначили константи у фрагменті коду вище. Однак є два інших типи значень, які потенційно можемо використовувати, а саме заповнювачі, тобто значення, які не призначені і які будуть ініціалізовані сеансом під час його запуску. Як і назва, це просто заповнювач для тензора, який завжди буде подаватися під час запуску сеансу; Існують також змінні, які є значеннями, які можуть змінюватися. Як вже стало зрозуміло, константи – це значення, які не змінюються.

Результатом рядків коду є абстрактний тензор у графі обчислень. Однак, всупереч тому, що можна було очікувати, результат насправді не обчислюється. Він просто визначив модель, але жоден процес для обчислення результату не запускався. Можемо побачити це на роздруківці: насправді немає результату, який ми хотіли б бачити (а саме 30). Це означає, що TF має ліниве оцінювання.

Однак, якщо хочемо побачити результат, нам доведеться запустити цей код в інтерактивному сеансі. Можемо це зробити кількома способами, як показано в фрагментах коду нижче:

```
# Імпорт `tensorflow`
import tensorflow as tf

# Ініціалізація двох констант
x1 = tf.constant([1,2,3,4])
x2 = tf.constant([5,6,7,8])
```

```
# Множення
result = tf.multiply(x1, x2)

# Ініціалізація Session
sess = tf.Session()

# Виведення результату
print (sess.run(result))

# Закривання сесії
sess.close()
```

Звертаємо увагу, що для запуску інтерактивної сесії можна використати код, наведений нижче. Запустимо `result` і автоматично закриємо сесію після виводу `output`:

```
# Імпорт `tensorflow`
import tensorflow as tf

# Ініціалізація двох констант
x1 = tf.constant([1,2,3,4])
x2 = tf.constant([5,6,7,8])

# Множення
result = tf.multiply(x1, x2)

# Ініціалізація Session і виклик `result`
with tf.Session() as sess:
    output = sess.run(result)
    print(output)
```

В цих блоках ми тільки що визначили сесію за замовчуванням, але можна задавати і параметри. Наприклад, можна вказати аргумент `config`, а потім

використати буфер протоколу `ConfigProto` для додавання параметрів конфігурації сесії.

Наприклад, якщо додати

```
config= tf.ConfigProto (log_device_placement = True)
```

до сесії, то переконаємося, що зареєстрували пристрій GPU або CPU, назначений для цієї операції. Потім отримаємо інформацію про те, які пристрої використовуються в сесії для кожної операції. Якщо маємо м'які обмеження для розміщення пристроїв, то можемо використати наступну конфігурацію сесії:

```
config= tf.ConfigProto (allow_soft_placement = True)
```

Тепер, коли встановили та імпортували TF в робочу область, а також вивчили основи роботи з пакетом, відкладемо ненадовго ці знання і звернемося до даних. Як і завжди, перш ніж приступати до моделювання нейронної мережі, потрібно уважно вивчити свої дані і зрозуміти їх структуру.

5.3. Робота з реальними даними

Перейдемо до роботи з реальними даними. Будемо працювати з дорожніми знаками Бельгії. Дорожній трафік – зрозуміла тема, але не завадить уточнити, які дані включені в датасет, перед тим як приступити до програмування.

Далі ми отримаємо всю інформацію, необхідну для продовження вивчення:

• Текст на дорожніх знаках в Бельгії зазвичай наведено голландською та французькою мовами. Це корисно знати, але для датасету, з яким будемо працювати, це не дуже важливо.

• У Бельгії шість категорій дорожніх знаків: попереджувальні знаки, знаки пріоритету, забороняючі знаки, розпорядчі знаки, знаки, пов'язані з паркуванням та стоянкою біля доріг, і, нарешті, позначення.

• 1 січня 2017 року з бельгійських доріг було знято понад 30 000 дорожніх знаків. Це були забороняючі знаки, які обмежували максимальну швидкість руху.

• Зняття знаків пов'язане з тривалою дискусією в Бельгії (і в усьому Європейському Союзі) про надмірну кількість дорожніх знаків.

Завантаження датасету

Тепер, коли ми познайомилися зі специфікою бельгійського трафіку, давайте завантажимо датасет. Ми повинні завантажити два zip-файли в розділі «BelgiumTS for Classification (cropped images)» з назвами «BelgiumTSC_Training» і «BelgiumTSC_Testing».

Примітка. Після завантаження файлів поглянемо на структуру папок, які ми завантажили. Побачимо, що папки тестування і навчання містять 61 підпапку, відповідну 62 типам дорожніх знаків, які будемо використовувати далі для класифікації. Крім того, виявимо, що файли мають розширення .ppm або Portable Pixmap Format.

Давайте виконаємо імпорт даних в робочу область. Почнемо з рядків коду, які відображаються під користувальницької функцією `load_data()`:

- Спочатку встановимо `ROOT_PATH`. Це шлях до каталога, в якому знаходяться дані для навчання і тестування.
- Потім додамо шлях до нашого `ROOT_PATH` за допомогою функції `join()`. Збережемо цей шлях в `train_data_directory` і `test_data_directory`.
- Бачимо, що після цього можна викликати функцію `load_data()` і передати в неї `train_data_directory`.
- Тепер сама функція `load_data()` запускається шляхом збору всіх підкаталогів, присутніх в `train_data_directory`; вона робить це за допомогою включення списків, що є природним способом складання списків – тобто, якщо ми знайдемо щось в `train_data_directory`, то перевіряємо, чи папка це і, якщо так, то додаємо її в свій список. Пам'ятаємо, що кожен підкаталог є міткою.
- Потім нам треба перебрати підкаталоги. Спочатку ініціалізуємо два списки, `labels` і `images`. Потім збираємо шлях підкаталогів та імена файлів зображень, які зберігаються в цих підкаталогах. Після цього можна зібрати дані в двох списках за допомогою функції `append()`.

```
def load_data(data_directory):
    directories = [d for d in os.listdir(data_directory)
                   if os.path.isdir(os.path.join(data_directory,
d))]
    labels = []
    images = []
    for d in directories:
        label_directory = os.path.join(data_directory, d)
        file_names = [os.path.join(label_directory, f)
                      for f in os.listdir(label_directory)
                      if f.endswith(".ppm")]
        for f in file_names:
            images.append(skimage.data.imread(f))
            labels.append(int(d))
    return images, labels

ROOT_PATH = "/your/root/path"
train_data_directory = os.path.join(ROOT_PATH,
"TrafficSigns/Training")
test_data_directory = os.path.join(ROOT_PATH,
"TrafficSigns/Testing")

images, labels = load_data(train_data_directory)
```

Звернемо увагу, що в наведеному вище блоці дані для навчання і тестування знаходяться в папках з назвами «training» і «testing», які є підкаталогами іншого каталогу «TrafficSigns». На комп'ютері це може виглядати приблизно так: «/Users/Name/Downloads/TrafficSigns», а потім дві папки з назвами «training» і «testing».

Дослідження даних

Дані завантажені, тому прийшов час для їх дослідження. Для початку можна провести елементарний аналіз за допомогою атрибутів `ndim` і `size` масиву `images`.

Звертаємо увагу, що змінні `images` і `labels` є списками, тому нам, можливо, доведеться скористатися `np.array()` для перетворення змінних в масив в робочій області. Але тут це вже зроблено для нас.

```
# Вивести розмірність 'images'
print(images.ndim)

# Вивести кількість елементів в 'images'
print(images.size)

# Вивести перше значення 'images'
images[0]
```

Звернемо увагу, що значення `images[0]`, яке ми вивели, насправді являють собою одне зображення, представлене масивами в масиві. Це може здатися нелогічним, але ми звикнемо до цього, коли будемо працювати із зображеннями в проєктах машинного та глибокого навчання.

Далі, звернемося до `labels`:

```
# Вивести розмірність 'labels'
print(labels.ndim)

# Вивести число елементів в 'labels'
print(labels.size)

# Вивести довжину масиву 'labels'
print(len(set(labels)))
```

Ці цифри дають нам уявлення про те, чи коректно проведений імпорт, і який розмір наших даних. На перший погляд все працює так, як очікували, і видно, що розмір масиву значний, якщо врахувати, що маємо справу з масивами всередині масивів.

Порада. Можна спробувати додати в масиви наступні атрибути, щоб отримати додаткову інформацію про розподіл пам'яті: довжину одного елемента

масиву в байтах і загальну кількість байтів, зайнятих елементами масиву: `flags`, `itemsize` і `nbytes`.

Можна вивчити розподіл дорожніх знаків за типами:

```
# Імпортувати модуль 'pyplot'
import matplotlib.pyplot as plt

# Побудувати гістограму з 64 точками - значеннями 'labels'
plt.hist(labels, 62)

# Вивести графік
plt.show()
```

Давайте подивимось на отриману гістограму (рис.5.2):

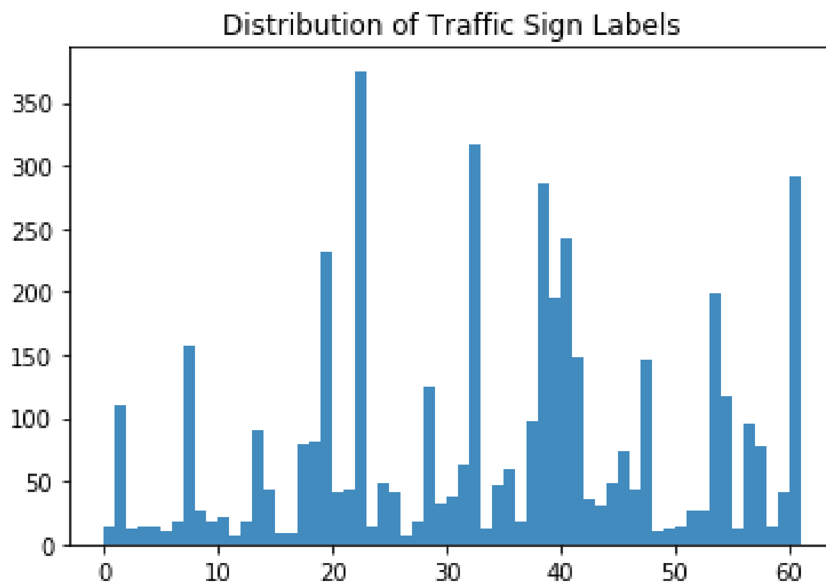

Рисунок 5.2 – Розподіл дорожніх знаків за типами

Бачимо, що не всі типи дорожніх знаків однаково представлені в датасеті. Це те, з чим ми зіткнемося пізніше, коли будемо працювати з даними, перш ніж приступати до моделювання нейронної мережі. На перший погляд, видно: кількість знаків типів 22, 32, 38 і 61 переважає над іншими. Запам'ятаємо це, бо далі ця інформація стане в нагоді.

Візуалізація даних

Ми вже маємо невелике уявлення про дані. Але коли дані – зображення, їх, звичайно, потрібно візуалізувати.

Давайте подивимося на кілька випадкових знаків:

- По-перше, переконаємося, що імпортуємо модуль `pyplot` пакета `matplotlib` під загальноприйнятою назвою `plt`.
- Потім створимо список з 4 випадкових чисел. Вони будуть використовуватися для вибору дорожніх знаків з масиву `images`, завантаженого раніше. У наступному прикладі це будуть числа 300, 2250, 3650 і 4000.
- Потім для кожного елемента цього списку, від 0 до 4, створимо графіки без осей (щоб вони не заважали зосередитися на зображеннях). На цих графіках побачимо конкретні зображення з масиву `images`, які відповідають номеру індексу i. На першому кроці циклу отримаємо i = 300, у другому - 2250 і так далі. Нарешті, потрібно розташувати графіки так, щоб між ними було достатньо простору.
- Останнє – виведемо наші графіки за допомогою функції `show()`

Код:

```
# Імпорт модуля `pyplot` бібліотеки `matplotlib`
import matplotlib.pyplot as plt

# Визначення (випадково) індексів зображень, які хочемо бачити
traffic_signs = [300, 2250, 3650, 4000]

# Заповнимо підграфіки випадковими зображеннями, які визначили
for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(images[traffic_signs[i]])
    plt.subplots_adjust(wspace=0.5)

plt.show()
```

Як можна здогадатися, знаки кожного з 62 типів відрізняються один від одного. Що ще можна помітити? Глянемо на зображення нижче (рис.5.3):

Рисунок 5.3 – Різниця розмірів зображень

Ці чотири зображення мають різний розмір.

Звичайно, ми могли б погратися з числами в списку `traffic_signs` і детальніше вивчити це питання, але, як би там не було, це важливий нюанс, який доведеться враховувати, коли почнемо працювати з даними, які подаємо в нейронну мережу.

Давайте підтвердимо гіпотезу про різні розміри, виводячи роздільну здатність знімків, тобто мінімальні та максимальні значення кожного виведеного зображення. Код нижче дуже схожий на той, який використовувався для створення графіка, але відрізняється тим, що зараз виводиться не лише зображення, а і його розмір (рис.5.4):

```
# Імпорт 'matplotlib'
import matplotlib.pyplot as plt

# Задаємо (випадково) номери зображень, які хочемо вивести
traffic_signs = [300, 2250, 3650, 4000]

# Заповнюємо графіки зображеннями і виводимо розміри
for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(images[traffic_signs[i]])
```

```
    plt.subplots_adjust(wspace=0.5)
    plt.show()
    print("shape: {0}, min: {1}, max:
{2}".format(images[traffic_signs[i]].shape,

images[traffic_signs[i]].min(),

images[traffic_signs[i]].max()))
```

Звертаємо увагу на те, як ми використали `format()` в рядку «`shape: {0}, min: {1}, max: {2}`», щоб заповнити аргументи {0}, {1} и {2}.

Рисунок 5.4 – Зображення із зазначенням розміру

Тепер, коли ми побачили самі зображення, можна повернутись до гістограми, яка була побудована на перших етапах вивчення датасету; можемо легко зробити це, вивівши по одному зображенню для кожного типу знаків:

```
# Імпорт модуля 'pyplot' 'matplotlib'
import matplotlib.pyplot as plt
```

```
# Задаємо типи
unique_labels = set(labels)

# Ініціалізація графіка
plt.figure(figsize=(15, 15))

# Задаємо лічильник
i = 1
# Для кожного типу:
for label in unique_labels:

    # Вибираємо перше зображення кожного типу:
    image = images[labels.index(label)]
    # Задаємо 64 графіка
    plt.subplot(8, 8, i)
    # Вимикання осей
    plt.axis('off')
    # Додаємо заголовок кожному графіку
    plt.title("Label {0} ({1})".format(label,
labels.count(label)))
    # Збільшуємо значення лічильника на 1
    i += 1
    # Виводимо перше зображення
    plt.imshow(image)

# Виводимо весь графік
plt.show()
```

Звертаємо увагу, що навіть якщо визначаємо 64 графіки, не на всіх з них будуть зображення (оскільки є всього 62 типи знаків) (рис.5.5). Знову ж таки, ми не виводимо осі, щоб не відволікатися на них.

Рисунок 5.5 – Типи знаків

Як було видно з гістограми, кількість фотографій знаків з типами 22, 32, 38 і 61 значно більша за інші. Це видно з графіка вище: є 375 знімків з міткою 22, 316 знімків з міткою 32, 285 знімків з міткою 38 і, нарешті, 282 знімків з міткою 61.

Одне з найцікавіших питань, яке можна поставити зараз: чи є зв'язок між усіма цими знаками, можливо, всі вони є знаками позначення?

Давайте розглянемо докладніше: видно, що мітки 22 і 32 є знаками заборони, але мітки 38 і 61 є вказівними знаками і знаками пріоритету, відповідно. Це означає, що між цими чотирма знаками немає безпосереднього зв'язку, за винятком того, що половина знаків, найбільш широко представлених у датасетах, є забороняючими.

Вилучення ознак

Тепер, коли ми ретельно вивчили свої дані, підемо далі. Давайте коротко відзначимо, що ми виявили, щоб переконатися, що не забули ніякі моменти:

• Зображення мають різний розмір.

• Є 62 мітки (пам'ятаємо, що нумерація міток починаються з 0 і закінчуються 61).

• Розподіл типів знаків руху є досить нерівномірним; між знаками, які у великій кількості були присутні в наборі даних, немає жодного зв'язку.

Тепер, коли маємо чітке уявлення про те, з чим потрібно розібратися, можна підготувати дані для передачі в нейронну мережу або будь-яку модель, до якої захочемо їх подавати. Почнемо з отримання ознак – потрібно зробити однаковий розмір зображень і перевести їх у чорно-білу гаму. Перетворювати зображення на відтінки сірого потрібно тому, що колір має менше значення при класифікації. Однак для розпізнавання символів колір відіграє велику роль. Тому у цих випадках перетворення робити не треба.

Масштабування зображень

Щоб зробити розміри зображень однаковими, можна скористатися функцією `skimage` або бібліотекою Scikit-Image, яка є набором алгоритмів для обробки зображень.

У другому випадку буде корисний модуль `transform`, тому що в ньому є функція `resize()`; вона знову використовує включення списку, щоб зробити роздільну здатність знімків рівним 28×28 пікселів. Повторимося: фактично ми складаємо список – для кожного зображення в масиві `image` виконуємо операцію перетворення, яку запозичимо з бібліотеки `skimage`. Нарешті, зберігаємо результат у змінній `images28`:

```
# Імпорт модуля 'transform' із 'skimage'
import transform

# Масштабування зображень в 'image'
array images28 = [transform.resize(image, (28, 28)) for image in
images]
```

Звертаємо увагу, що зображення тепер чотиривимірні: якщо ми конвертуємо `images28` в масив і прив'язуємо атрибут `shape`, видно, що розміри

`images28` рівні (4575, 28, 28, 3). Зображення 784-мірні (бо наші зображення мають розмір 28 на 28 пікселів). Можемо перевірити результат операції масштабування шляхом повторного використання коду, який використовували вище, для побудови 4 випадкових зображень за допомогою змінної `traffic_signs`; просто не забуваємо лише змінити зображення на `images28`.
Результат виконання (рис.5.6):

Рисунок 5.6 – Результат масштабування зображень

Звертаємо увагу, що оскільки ми змінили масштаб, значення `min` і `max` також змінилися; зараз всі вони лежать в одному діапазоні, що дійсно чудово, тому що тепер нам не треба проводити нормування даних.

Перетворення зображень на відтінки сірого

Як зазначено у вступі, колір на фотографіях має менше значення, коли займаємося класифікацією. Ось чому ми зіткнулися з проблемою перетворення зображень у відтінки сірого.

Звертаємо увагу, що ми можемо самостійно перевірити, що станеться з кінцевими результатами роботи нашої моделі, якщо не виконаємо цей конкретний крок. Як і при масштабуванні можемо використати бібліотеку Scikit-Image; у цьому випадку нам знадобиться модуль `color` з функцією `rgb2gray()`.

Все просто, але не забуваємо перетворити змінну `images28` на масив, тому що функція `rgb2gray()` як аргумент приймає саме масиви.

```
# Імпорт `rgb2gray` із`skimage.color`
from skimage.color import rgb2gray

# Конвертація `images28` в масив
images28 = np.array(images28)

# Конвертація `images28` у відтінки сірого
images28 = rgb2gray(images28)
```

Перевіряємо результат нашого перетворення у відтінки сірого, вивівши деякі із зображень; для цього використовується дещо модифікований код:

```
import matplotlib.pyplot as plt

traffic_signs = [300, 2250, 3650, 4000]

for i in range(len(traffic_signs)):

    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(images28[traffic_signs[i]], cmap="gray")
    plt.subplots_adjust(wspace=0.5)

# Виведення графіка
plt.show()
```

Звертаємо увагу, що нам обов'язково потрібно вказати карту кольору або `cmap` і виставити значення '`gray`' для виведення зображень у відтінках сірого (рис.5.7). Це пов'язано з тим, що `imshow()` за умовчанням використовує теплову карту кольору.

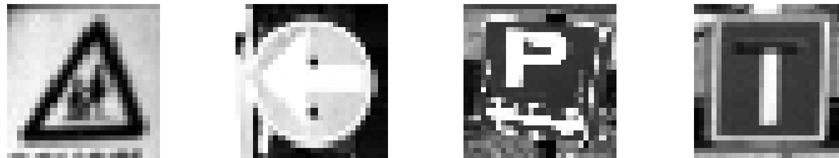

Рисунок 5.7 – Зображення у відтінках сірого

Ці два кроки є дуже простими; інші операції, які можна застосувати до зображень (поворот, розмиття, зміщення, зміна яскравості), призводять до розростання даних. Якщо хочемо, можемо також автоматизувати виконання всієї послідовності дій, яку робимо зі своїми зображеннями.

Тепер, коли ми вивчили та підготували свої дані, настав час створити архітектуру нейронної мережі за допомогою пакету TF.

5.4. Моделювання нейронної мережі

Побудуємо нейронну мережу шар за шаром.

Спочатку імпортуємо `tensorflow` в робочу область під назвою `tf`. Потім ініціалізуємо Graph за допомогою функції `Graph()`. Використовуємо цю функцію для визначення обчислень. Зауважимо, що за допомогою `Graph` ми нічого не обчислюємо, оскільки вона не приймає на вхід жодних змінних. `Graph` лише визначає операції, які хочемо виконати пізніше.

За допомогою `as_default()` встановлюємо контекст за замовчуванням. `as_default()` повертає менеджер контексту, який встановлює заданий нами Graph графом за замовчуванням. Використовуємо цей метод, якщо хочемо створити кілька графів в одному процесі: за допомогою цієї функції створюється глобальний граф за замовчуванням, до якого будуть додаватися всі операції, якщо ми не створимо цілеспрямовано ще один.

Тепер можна додавати операції до графу. Потрібно створити модель нейронної мережі, а потім при її компіляції визначити функцію втрат, оптимізатор та метрику. Коли ми працюємо з TF, це робиться за один крок:

• По-перше, ми задаємо плейсхолдери для вхідних даних та міток, тому що поки що не задаєте «справжні» дані. Пам'ятаймо, що плейсхолдери – це неініціалізовані змінні, які будуть ініціалізовані сесією під час її запуску. Коли ми нарешті запустимо сесію, ці плейсхолдери отримають значення датасета, які передамо у функцію `run()`.

• Потім створюємо мережу. Спочатку потрібно виконати згладжування вхідних даних за допомогою функції `flatten()`, яка створить масив розмірності [None, 784] замість масиву розмірності [None, 28, 28], який є вихідним масивом наших зображень у відтінках сірого.

• Після того, як ми зробили згладжування даних, потрібно створити повністю підключений шар, який генерує логіти розміру [None, 62]. Логіти - це функції, що працюють з немасштабованим вихідним результатом попередніх шарів і використовують відносну шкалу для перевірки лінійності одиниць виміру.

• Після побудови багатошарового персептрону можна визначити функцію втрат. Вибір функції втрат залежить від завдання, яке ми вирішуємо. У нашому випадку використовуємо цю:
`sparse_softmax_cross_entropy_with_logits()`

• Дана функція обчислює розріджену крос-ентропію softmax між логітами та мітками. Інакше кажучи, вона вимірює ймовірність помилки у дискретних завданнях класифікації, у яких класи є взаємовиключними. Це означає, що кожен елемент даних належить лише одному класу. У нашому випадку, наприклад, дорожній знак містить лише одну мітку. Пам'ятаємо, що хоча регресія використовується для прогнозування безперервних значень, класифікація використовується для прогнозування дискретних значень або класів елементів даних. Виконуємо згортку цієї функції з `reduce_mean()`, яка обчислює середнє значення елементів вздовж вимірювань тензора.

• Також потрібно встановити оптимізатор навчання. Найпопулярніші алгоритми оптимізації – SGD, ADAM та RMSprop. Залежно від вибраного алгоритму необхідно налаштувати параметри, наприклад швидкість навчання. У

даному випадку ми вибираємо оптимізатор ADAM, для якого швидкість навчання взята 0,001.

• Нарешті, ініціалізуємо операції, які виконуються перед початком навчання.

```
# Імпорт `tensorflow`
import tensorflow as tf

# Ініціалізація плейсхолдерів
x = tf.placeholder(dtype = tf.float32, shape = [None, 28, 28])

y = tf.placeholder(dtype = tf.int32, shape = [None])

# Згладжування вхідних даних
images_flat = tf.contrib.layers.flatten(x)

# Повністю підключений шар
logits = tf.contrib.layers.fully_connected(images_flat, 62,
tf.nn.relu)

# Визначення функції втрат
losstf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
labels = y, logits = logits))

# Визначення оптимізатора
train_op =
tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)

# Конвертація логітів в індекси міток
correct_pred = tf.argmax(logits, 1)

# Визначення метрики точності
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

Ми тільки що успішно створили свою першу нейронну мережу за допомогою TF.

При бажанні, також можна вивести значення (більшості) змінних, щоб отримати коротке зведення того, що ми тільки що запрограмували:

```
print("images_flat: ", images_flat)
print("logits: ", logits)
print("loss: ", loss)
print("predicted_labels: ", correct_pred)
```

Порада. Якщо бачимо помилку `module 'pandas' has no attribute 'computation'`, треба оновити пакети `dask`, запустивши в командному рядку команду

```
pip install --upgrade dask.
```

Запуск нейронної мережі

Тепер, коли ми зібрали модель мережі шар за шаром, настав час запустити її. Для цього спочатку ініціалізуємо сесію за допомогою `Session()`, якою передаємо граф `graph`, визначений раніше. Потім можемо запустити сесію за допомогою `run()`, якою передаємо ініціалізовані операції у формі змінної `init`, яку також визначили раніше.

Потім можна використовувати цю ініціалізовану сесію для запуску тренувальних циклів. У цьому випадку ми вибираємо 201, тому що хочемо зареєструвати останнє значення `loss_value`. У циклі потрібно запустити сесію з оптимізатором навчання та метрикою втрат (або точністю), яку визначили раніше. Також потрібно передати аргумент функції `feed_dict`, за допомогою якої подаємо дані до моделі. Через кожні 10 циклів отримуємо лог, який дасть нам більше інформації про втрати.

Як було показано в розділі про основи TF, не потрібно закривати сесію вручну – це вже зроблено для нас. Є бажання випробувати іншу конфігурацію? Нам, ймовірно, знадобиться зробити це за допомогою `sess.close()`, якщо ми визначили свою сесію як `sess`, як у блоці нижче:

```
tf.set_random_seed(1234)
```

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())

for i in range(201):
    print('EPOCH', i)
    _, accuracy_val = sess.run([train_op, accuracy], feed_dict={x: images28, y: labels})
    if i % 10 == 0:
        print("Loss: ", loss)

    print('DONE WITH EPOCH')
```

Також можемо запустити наступний блок, але він відразу ж закриває сесію, як було показано раніше:

```
tf.set_random_seed(1234)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(201):
        _, loss_value = sess.run([train_op, loss], feed_dict={x: images28, y: labels})
        if i % 10 == 0:
            print("Loss: ", loss)
```

Звертаємо увагу, що використовується функція `global_variables_initializer()`, тому що функція `initialize_all_variables()` застаріла.

Ми успішно навчили свою модель.

Оцінка нашої нейронної мережі

Ми ще не закінчили: залишилось оцінити нейронну мережу. Отримаємо представлення про роботу своєї моделі, вибравши 10 випадкових зображень і

порівнявши мітки передбачення з реальними. Скористаємося `matplotlib` і виведемо дорожні знаки для візуального порівняння:

```
# Імпорт `matplotlib`
import matplotlib.pyplot as plt
import random

# Вибір 10 випадкових зображень
sample_indexes = random.sample(range(len(images28)), 10)
sample_images = [images28[i] for i in sample_indexes]
sample_labels = [labels[i] for i in sample_indexes]

# Запуск операції "correct_pred"
predicted = sess.run([correct_pred], feed_dict={x:
sample_images})[0]

# Виведення справжніх і передбачених міток
print(sample_labels) print(predicted)

# Виведення передбачених і справжніх зображень
fig = plt.figure(figsize=(10, 10))
for i in range(len(sample_images)):
    truth = sample_labels[i]
    prediction = predicted[i]
    plt.subplot(5, 2,1+i)
    plt.axis('off') color='green' if truth == prediction else
'red'
    plt.text(40, 10, "Truth: {0}nPrediction: {1}".format(truth,
prediction),
             fontsize=12, color=color)
    plt.imshow(sample_images[i], cmap="gray")
plt.show()
```

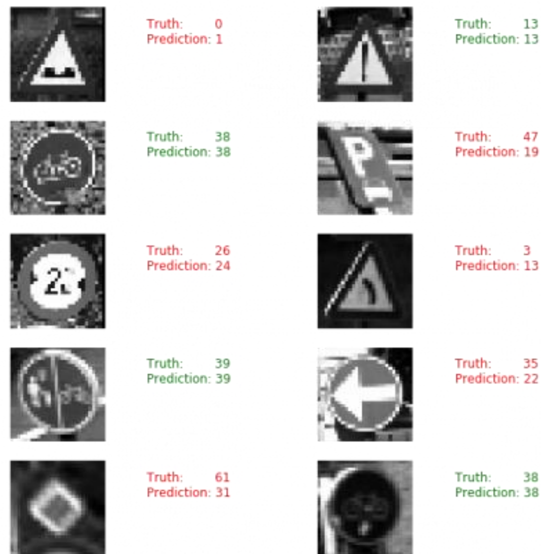

Рисунок 5.8 – Випадкові зображення для візуального порівняння

Перегляд випадкових зображень, однак, не дає багато інформації про те, наскільки насправді добре модель працює. Для цього потрібні тестові дані.

Звертаємо увагу, що використовується функція `load_data()`, яка була визначена на початку.

```
# Імпорт`skimage`
from skimage import transform

# Завантаження даних перевірки
test_images, test_labels = load_data(test_data_directory)

# Перетворення зображень в 28 x 28 пікселів
test_images28 = [transform.resize(image, (28, 28)) for image in
test_images]

# Конвертація в відтінки сірого
from skimage.color import rgb2gray

test_images28 = rgb2gray(np.array(test_images28))

# Виведення передбачень і повного набору даних перевірки
```

```
predicted = sess.run([correct_pred], feed_dict={x:
test_images28})[0]

# Обчислення співпадінь
match_count = sum([int(y == y_) for y, y_ in zip(test_labels,
predicted)])

# Обчислення точності
accuracy = match_count / len(test_labels)
# Виведення точності
print("Accuracy: {:.3f}".format(accuracy))
```

Не забуваємо закрити сесію за допомогою `sess.close()`, якщо не використовували команду `with tf.Session() as sess:`, щоб почати сеанс TF.

5.5. Завдання

1. Повторити розглянуту побудову нейронної мережі.
2. Відстежити помилку при навчанні і тестуванні під час навчання нейронної мережі. Зупинити навчання, коли обидві помилки після зменшення починають раптово збільшуватися – ознака того, що нейронна мережа почала перенавчатися.
3. Спробувати та порівняти між собою інші оптимізатори: Stochastic Gradient Descent (SGD) і RMSprop.

5.3. Зміст звіту

4. Висновки за результатами виконання роботи.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

5.4. Контрольні питання

1. Який буде результат множення двох констант: `x1 = ([1,2,3,4])`і `x2 = [5,6,7,8]`?

2. Для чого вирівнюють розміри зображень?

3. Для чого виконується перетворення зображень в чорно-білу гаму?

Лабораторна робота 6

ПОЧАТОК РОБОТИ З KERAS, DEEP LEARNING і PYTHON

Мета роботи: Вивчити використання Keras, як надбудови TensorFlow, для побудови мережі глибокого навчання розпізнавання зображень.

Зміст. Розглядається покрокове виконання фрагментів коду в середовищі Python. Виконується завантаження набору даних з диску, навчання моделі для класифікації вхідного зображення.

6.1. Теоретичні відомості

Розглянемо, як використовувати бібліотеку Keras для навчання нейронної мережі з власним набором зображень.

Більшість навчальних посібників з Keras засновані на роботі зі стандартними наборами даних, такими як MNIST – розпізнавання рукописного введення цифр, або CIFAR-10 – розпізнавання базових об'єктів. Вони допомагають почати використовувати Keras, але не зможуть навчити нас працювати з власними наборами зображень – ми просто викликаємо допоміжні функції для завантаження заздалегідь скомпільованих наборів даних. Тому, замість того, щоб знову звертатися до наперед компільованих наборі даних, розглянемо, як навчити нашу нейронну мережу на оригінальному наборі зображень, як цього вимагають реальні завдання.

Передбачається покрокове виконання фрагментів коду, для чого знадобиться компілятор Python або середовище Jupyter Notebook.

Набір даних

Спочатку визначимося з методикою підготовки даних та обговоримо структуру проєкту.

Насамперед, відзначимо, що MNIST і CIFAR-10 не є найцікавішими прикладами. Адже навіть навчальну та тестову вибірку вже зробили за нас. А

якщо хочемо використовувати власні зображення, то, швидше за все, не знаючи, з чого почати, виникнуть такі питання:

• звідки ці допоміжні функції завантажують дані?

• у якому форматі мають бути зображення на диску?

• як завантажити свій набір даних в пам'ять?

• яке попереднє оброблення треба виконати?

Спробуємо у всьому цьому розібратися. Для початку візьмемо готовий датасет з тваринами, що складається з фотографій собак, котів та панд (рис.6.1).

Рисунок 6.1 – Датасет з фотографіями тварин

Мета – правильно класифікувати зображення, що містить кота, собаку або панду.

В наборі 3000 зображень і він стане початковим матеріалом, за допомогою якого зможемо швидко навчити модель глибокого навчання (DL – Deep Learning), використовуючи CPU або GPU, і при цьому отримати прийнятну точність.

У процесі роботи з набором даних зможемо зрозуміти, як виконати такі дії:

• впорядкувати набір зображень на диску;

• завантажити зображення та мітки класу з диску;

• розділити дані на навчальну та тестову вибірки;

• навчити нашу нейромережу Keras;

• оцінити нашу модель на тестовій вибірці;

• використати свою навчену модель далі на нових даних.

Якщо хочемо створити набір даних з доступних в Інтернеті зображень, то зробити це можна простим способом за допомогою пошуку картинок Bing або трохи більш складним способом за допомогою пошуковика Google.

Структура проєкту

Розпакувавши zip-архів за посиланням, отримаємо таку структуру файлів та папок (рис.6.2):

```
$ tree --dirsfirst --filelimit 10
.
├── animals
│   ├── cats [1000 entries exceeds filelimit, not opening dir]
│   ├── dogs [1000 entries exceeds filelimit, not opening dir]
│   └── panda [1000 entries exceeds filelimit, not opening dir]
├── images
│   ├── cat.jpg
│   ├── dog.jpg
│   └── panda.jpg
├── output
│   ├── simple_nn.model
│   ├── simple_nn_lb.pickle
│   ├── simple_nn_plot.png
│   ├── smallvggnet.model
│   ├── smallvggnet_lb.pickle
│   └── smallvggnet_plot.png
├── pyimagesearch
│   ├── __init__.py
│   └── smallvggnet.py
├── predict.py
├── train_simple_nn.py
└── train_vgg.py

7 directories, 14 files
```

Рисунок 6.2 – Структура папок і файлів проєкту

Як згадувалося раніше, ми працюємо із набором даних Animals. Звернемо увагу, як він розташований у дереві проєкту. Усередині `animals/` знаходяться каталоги трьох класів: `cats/`, `dogs/`, `panda/`. У кожному з них міститься 1000 зображень, що належать до відповідного класу.

Якщо працюємо з власним набором зображень, рекомендуємо організувати його таким же чином. В ідеалі у нас має бути щонайменше 1000 зображень для кожного класу. Це не завжди можливо, але принаймні класи повинні бути збалансовані. Якщо в одному з класів буде набагато більше зображень, ніж в інших, це може спричинити зміщення моделі.

Далі йде каталог `images/`. Він містить три зображення для тестування моделі, які будемо використовувати, щоб продемонструвати, як:

1. Завантажити навчену модель з диска.

2. Класифікувати вхідне зображення, яке є частиною вихідного набору даних.

Папка `output/` містить три типи файлів, які створюються шляхом навчання:

– `.model`: серіалізований файл моделі Keras, що створюється після навчання і може використовуватись у подальших сценаріях виведення.

– `.pickle`: серіалізований файл бінаризатора міток. Включає об'єкт, що містить імена класів і пов'язаний з файлом моделі.

– `.png`: краще завжди поміщати свої графіки навчання/перевірки в цю папку, оскільки вони відображають результат процесу.

Каталог `pyimagesearch/` – модуль, який знаходиться у папці проєкту. Класи, що містяться в ньому, можуть бути імпортовані у наші сценарії.

Ми розглянемо чотири `.py` файли. Почнемо навчання з простої моделі за допомогою сценарію `train_simple_nn.py`. В наступній ЛР перейдемо до навчання SmallVGGNet, використовуючи сценарію `train_vgg.py`. SmallVGGNet.py містить клас SmallVGGNet (згорткову нейронну мережу). Але що хорошого в серіалізованій моделі, якщо ми не можемо її застосувати? В `predict.py` знаходиться зразок коду для завантаження моделі та файлу мітки для розпізнавання зображень. Цей сценарій знадобиться лише після того, як ми успішно навчимо модель з достатньою точністю. Завжди корисно запускати його для перевірки моделі на зображеннях, які відсутні у вихідних даних.

Встановлення Keras

Для роботи над проєктом знадобиться встановити Keras, TensorFlow та OpenCV. Якщо це програмне забезпечення ще не встановлене, можемо скористатися простими посібниками зі встановлення:

– Посібник із встановлення OpenCV (для Ubuntu, MacOS або RPi).

– Встановлення Keras з TensorFlow. За допомогою `pip` можемо встановити Keras та TensorFlow менше ніж за дві хвилини. Комп'ютер або пристрій має бути достатньо продуктивним. Тому не рекомендується встановлювати ці пакети на RPi, хоча на такому мінікомп'ютері можуть добре працювати вже навчені та не надто об'ємні моделі.

– встановлення `imutil, scikit-learn` та `matplotlib`:

```
pip install --upgrade imutils
pip install --upgrade scikit-learn
pip install --upgrade matplotlib
```

Завантаження даних з диску

Тепер, коли Keras встановлений в нашій системі, можемо приступити до реалізації першого простого сценарію навчання нейронної мережі з використанням Keras. Пізніше ми реалізуємо повноцінну згорткову нейронну мережу, але давайте поступово. Завантаження зображень показано на рис.6.3.

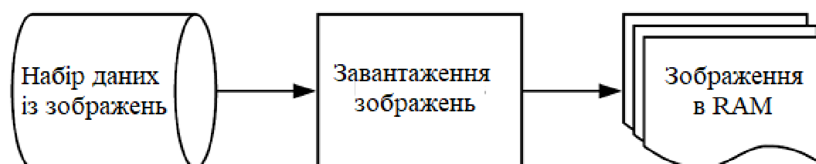

Рисунок 6.3 – Завантаження зображень в оперативну пам’ять

Відкриваємо файл `train_simple_nn.py` і вставляємо в нього наступний код:

```
# імпортуємо бекенд Agg із matplotlib для збереження графіків   #1
import matplotlib                                                #2
matplotlib.use("Agg")                                            #3
                                                                 #4
# імпортуємо необхідні пакети                                    #5
from sklearn.preprocessing import LabelBinarizer                 #6
from sklearn.model_selection import train_test_split             #7
from sklearn.metrics import classification_report                #8
from keras.models import Sequential                              #9
from keras.layers.core import Dense                              #10
from tensorflow.keras.optimizers import SGD                      #11
```

```
from imutils import paths                                          #12
import matplotlib.pyplot as plt                                    #13
import numpy as np                                                 #14
import argparse                                                    #15
import random                                                      #16
import pickle                                                      #17
import cv2                                                         #18
import os                                                          #19
                                                                   #20
```

В сценарії праворуч вказані номера рядків коду, що допоможе нам їх коментувати.

Розглянемо інструменти, які використовуються в наведеному сценарії:

- `matplotlib`: готовий пакет для Python. В рядку 3 ми підключаємо бекенд "`Agg`", який дозволяє зберігати графіки на диск.

- `sklearn`: бібліотека `scikit-learn` допомагає бінаризувати наші мітки, розділити дані на навчальну і тестову вибірки та згенерувати звіт про навчання в терміналі.

- `keras`: високорівневий фронтенд для TF та інших бекендів глибокого навчання.

- `imutils`: пакет із зручними функціями, модуль `path` буде використовуватися для генерації списку шляхів до файлів зображень.

- `numpy`: пакет для роботи з числами в Python. Якщо ми вже встановили OpenCV і `scikit-learn`, то у нас уже є NumPy, як залежний від них пакет.

- `cv2`: це OpenCV. Зараз будемо використовувати версію 2, навіть якщо зазвичай використовуємо OpenCV 3 або вище.

Все решта уже вбудовано у Python.

Тепер маємо уявлення про те, для чого необхідний кожний `import` і для яких задач ми будемо їх використовувати.

Давайте розберемося з аргументами командного рядка за допомогою `argparse`:

```
# створюємо парсер аргументів і передаємо їх                      #21
ap = argparse.ArgumentParser()                                     #22
```

```
ap.add_argument("-d", "--dataset", required=True,                    #23
   help="path to input dataset of images")                            #24
ap.add_argument("-m", "--model", required=True,                      #25
   help="path to output trained model")                               #26
ap.add_argument("-l", "--label-bin", required=True,                  #27
   help="path to output label binarizer")                             #28
ap.add_argument("-p", "--plot", required=True,                       #29
   help="path to output accuracy/loss plot")                          #30
args = vars(ap.parse_args())                                          #31
                                                                      #32
```

Наш сценарій буде динамічно обробляти інформацію, яка поступає з командного рядка під час виконання за допомогою вбудованого в Python модуля `argparse`.

Маємо 4 аргументи командного рядка:

`--dataset`: шлях до набору зображень на диску.

`--model`: наша модель буде серіалізована і записана на диск. Цей аргумент надає шлях до вихідного файлу моделі.

`--label-bin`: мітки набору даних серіалізуются на диск для можливості їх виклику в інших сценаріях. Це шлях до вихідного бінаризованого файлу мітки.

`--plot`: шлях до вихідного файлу графіка навчання. Розглянемо цей графік, щоб перевірити недонавчання або перенавчання наших даних.

Маючи інформацію про набір даних, давайте завантажимо зображення і мітки класів:

```
# ініціалізуємо дані і мітки                                         #33
print("[INFO] loading images...")                                     #34
data = []                                                             #35
labels = []                                                           #36
                                                                      #37
# беремо шляхи до зображень і рандомно перемішуємо                   #38
imagePaths = sorted(list(paths.list_images(args["dataset"])))         #39
random.seed(42)                                                       #40
random.shuffle(imagePaths)                                            #41
                                                                      #42
```

```
# цикл за зображеннями                                          #43
for imagePath in imagePaths:                                    #44
    # завантажуємо зображення, змінюємо розмір на 32x32         #45
    # пікселів (без врахування співвідношення сторін),          #46
    # згладжуємо його в 32x32x3=3072 пікс. і додаємо в список   #47
    image = cv2.imread(imagePath)                               #48
    image = cv2.resize(image, (32, 32)).flatten()               #49
    data.append(image)                                          #50
                                                                #51
    # витягуємо мітку класу зі шляху до зображення              #52
    # і оновлюємо список міток                                  #53
    label = imagePath.split(os.path.sep)[-2]                    #54
    labels.append(label)                                        #55
                                                                #56
```

В сценарії:

1. Ініціалізуємо списки для наших даних (`data`) і міток (`labels`) (рядки 35 і 36). Пізніше це будуть масиви NumPy.

2. Випадковим чином перемішуємо `imagePaths` (рядки 39-41). Функція `paths.list_images` знайде шляхи до всіх вхідних зображень в каталозі нашого датасета перед тим, як відсортуємо і перемішаємо (`shuffle`) їх. Встановимо константне значення `seed` так, щоб випадкове перевпорядкування було відтворюваним.

3. Починаємо цикл за всіма `imagePaths` в наборі даних (рядок 44).

Для кожного `imagePath`:

а) Завантажуємо зображення `image` в пам’ять (рядок 48).

б) Змінюємо його розмір на `32x32` пікселя (без врахування співвідношення сторін) і згладжуємо (`flatten`) (рядок 49). Дуже важливо правильно змінити розмір зображень (`resize`), оскільки це необхідно для конкретної нейронної мережі. Кожна нейромережа вимагає різної роздільної здатності зображень, тому просто будемо пам’ятати про це. Згладжування даних дозволяє легко передавати необроблені інтенсивності пікселів в нейрони вхідного шару. Пізніше побачимо,

що для VGGNet будемо відразу передавати в мережу всі дані, оскільки вона є згортковою. Але в цьому прикладі розглядається проста незгорткова мережа.

в) Додаємо змінене зображення до масиву даних (рядок 50).

г) Витягуємо мітку класу зображення з його шляху (рядок 54) і додаємо до решти міток (рядок 55). Список міток включає класи, які відповідають кожному зображенню в масиві даних.

Тепер можемо легко застосувати операції з масивами до наших даних і міток:

```
# масштабуємо інтенсивності пікселів в діапазон [0, 1]          #57
data = np.array(data, dtype="float") / 255.0                     #58
labels = np.array(labels)                                        #59
                                                                 #60
```

В рядку 58 відображаємо інтенсивність пікселя з діапазону цілих чисел [0, 255] в безперервний натуральний діапазон [0, 1] (звичайний етап попереднього оброблення).

Також конвертуємо мітки в масив NumPy (рядок 59).

Створення навчальної та тестової вибірок

Тепер, коли завантажили дані з диску, треба розділити їх на навчальну і тестову вибірки (рис.6.4).

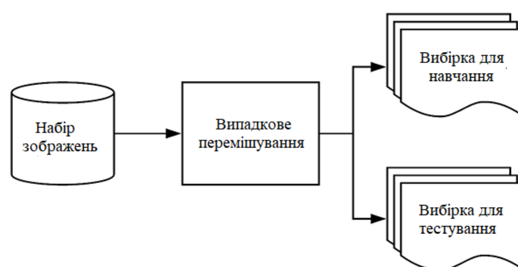

Рисунок 6.4 – Поділ даних на дві вибірки

```
# ділимо дані на навчальну і тестову вибірки, виділивши 75%      #61
# даних для навчання і решту 25% для тестування                  #62
(trainX, testX, trainY, testY) = train_test_split(data,          #63
    labels, test_size=0.25, random_state=42)                     #64
                                                                 #65
```

Зазвичай більша частина даних виділяється для навчання, а близько 20-30% для тестування. Scikit-learn надає зручну функцію `train_test_split`, яка розділить дані.

`trainX` і `testX` – це зображення, а `trainY` і `testY` – відповідні мітки.

Мітки класів зараз представлені у вигляді рядків, але Keras буде вважати, що:

1. Мітки кодуються цілими числами.
2. Для цих міток виконується One-Hot Encoding, в результаті чого кожна мітка буде представлена у вигляді вектору, а не цілого числа.

Для того, щоб виконати це кодування, можна використати клас `LabelBinarizer` із scikit-learn:

```
# конвертуємо мітки з цілих чисел у вектори (для 2-х класів при #66
# бінарній класифікації необхідно використати функцію Keras     #67
# to_categorical замість LabelBinarizer із scikit-learn, яка    #68
# не повертає вектор)                                           #69
lb = LabelBinarizer()                                           #70
trainY = lb.fit_transform(trainY)                               #71
testY = lb.transform(testY)                                     #72
                                                                #73
```

В рядку 70 ініціалізуємо об'єкт `LabelBinarizer`.

Виклик `fit_transform` знаходить всі унікальні мітки класу в `testY`, а потім перетворює їх в мітки One-Hot Encoding.

Виклик `.transform` виконує лише один крок One-Hot Encoding – унікальний набір можливих міток класів уже був визначений викликом `fit_transform`.

Приклад:

```
[1, 0, 0] # відноситься до котів
[0, 1, 0] # відноситься до собак
[0, 0, 1] # відноситься до панди
```

Визначення архітектури моделі Keras

Наступний крок – визначення архітектури нашої нейронної мережі з використанням Keras. Будемо використовувати мережу з одним вхідним шаром (3072 вузла), одним вихідним (3 вузла) і двома прихованими (1024 і 512 вузлів) (рис.6.5).

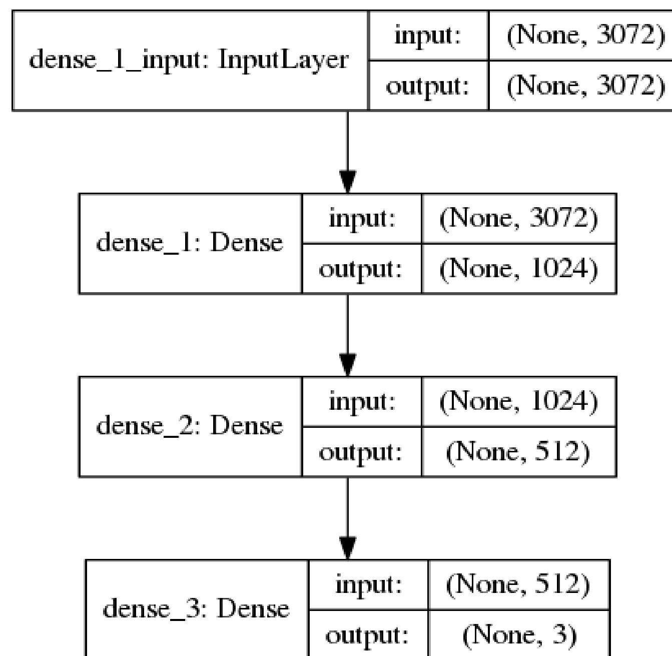

Рисунок 6.5 – Архітектура нейронної мережі

```
# визначаємо архітектуру 3072-1024-512-3 за допомогою Keras     #74
model = Sequential()                                             #75
model.add(Dense(1024, input_shape=(3072,), activation="sigmoid"))
model.add(Dense(512, activation="sigmoid"))                      #77
model.add(Dense(len(lb.classes_), activation="softmax"))         #78
                                                                 #79
```

Оскільки наша модель дуже проста, визначаємо її прямо в цьому сценарії (зазвичай для архітектури моделі доводиться створювати окремий клас).

Вхідний шар і перший прихований шар визначені в рядку 76. `input_shape` буде рівний `3072`, так як маємо 32x32x3=3072 пікселів в згладженому вхідному зображенні. Перший прихований шар буде мати `1024` вузли.

Другий прихований шар має `512` вузлів (рядок 77).

І, нарешті, кількість вузлів вихідного шару (рядок 78) буде рівна числу можливих міток класів – у нашому випадку, вихідний шар буде мати три вузла, один для кожної мітки класу (“cats”, “dogs”, і “panda” відповідно).

Компіляція моделі

Після того, як визначили архітектуру нашої нейронної мережі, нам треба скомпілювати її (рис.6.6).

Рисунок 6.6 – Підготовка нейронної мережі до навчання

```
# ініціалізуємо швидкість навчання і загальне число епох        #80
INIT_LR = 0.01                                                 #81
EPOCHS = 75                                                    #82
                                                               #83
# компілюємо модель, використовуючи SGD як оптимізатор і         #84
# категорійну крос-ентропію в якості функції втрат для бінарної #85
# (класифікації необхідно використати binary_crossentropy)      #86
print("[INFO] training network...")                            #87
opt = SGD(lr=INIT_LR)                                          #88
model.compile(loss="categorical_crossentropy", optimizer=opt,  #89
   metrics=["accuracy"])                                       #90
                                                               #91
```

Спочатку ініціалізуємо швидкість навчання і загальне число епох (повних проходів за вибіркою) (рядки 81 і 82).

Потім скомпілюємо модель, використовуючи метод стохастичного градієнтного спуску (SGD) і "`categorical_crossentropy`" (категоріальну крос-ентропію) в якості функції втрат.

Категоріальна крос-ентропія використовується майже для всіх нейромереж, навчених виконувати класифікацію. Єдиний виняток, коли є лише два класи і дві можливі мітки. В цьому випадку використовується бінарна крос-ентропія ("binary_crossentropy").

Навчання моделі

Тепер, коли наша модель Keras скомпільована, можемо "підігнати" (fit) (тобто, навчити) її (рис.6.7).

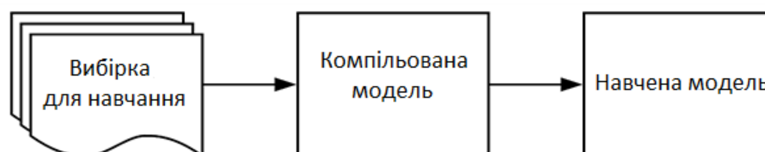

Рисунок 6.7 – Навчання нейромережі

```
# навчаємо нейромережу                                              #92
H = model.fit(trainX, trainY, validation_data=(testX, testY),       #93
    epochs=EPOCHS, batch_size=32)                                   #94
                                                                    #95
```

Тут нам відомо про все, крім batch_size (розмір пакету). Параметр batch_size контролює розмір кожної групи даних для передачі мережею. Потужні GPU можуть обробляти великі пакети, але рекомендується стартувати від розмірів 32 і 64.

Оцінка моделі

Ми навчили модель, тепер треба оцінити її за допомогою тестової вибірки (рис.6.8).

Рисунок 6.8 – Оцінка моделі за допомогою тестової вибірки

Оцінка моделі дозволяє отримати неупереджену уяву про те, наскільки добре наша модель працює з даними, на яких вона ніколи не навчалась.

Для оцінки моделі Keras можна використати комбінацію методів `.predict` і `classification_report` із scikit-learn:

```
# оцінюємо нейромережу                                              #96
print("[INFO] evaluating network...")                               #97
predictions = model.predict(testX, batch_size=32)                   #98
print(classification_report(testY.argmax(axis=1),                   #99
    predictions.argmax(axis=1), target_names=lb.classes_))         #100
                                                                   #101
# будуємо графіки втрат і точності                                 #102
N = np.arange(0, EPOCHS)                                           #103
plt.style.use("ggplot")                                            #104
plt.figure()                                                       #105
plt.plot(N, H.history["loss"], label="train_loss")                 #106
plt.plot(N, H.history["val_loss"], label="val_loss")               #107
plt.plot(N, H.history["acc"], label="train_acc")                   #108
plt.plot(N, H.history["val_acc"], label="val_acc")                 #109
plt.title("Training Loss and Accuracy (Simple NN)")                #110
plt.xlabel("Epoch #")                                              #111
plt.ylabel("Loss/Accuracy")                                        #112
plt.legend()                                                       #113
plt.savefig(args["plot"])                                          #114
                                                                   #115
```

Якщо запустимо цей сценарій, то побачимо, що нейронна мережа почала навчатися, і тепер можемо оцінити модель на тестових даних (рис.6.9):

```
python train_simple_nn.py --dataset animals --model
output/simple_nn.model --label-bin output/simple_nn_lb.pickle
--plot output/simple_nn_plot.png
```

```
$ python train_simple_nn.py --dataset animals --model output/simple_nn.model \
    --label-bin output/simple_nn_lb.pickle --plot output/simple_nn_plot.png
Using TensorFlow backend.
[INFO] loading images...
[INFO] training network...
Train on 2250 samples, validate on 750 samples
Epoch 1/80
2250/2250 [==============================] - 1s 311us/sample - loss: 1.1041 - accuracy: 0.3516
- val_loss: 1.1578 - val_accuracy: 0.3707
Epoch 2/80
...
Epoch 80/80
2250/2250 [==============================] - 0s 181us/sample - loss: 0.7687 - accuracy: 0.6164
- val_loss: 0.8361 - val_accuracy: 0.6120
[INFO] evaluating network...
              precision    recall  f1-score   support

        cats       0.57      0.59      0.58       236
        dogs       0.55      0.31      0.39       236
       panda       0.66      0.89      0.76       278

    accuracy                           0.61       750
   macro avg       0.59      0.60      0.58       750
weighted avg       0.60      0.61      0.59       750

[INFO] serializing network and label binarizer...
```

Рисунок 6.9 – Процес оцінювання нейромережі

Оскільки мережа невелика (як і набір даних), цей процес в середньому займає біля двох секунд. Можна бачити, що наша нейромережа точна на 61%.

Так як шанс випадкового вибору правильної мітки для зображення рівний 1/3, то можемо стверджувати, що мережа фактично вивчила шаблони, які можуть використовуватися для розрізнення трьох класів.

Також ми зберегли наступні графіки (рис.6.10):

- втрати при навчанні (`train_loss`)
- втрати при оцінюванні (`val_loss`),
- точність навчання (`train_acc`),
- точність оцінювання (`val_acc`).

З їх допомогою можемо визначити перенавчання або недонавчання моделі.

Можна побачити невелике перенавчання, яке починається десь після 45 кроку, коли між втратами при навчанні і оцінюванні з’являється явний розрив.

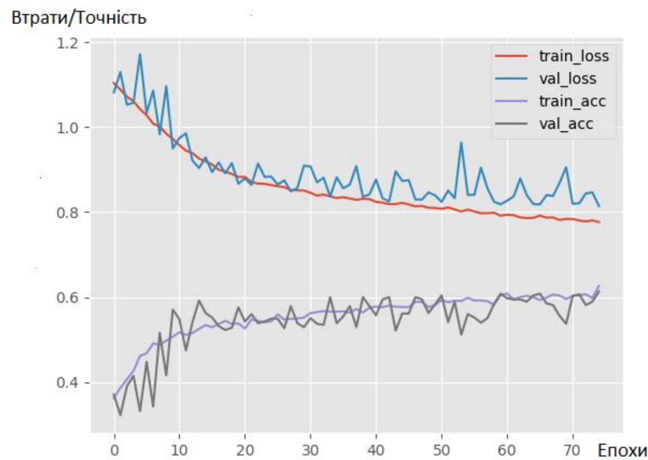

Рисунок 6.10 – Втрати і точність при навчання та оцінювання

Нарешті, можемо зберегти нашу модель на диск, щоб пізніше використати її, не займаючись навчанням знову:

```
# зберігаємо модель і бінаризатор міток на диск                    #116
print("[INFO] serializing network and label binarizer...")      #117
model.save(args["model"])
f = open(args["label_bin"], "wb")                                 #118
f.write(pickle.dumps(lb))                                         #119
f.close()                                                         #121
```

Розпізнавання зображень з використанням навченої моделі

Зараз наша модель навчена – але якщо нам знову треба буде класифікувати нові зображення? Як завантажити модель з диску? Як обробити зображення для класифікації?

Для початку відкриємо сценарій `predict.py` і вставимо туди наступний код:

```
# імпортуємо необхідні пакети                                       #1
from keras.models import load_model                                #2
import argparse                                                    #3
import pickle                                                      #4
import cv2                                                         #5
                                                                   #6
# створюємо парсер аргументів і передаємо їх                       #7
ap = argparse.ArgumentParser()                                     #8
ap.add_argument("-i", "--image", required=True,                    #9
```

```
    help="path to input image we are going to classify")        #10
ap.add_argument("-m", "--model", required=True,                  #11
    help="path to trained Keras model")                          #12
ap.add_argument("-l", "--label-bin", required=True,              #13
    help="path to label binarizer")                              #14
ap.add_argument("-w", "--width", type=int, default=28,           #15
    help="target spatial dimension width")                       #16
ap.add_argument("-e", "--height", type=int, default=28,          #17
    help="target spatial dimension height")                      #18
ap.add_argument("-f", "--flatten", type=int, default=-1,         #19
    help="whether or not we should flatten the image")           #20
args = vars(ap.parse_args())                                     #21
                                                                 #22
```

Спочатку імпортуємо необхідні пакети і модулі.

`load_model` дозволяє завантажити модель Keras з диску. OpenCV буде використовуватися для виведення зображень. Модуль `pickle` завантажує бінаризатор міток.

Далі знову розберемо аргументи командного рядка:

`--image`: шлях до вхідного зображення.

`--model`: шлях до нашої навченої і серіалізованої моделі.

`--label-bin`: шлях до бінаризатора міток.

`--width`: ширина зображення. Пам’ятаємо, що ми не можемо просто вказати тут *будь-що*. Нам треба вказати ширину, для якої назначена модель.

`--height`: висота вхідного зображення. Також повинна відповідати конкретній моделі.

`--flatten`: чи треба згладжувати зображення (за замовчуванням не будемо цього робити).

Завантажимо зображення і змінимо його розмір, виходячи з аргументів командного рядка:

```
# завантажуємо вхідне зображення і підганяємо його розмір        #23
image = cv2.imread(args["image"])                                #24
output = image.copy()                                            #25
```

```
image = cv2.resize(image, (args["width"], args["height"]))       #26
                                                                 #27
# масштабуємо значення пікселів до діапазону [0, 1]              #28
image = image.astype("float") / 255.0                            #29
                                                                 #30
```

Якщо треба, зображення можна згладити:

```
# перевіряємо, чи треба згладити і додати розмір                 #31
# пакета                                                         #32
if args["flatten"] > 0:                                          #33
    image = image.flatten()                                      #34
    image = image.reshape((1, image.shape[0]))                   #35
                                                                 #36
# в іншому випадку працюємо з CNN -- не згладжуємо               #37
# зображення,а просто додаємо розмір пакета                      #38
else:                                                            #39
    image = image.reshape((1, image.shape[0], image.shape[1],    #40
        image.shape[2]))                                         #41
                                                                 #42
```

У випадку з CNN вказуємо розмір пакета, але не виконуємо згладжування (рядки 39-41). Приклад з CNN розглянемо в наступному модулі.

Тепер завантажимо нашу модель і бінаризатор міток в пам’ять і спробуємо розпізнати зображення:

```
# завантажуємо модель і бінаризатор міток                        #43
print("[INFO] loading network and label binarizer...")           #44
model = load_model(args["model"])                                #45
lb = pickle.loads(open(args["label_bin"], "rb").read())          #46
                                                                 #47
# розпізнаємо зображення                                         #48
preds = model.predict(image)                                     #49
                                                                 #50
# знаходимо індекс мітки класу з найбільшою ймовірністю          #51
# відповідності                                                  #52
i = preds.argmax(axis=1)[0]                                      #53
label = lb.classes_[i]                                           #54
                                                                 #55
```

Модель і бінаризатор завантажуються в рядках 45 і 46.

Розпізнавання зображень (прогнозування належності об'єкта до одного з класів) здійснюється за допомогою метода `model.predict` (рядок 49).

Як же виглядає масив `preds`?

```
(Pdb) preds
array([[5.4622066e-01, 4.5377851e-01, 7.7963534e-07]],
dtype=float32)
```

Двовимірний масив має індекс зображення (1) в пакеті (тут він лише один, оскільки було передане одне зображення) і відсотки (2), які відповідають можливій належності зображення до кожної мітки класу:

- cats: 54.6%
- dogs: 45.4%
- panda: ~0%

Тобто, наша нейромережа, ймовірніше всього, бачить кішку, і точно не бачить панду.

В рядку 53 знаходимо індекс найбільшого значення (в даному випадку нульовий).

А в рядку 54 витягуємо рядкову мітку "cats" з бінаризатора міток.

Тепер відобразимо результати:

```
# малюємо мітку класу + ймовірність на вихідному зображенні     #56
text = "{}: {:.2f}%".format(label, preds[0][i] * 100)           #57
cv2.putText(output, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
    (0, 0, 255), 2)                                             #59
                                                                #60
# показуємо вихідне зображення                                  #61
cv2.imshow("Image", output)                                     #62
cv2.waitKey(0)                                                  #63
```

Форматуємо текстовий вивід в рядку 57 (мітку класу і прогнозоване значення у відсотках).

Потім розміщуємо текст на вихідному зображенні (рядки 58 і 59).

Нарешті, виводимо картинку на екран і чекаємо, поки користувач не натисне будь яку клавішу (рядки 62 і 63).

Наш сценарій для розпізнавання зображень виявився досить простим.

Тепер можемо відкрити термінал і спробувати запустити навчену нейромережу на власних фотографіях:

```
python predict.py --image images/cat.jpg --model
output/simple_nn.model \
      --label-bin output/simple_nn_lb.pickle --width 32 --height
32 --flatten 1
Using TensorFlow backend.
[INFO] loading network and label binarizer...
```

Переконаймося, що ми копіювали/вставили команду повністю (з аргументами командного рядка) з папки зі сценарієм.

Наша проста нейромережа класифікувала вхідне зображення як кота з ймовірністю 58,1%, не дивлячись на те, що він сховався в тюльпанах (рис.6.11).

Рисунок 6.11 – Зображення кота, використане в нейромережі

Примітка: Звертаємо увагу, що отримані вами результати можуть відрізнятися від наведених в даному модулі. Швидше всього, це відбувається із-за того, що процес навчання щоразу може проходити по-різному, адже початкові вагові коефіцієнти обираються випадково.

Вихідний код

Код і датасет до ЛР можна завантажити звідси (розмір архіву 246 МБ).

6.2. Завдання

1. Повторити приклад, наведений в даній роботі. *Примітка:* перевірку розпізнавання зображень бажано виконати на зображенні, яке не входить в навчальні та тестові підвибірки.

2. Зробити скріншоти результатів, оформити протокол і завантажити його в Moodle.

6.3. Зміст звіту

2. Сценарій на Python з коментарями.

3. Скріншот результатів виконання сценарію.

4. Висновки за результатами виконання роботи.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

6.4. Контрольні питання

1. Які функції виконують `imutil, scikit-learn` та `matplotlib`?

2. Які функції виконують `argparse`?

3. Що таке «згладжування зображення»?

Лабораторна робота 7

НАВЧАННЯ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ З KERAS

Мета роботи: Дослідити реалізацію згорткові нейронні мережі, яка дозволяє класифікувати зображення з високою точністю.

Зміст. Вивчається SmallVGGNet – зменшений варіант VGGNet. Розглядаються принципи роботи фільтрів в згортковій мережі, функції максимуму, методи побудови шарів мережі, навчання мережі та тестування.

7.1. Загальні відомості

Розглянуте в попередній лабораторній роботі використання стандартної нейронної мережі прямого розповсюдження для класифікації зображень – не краще рішення.

Набагато розумніше було б замість цього використати згорткові нейронні мережі (CNN), які призначені для роботи з інтенсивностями пікселів та вивчення різного типу фільтрів, що дозволяє класифікувати зображення з високою точністю.

Однією з поширених CNN є мережа VGGNet-19, яка має 19 шарів (без шарів підвибірок) (рис.7.1):

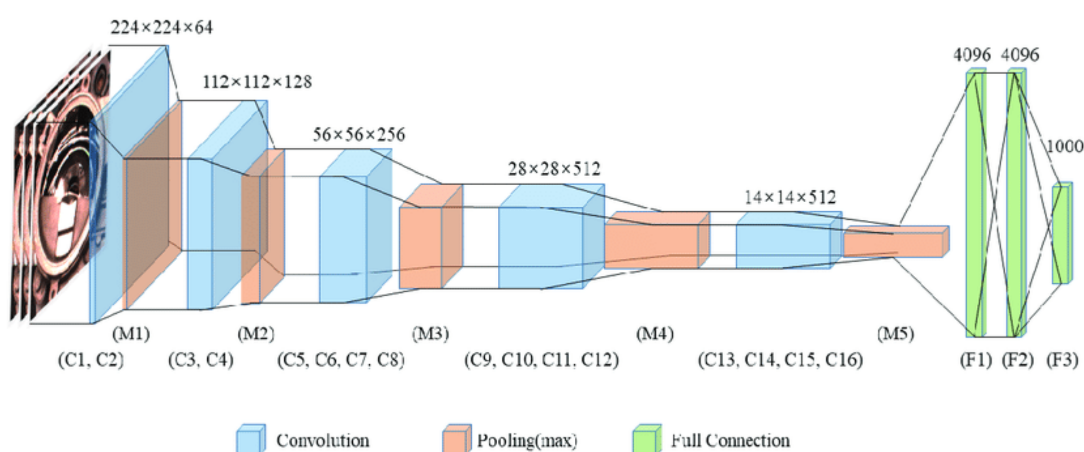

Рисунок 7.1 – Мережа VGGNet-19

В нашій ЛР буде використаний зменшений варіант VGGNet (назвемо її “SmallVGGNet”). В ній лише значно менше шарів (рис.7.2):

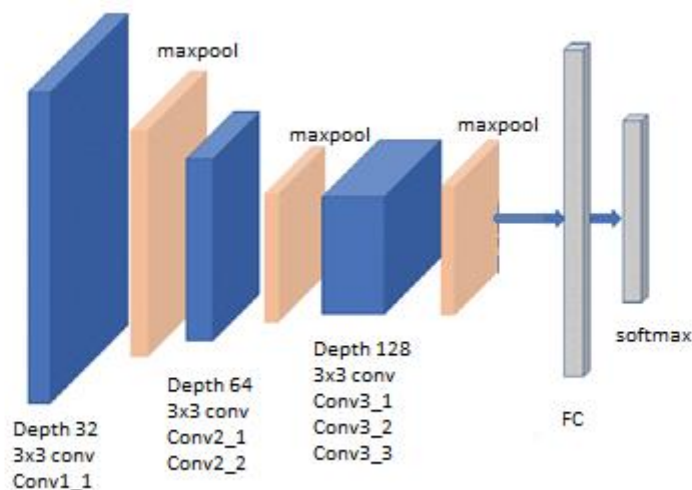

Рисунок 7.2 – Мережа SmallVGGNet

VGGNet-подібні моделі мають дві спільні особливості:

1. Використовуються лише згорткові фільтри 3x3.
2. Згорткові шари (“convolution layers”) чергуються з шарами підвибірки (“pooling layers”).

7.2. Реалізація SmallVGGNet

Відкриваємо файл `smallvggnet.py` і вставляємо туди наступний код:

```
# імпортуємо необхідні пакети                                    #1
from keras.models import Sequential                              #2
from tensorflow.keras.layers.normalization import
 BatchNormalization                                              #3
from keras.layers.convolutional import Conv2D                    #4
from keras.layers.convolutional import MaxPooling2D              #5
from keras.layers.core import Activation                         #6
from keras.layers.core import Flatten                            #7
from keras.layers.core import Dropout                            #8
from keras.layers.core import Dense                              #9
from keras import backend as K                                   #10
                                                                 #11
```

Бачимо, все, що треба для SmallVGGNet, імпортується з Keras. Кожний з модулів описаний в документації Keras.

Визначимо наш клас `SmallVGGNet` (рядок 12) і функцію для методу збірки (`build`) (рядок 14):

```
class SmallVGGNet:                                                  #12
    @staticmethod                                                   #13
    def build(width, height, depth, classes):                       #14
        # ініціалізуємо модель і розмір вхідного зображення        #15
        # для порядку каналів “channel_last” і розмір каналу        #16
        model = Sequential()                                        #17
        inputShape = (height, width, depth)                         #18
        chanDim = -1                                                #19
                                                                    #20
        # якщо використовуємо порядок "channels first",             #21
        # оновлюємо вхідне зображення і розмір каналу               #22
        if K.image_data_format() == "channels_first":               #23
            inputShape = (depth, height, width)                     #24
            chanDim = 1                                             #25
                                                                    #26
```

Для збірки необхідні 4 параметри: ширина вхідних зображень (`width`), висота (`height`), глибина (`depth`) і число класів (`classes`).

Глибина також може інтерпретуватися як число каналів. Оскільки ми використовуємо RGB-зображення, то при виклику метода `build` будемо передавати глибину рівну 3.

Спочатку ініціалізуємо послідовну (`Sequential`) модель (рядок 17).

Потім визначаємо порядок каналів. Keras підтримує `"channels_last"` (TensorFlow) і `"channels_first"` (Theano). Рядки 18-25 дозволяють використати будь-який з них.

Тепер додаємо кілька шарів в мережу:

```
            # додаємо шар CONV => RELU => POOL                     #27
            model.add(Conv2D(32, (3, 3), padding="same",            #28
            input_shape=inputShape))                                #29
            model.add(Activation("relu"))                           #30
            model.add(BatchNormalization(axis=chanDim))             #31
            model.add(MaxPooling2D(pool_size=(2, 2)))               #32
```

```
        model.add(Dropout(0.25))                          #33
                                                          #34
```

В цьому блоці додаються шари `CONV => RELU => POOL`.

Перший шар `CONV` (рис.7.3) має 32 фільтра розміром 3x3. На виході шару формується результат множення значення кожного з 9 пікселів на відповідні значення фільтра (kernel) і їх додавання. Отриманий результат присвоюється вихідній комірці. Далі фільтр зсуваться на один піксель горизонтально або вертикально і описаний алгоритм повтоюється.

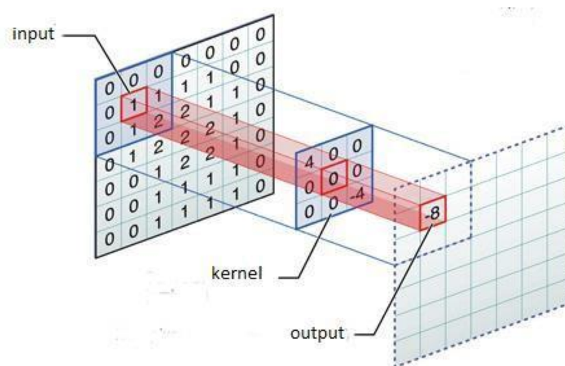

Рисунок 7.3 – Формування вихідного значення в шарі `CONV`

На рис.7.4 демонструється принцип роботи параметра `padding "same"`. При його використанні розмір наступного шару не змінюється.

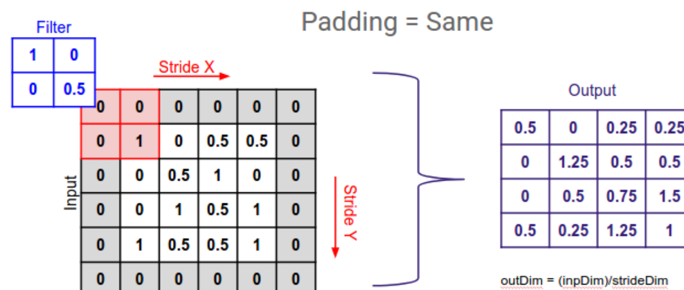

Рисунок 7.4 – Принцип роботи параметра `Padding = Same`

До шарів `POOL` застосовується функція поступового зменшення розміру (тобто ширини і висоти) вхідного шару. На рис.7.5 показано, як працює фільтр в згортковій мережі, «згортаючи» зображення. Чим більші розміри фільтра, тим за меншу кількість ітерацій ми «згортаємо» зображення.

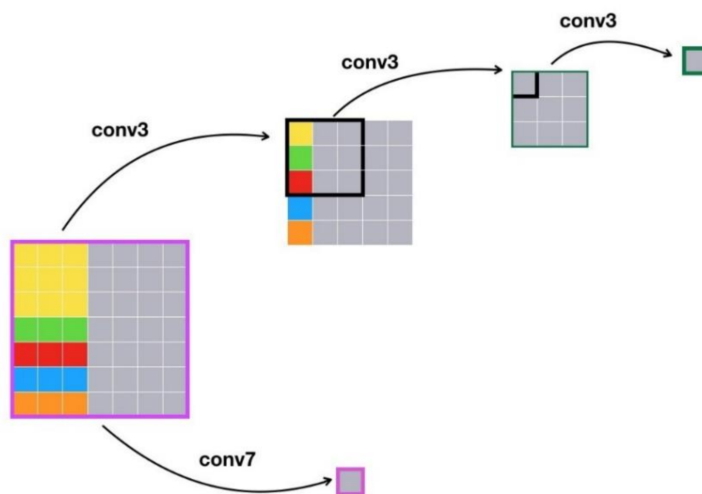

Рисунок 7.5 – Принцип роботи фільтра в згортковій мережі

Зазвичай в архітектурі CNN між послідовно розміщеними шарами `CONV` вставляються шари `POOL`. У нашому випадку використовується `MaxPooling`, який має розмір 2x2. Визначається максимальне значення серед 4-х і воно буде вихідним для шару.

Image Matrix

2	1	3	1
1	0	1	**4**
0	6	**9**	5
7	1	4	1

Max Pool

2	4
7	9

Рисунок 7.6 – Принцип роботи функції максимуму `MaxPooling`

Важливо вказати `inputShape` для першого шару, так як всі наступні розміри шарів будуть розраховуватися з використанням методу “просочування” (trickle-down).

В даній архітектурі використана функція активації ReLU. Також використовуються: розглянута функція максимуму (MaxPooling), пакетна нормалізація (Batch Normalization) і метод виключень (Dropout).

Пакетна нормалізація дозволяє масштабувати вхідні дані для передавання їх на наступний шар мережі. Доведено, що метод ефективно стабілізує і зменшує кількість кроків навчання CNN.

Метод виключення деактивує випадкові нейрони між шарами. В результате процес стає більш стійким: зменшується перенавчання, підвищується точність; і нейромережа зможе краще розпізнавати незнайомі зображення. В нашому випадку (рядок 33) 25% нейронних з'єднань випадковим чином деактивуються між шарами для кожної ітерації навчання.

Переходимо до наступних шарів:

```
# додаємо шари (CONV => RELU) * 2 => POOL             #35
model.add(Conv2D(64, (3, 3), padding="same"))          #36
model.add(Activation("relu"))                          #37
model.add(BatchNormalization(axis=chanDim))            #38
model.add(Conv2D(64, (3, 3), padding="same"))          #39
model.add(Activation("relu"))                          #40
model.add(BatchNormalization(axis=chanDim))            #41
model.add(MaxPooling2D(pool_size=(2, 2)))              #42
model.add(Dropout(0.25))                               #43
                                                       #44
```

Звертаємо увагу, що розміри фільтра залишаються попередніми (3x3), а загальне число фільтрів збільшується з 32 до 64.

Потім йде набір шарів `(CONV => RELU) * 3 => POOL`:

```
# додаємо шари (CONV => RELU) * 3 => POOL             #45
model.add(Conv2D(128, (3, 3), padding="same"))         #46
model.add(Activation("relu"))                          #47
model.add(BatchNormalization(axis=chanDim))            #48
model.add(Conv2D(128, (3, 3), padding="same"))         #49
model.add(Activation("relu"))                          #50
model.add(BatchNormalization(axis=chanDim))            #51
model.add(Conv2D(128, (3, 3), padding="same"))         #52
model.add(Activation("relu"))                          #53
model.add(BatchNormalization(axis=chanDim))            #54
model.add(MaxPooling2D(pool_size=(2, 2)))              #55
model.add(Dropout(0.25))                               #56
                                                       #57
```

Знову ж таки, число фільтрів подвоїлось з 64 до 128, а розмір залишився попереднім. Збільшення загальної кількості фільтрів при зменшенні розміру вхідних даних в CNN – звичайна практика.

І, нарешті, останній набір шарів:

```
# додаємо перший (і єдиний) набір шарів FC => RELU  #58
model.add(Flatten())                                #59
model.add(Dense(512))                               #60
model.add(Activation("relu"))                       #61
model.add(BatchNormalization())                     #62
model.add(Dropout(0.5))                             #63
                                                    #64
# класифікатор softmax                              #65
model.add(Dense(classes))                           #66
model.add(Activation("softmax"))                    #67
                                                    #68
# повертаємо зібрану архітектуру нейронної мережі   #69
return model                                        #70
```

Повністю зв’язані шари в Keras позначаються як `Dense`. Останній шар з’єднаний з трьома виходами (так як в нашому наборі даних три класи). Шар `softmax` повертає ймовірність належності до визначеного класу для кожної мітки.

Тепер, коли реалізували нейромережу SmallVGGNet, напишемо сценарій для її навчання на наборі даних Animals.

Більша частина коду така ж, як і в попередньому модулі. Відкриваємо сценарій `train_vgg.py`:

```
# імпортуємо бекенд Agg із matplotlib для збереження графіків   #1
import matplotlib                                               #2
matplotlib.use("Agg")                                           #3
                                                                #4
# підключаємо необхідні пакети                                  #5
from pyimagesearch.smallvggnet import SmallVGGNet               #6
from sklearn.preprocessing import LabelBinarizer                #7
from sklearn.model_selection import train_test_split            #8
```

```
from sklearn.metrics import classification_report                #9
from keras.preprocessing.image import ImageDataGenerator         #10
from tensorflow.keras.optimizers import SGD                      #11
from imutils import paths                                        #12
import matplotlib.pyplot as plt                                  #13
import numpy as np                                               #14
import argparse                                                  #15
import random                                                    #16
import pickle                                                    #17
import cv2                                                       #18
import os                                                        #19
                                                                 #20
```

Імпортовані модулі мають дві особливсті:

1. Ми завантажуємо модель SmallVGGNet: `from pyimagesearch.smallvggnet import  SmallVGGNet`.
2. Дані будуть доповнюватися за допомогою `ImageDataGenerator`.

Тепер аргументи командного рядка:

```
# створюємо парсер аргументів і передаємо їх                     #21
ap = argparse.ArgumentParser()                                   #22
ap.add_argument("-d", "--dataset", required=True,                #23
    help="path to input dataset of images")                      #24
ap.add_argument("-m", "--model", required=True,                  #25
    help="path to output trained model")                         #26
ap.add_argument("-l", "--label-bin", required=True,              #27
    help="path to output label binarizer")                       #28
ap.add_argument("-p", "--plot", required=True,                   #29
    help="path to output accuracy/loss plot")                    #30
     args = vars(ap.parse_args())                                #31
                                                                 #32
```

Бачимо, що аргументи такі ж, як і в попередньому модулі.

Завантажуємо і попередньо обробляємо дані:

```
# ініціалізуємо дані і мітки                                     #33
print("[INFO] loading images...")                                #34
data = []                                                        #35
```

```
labels = []                                                  #36
                                                             #37
# беремо шляхи до зображень і випадково їх перемішуємо      #38
imagePaths = sorted(list(paths.list_images(args["dataset"])))   #39
random.seed(42)                                              #40
random.shuffle(imagePaths)                                   #41
                                                             #42
# цикл за зображеннями                                       #43
for imagePath in imagePaths:                                 #44
    # завантажуємо зображення,змінюємо розмір на 64x64 пікселів#45
    # (необхідні розміри для SmallVGGNet), змінене зображення   #46
    # зберігаємо в списку                                    #47
    image = cv2.imread(imagePath)                            #48
    image = cv2.resize(image, (64, 64))                      #49
    data.append(image)                                       #50
                                                             #51
    # витягуємо мітку класу із шляху до зображення оновляємо    #52
    # список міток                                           #53
    label = imagePath.split(os.path.sep)[-2]                 #54
    labels.append(label)                                     #55
                                                             #56
# масштабуємо інтенсивність пікселів в діапазон [0, 1]       #57
data = np.array(data, dtype="float") / 255.0                 #58
labels = np.array(labels)                                    #59
                                                             #60
```

Знову майже ніяких відмінностей.

Розділяємо дані на навчальну і тестову вибірки і бінаризуємо мітки:

```
# розбиваємо дані на навчальну і тестову вибірки, використавши 75%
# даних для навчання і решту 25% для тестування             #62
(trainX, testX, trainY, testY) = train_test_split(data,      #63
    labels, test_size=0.25, random_state=42)                 #64
                                                             #65
# конвертуємо мітки з цілих чисел у вектори (для 2-х класів при #66
# бінарній класифікації треба використовувати функцію Keras    #67
# “to_categorical” замість “LabelBinarizer” із scikit-learn, яка
```

```
# не повертає вектор)                                                  #69
lb = LabelBinarizer()                                                  #70
trainY = lb.fit_transform(trainY)                                      #71
testY = lb.transform(testY)                                            #72
                                                                       #73
```

Тепер доповнюємо дані:

```
# створюємо генератор для додавання зображень                          #74
aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
    height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,           #76
    horizontal_flip=True, fill_mode="nearest")                         #77
                                                                       #78
# ініціалізуємо нашу VGG-подібну згорткову нейромережу                 #79
model = SmallVGGNet.build(width=64, height=64, depth=3,                #80
    classes=len(lb.classes_))                                          #81
                                                                       #82
```

В рядках 75-77 ініціалізуємо генератор для додавання зображень.

Це дозволить нам створити додаткові навчальні дані з вже існуючих шляхом повертання, зсуву, обрізання і збільшення зображень.

Доповнення даних дозволить уникнути перенавчання і збільшить ефективність моделі. Рекомендується завжди виконувати цю операцію, якщо немає явних причин цього не робити.

Щоб зібрати нашу SmallVGGNet, просто викликаємо метод `SmallVGGNet.build` в процесі передачі необхідних параметрів (рядки 80 і 81).

Компілюємо і навчимо модель:

```
# ініціалізуємо швидкість навчання, загальне число кроків              #83
# і розмір пакету                                                      #84
INIT_LR = 0.01                                                         #85
EPOCHS = 75                                                            #86
BS = 32                                                                #87
                                                                       #88
# компілюємо модель за допомогою SGD (для бінарної класифікації
# треба використовувати binary_crossentropy)                           #90
print("[INFO] training network...")                                    #91
opt = SGD(lr=INIT_LR, decay=INIT_LR / EPOCHS)                          #92
```

```
model.compile(loss="categorical_crossentropy", optimizer=opt,   #93
    metrics=["accuracy"])                                        #94
                                                                 #95
# Навчаємо нейромережу                                           #96
H = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
    validation_data=(testX, testY), steps_per_epoch=len(trainX) //
BS,                                                              #98
    epochs=EPOCHS)                                               #99
                                                                #100
```

Нарешті, оцінимо модель, побудувавши криві втрат/точності і збережемо її:

```
# оцінюємо нейромережу                                          #101
print("[INFO] evaluating network...")                           #102
predictions = model.predict(testX, batch_size=32)               #103
print(classification_report(testY.argmax(axis=1),               #104
     predictions.argmax(axis=1), target_names=lb.classes_))    #105
                                                                #106
# будуємо графіки втрат і точності                              #107
N = np.arange(0, EPOCHS)                                        #108
plt.style.use("ggplot")                                         #109
plt.figure()                                                    #110
plt.plot(N, H.history["loss"], label="train_loss")              #111
plt.plot(N, H.history["val_loss"], label="val_loss")            #112
plt.plot(N, H.history["acc"], label="train_acc")                #113
plt.plot(N, H.history["val_acc"], label="val_acc")              #114
plt.title("Втрати навчання/Точність(SmallVGGNet)")              #115
plt.xlabel("Епохи")                                             #116
plt.ylabel("Втрати/Точність")                                   #117
plt.legend()                                                    #118
plt.savefig(args["plot"])                                       #119
                                                                #120
# зберігаємо модель і бінаризатор міток на диск                 #121
print("[INFO] serializing network and label binarizer...")     #122
model.save(args["model"])                                       #123
f = open(args["label_bin"], "wb")                               #124
```

```
f.write(pickle.dumps(lb))                                             #125
f.close()                                                             #126
```

Прогнози робимо на тестовій вибірці, а потім оцінюємо точність класифікації (рядки 103-105). Побудова та збереження на диску графіків, моделі і міток аналогічно попередньому модулю.

Продовжимо навчати нашу модель. Відкриваємо термінал і виконуємо наступну команду:

```
python train_vgg.py --dataset animals --model
output/smallvggnet.model \
      --label-bin output/smallvggnet_lb.pickle \
      --plot output/smallvggnet_plot.png
Using TensorFlow backend.
[INFO] loading images...
[INFO] training network...
Epoch 1/75
70/70 [==============================] - 3s - loss: 1.3783 - acc:
0.5165 - val_loss: 2.3654 - val_acc: 0.3133
Epoch 2/75
70/70 [==============================] - 2s - loss: 1.0382 - acc:
0.5998 - val_loss: 2.7962 - val_acc: 0.3173
...
Epoch 74/75
70/70 [==============================] - 2s - loss: 0.4306 - acc:
0.8055 - val_loss: 0.6150 - val_acc: 0.7520
Epoch 75/75
70/70 [==============================] - 2s - loss: 0.4179 - acc:
0.8110 - val_loss: 0.5624 - val_acc: 0.7653
[INFO] evaluating network...
             precision    recall  f1-score   support
      cats        0.62      0.84      0.71       236
      dogs        0.75      0.50      0.60       236
     panda        0.95      0.92      0.93       278

avg / total       0.78      0.77      0.76       750
```

```
[INFO] serializing network and label binarizer...
```

Переконаємось, що ввели всі аргументи командного рядка.

Навчання на CPU займає досить тривалий час – кожна з 75 епох вимагає більше хвилини, і процес буде продовжуватися близько півтори години.

GPU завершує процес значно швидше – кожний крок ітерації виконується всього за 2 секунди, як і продемонстровано.

Подивимось на підсумковий графік навчання в каталозі `output/` (рис.7.7):

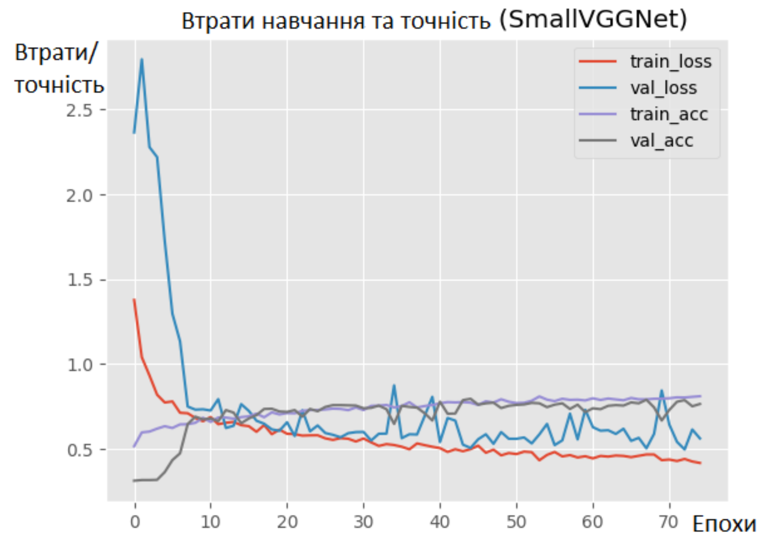

Рисунок 7.7 – Втрати навчання і точність

Як бачимо, досягнута точність 78% на наборі зображень Animals з використанням згорткової нейронної мережі, що значно вище, ніж попереднє значення в 60%.

Тепер можемо застосувати нашу навчену CNN до нових зображень:

```
python predict.py --image images/panda.jpg --model
output/smallvggnet.model \
      --label-bin output/smallvggnet_lb.pickle --width 64 --height
64
Using TensorFlow backend.
[INFO] loading network and label binarizer...
```

Реалізована CNN на 100% впевнена, що на зображенні панда (рис.7.8).

Рисунок 7.8 – Навчена CNN впевнена, що це панда

Алгоритми CNN використовуються для пошуку зображень, наприклад, в Google Photo, але розпізнавання і класифікація фотографій – не єдиний приклад використання згорткових нейромереж: вони також добре себе показали, наприклад, в задачах обробки природньої мови (Natural Language Processing, NLP).

Вихідний код

Код і датасет до ЛР можна завантажити звідси (розмір архиву 246 МБ).

С оригінальними матеріалами можна познайомитися на сайті pyimagesearch.com.

7.3. Завдання

3. Повторити приклади, наведені в даній роботі.
4. Зробити скріншоти результатів, оформити протокол і завантажити його в Moodle.

7.4. Зміст звіту

1. Сценарій на Python для керування кроковим двигуном з коментарями.
2. Висновки за результатами виконання роботи.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

7.5. Контрольні питання

1. Що таке повністю зв’язані шари?
2. Які функції методу “просочування” (trickle-down)?

Лабораторна робота 8

КЛАСИФІКАТОР ЗОБРАЖЕНЬ З ВИКОРИСТАННЯМ АЛГОРИТМУ k-NN

Мета роботи: Навчитися застосувати алгоритм k-NN для розпізнавання об'єктів на зображеннях.

Зміст. Розглядаються початкові налаштування для глибокого навчання, реалізація класифікатора k-NN: збирання набору даних, його розділення, тренування класифікатора, оцінювання.

8.1. Особливості застосування

Спробуємо створити власний класифікатор зображень. Почнемо зі створення декількох допоміжних утиліт для полегшення попередньої обробки та завантаження зображень з диска. Потім обговоримо класифікатор k-найближчих сусідів (k-Nearest Neighbors – k-NN) – алгоритм використання машинного навчання для класифікації зображень. Цей алгоритм настільки простий, що фактично взагалі не виконує жодного «навчання», але все-таки він важливий для розгляду, щоб можна було оцінити, як нейронні мережі вчаться на даних. Нарешті, покажемо, як застосувати алгоритм k-NN для розпізнавання різних видів тварин на зображеннях.

Робота з наборами зображень

Під час роботи з наборами зображень спочатку треба враховувати загальний розмір набору даних (в байтах). Чи є набір даних достатньо великим, але щоб вміщався в доступну оперативну пам'ять на нашому комп'ютері? Чи можемо завантажувати набір даних так, ніби завантажуємо велику матрицю або масив? Чи набір даних настільки великий, що перевищує пам'ять нашого комп'ютера, що вимагає від нас "розбити" набір даних на сегменти та завантажувати лише частину за раз?

Менші набори даних можемо завантажувати в основну пам’ять, однак для набагато більших наборів даних треба розробляти різні розумні методи, які ефективно обробляють завантаження зображень таким чином, щоб можна було без втрати пам'яті навчити класифікатор зображень.

Тим не менш, ми завжди повинні знати розмір набору даних, перш ніж навіть починати працювати з алгоритмами класифікації зображень. Як побачимо в даній лабораторній роботі, витрати часу на організацію, попередню обробку та завантаження набору даних є критичними аспектами побудови класифікатора зображень.

Представлення набору даних Animals

Набір даних Animals – це простий приклад набору даних, який зібраний, щоб продемонструвати, як тренувати класифікатори зображень за допомогою простих методів машинного навчання, а також вдосконалених алгоритмів глибокого навчання. Зображення всередині набору даних Animals належать до трьох різних класів: собаки, коти та панди, як ви можете бачити на рис.8.1, з 1000 прикладами зображень на клас, відповідно, загальною кількістю 3000 зображень. Зображення собак та котів були зібрані за посиланням Kaggle Dogs vs. Cats challenge, тоді як зображення панд – з набору даних ImageNet dataset.

Рисунок 8.1 – Зразок набору даних 3-х класів Animals

Маючи лише 3000 зображень, набір даних Animals може легко поміститися в основну пам'ять комп'ютера, що набагато швидше навчить наші моделі, не вимагаючи від нас написання будь-якого «накладного коду» для управління набором даних, які не могли б поміститися в пам'ять. Модель глибокого навчання можна швидко навчити на цьому наборі даних як на центральному, так і на графічному процесорі. Незалежно від налаштування обладнання, можемо використовувати цей набір даних для вивчення основ машинного навчання та глибокого навчання.

Мета даного заняття – використати класифікатор k-NN для спроби розпізнати кожен з вибраних видів на зображенні, використовуючи лише інтенсивність пікселів (тобто вилучення ознак не відбувається). Як побачимо далі, інтенсивність необроблених пікселів погано піддається алгоритму k-NN. Тим не менше, це важливий базовий експеримент для початку, щоб можна було зрозуміти, чому згорткові нейронні мережі можуть отримати таку високу точність щодо інтенсивності необроблених пікселів, тоді як традиційні алгоритми машинного навчання цього не роблять.

8.2. Початкові налаштування для глибокого навчання

Давайте визначимо структуру проєкту нашого набору інструментів:

```
|--- pyimagesearch
```

Як бачимо, у нас є єдиний модуль під назвою `pyimagesearch`. Весь код, який розробляємо, буде існувати всередині модуля `pyimagesearch`. Для цілей даної ЛР нам потрібно буде визначити два підмодулі:

```
|--- pyimagesearch
|      |--- __init__.py
|      |--- datasets
|      |      |--- __init__.py
|      |      |--- simpledatasetloader.py
|      |--- preprocessing
|      |      |--- __init__.py
|      |      |--- simplepreprocessor.py
```

Підмодуль наборів даних з назвою `SimpleDatasetLoader` розпочне нашу реалізацію класу. Будемо використовувати цей клас для завантаження наборів даних невеликих зображень з диска (які можуть розміститися в основній пам'яті), необов'язково попередньо обробляти кожне зображення в наборі даних відповідно до набору функцій, а потім повернути:

1. Зображення (тобто інтенсивність пікселів)
2. Мітка класу, пов'язана з кожним зображенням

Маємо також підмодуль попереднього оброблення. Існує ряд методів попереднього оброблення, які можемо застосувати до нашого набору зображень для підвищення точності класифікації, включаючи середнє віднімання, вибірку випадкових плям або просто зменшення розміру зображення до фіксованого розміру. У даному випадку наш клас `SimplePreprocessor` зробить останнє - завантажить зображення з диска та змінить його до фіксованого розміру, ігноруючи співвідношення сторін. У наступних розділах будемо використовувати `SimplePreprocessor` та `SimpleDatasetLoader` вручну.

Примітка. Хоча ми будемо розглядати весь модуль `pyimagesearch` для глибокого навчання, але пояснення файлів `__init__.py` залишаються як вправа для студентів. Ці файли просто містять імпорт міток і не мають значення для розуміння методів глибокого навчання та машинного навчання, що застосовуються до класифікації зображень. Щоб освіжити знання з мови Python, радимо розглянути основи імпорту пакетів.

Основний попередній процесор зображення

Алгоритми машинного навчання, такі як k-NN, SVM і навіть згорткові нейронні мережі, вимагають, щоб усі зображення в наборі даних мали фіксований розмір вектору ознак. Що стосується зображень, ця вимога передбачає, що наші зображення повинні бути попередньо оброблені та масштабовані, щоб мати однакову ширину та висоту.

Існує ряд способів зробити це зменшення та масштабування, починаючи від більш досконалих методів, що дотримують пропорції вихідного зображення, до масштабованого зображення, і закінчуючи простими методами, які ігнорують співвідношення сторін і просто стискають ширину та висоту до необхідних розмірів. Який саме метод нам слід використовувати, насправді залежить від складності факторів варіації – в деяких випадках ігнорування пропорції працює чудово; в інших випадках ми хочемо зберегти пропорції.

Почнемо з основного рішення: побудова препроцесора зображення, який змінює розмір зображення, ігноруючи співвідношення сторін. Створюємо файл `simplepreprocessor.py`, а потім вставляємо такий код:

```
# імпорт необхідних пакетів                                    #1
import cv2                                                     #2
                                                               #3
class SimplePreprocessor:                                      #4
    def __init__(self, width, height, inter=cv2.INTER_AREA):   #5
        # зберігаємо цільову ширину,висоту зображення та       #6
        # метод інтерполяції,використаний при зменшенні        #7
        self.width = width                                     #8
        self.height = height                                   #9
        self.inter = inter                                     #10
                                                               #11
    def preprocess(self, image):                               #12
        # зменшення зображення до фіксованого розміру,         #13
        # ігноруючи співвідношення сторін                      #14
        return cv2.resize(image, (self.width, self.height),    #15
            interpolation=self.inter)                          #16
```

Рядок 2 імпортує наш єдиний необхідний пакет, наші прив'язки OpenCV. Далі визначаємо конструктор до класу `SimplePreprocessor` у рядку 5. Конструктору потрібні два аргументи, за якими слідує третій необов'язковий, кожен з яких детально описаний нижче:

- `width`: цільова ширина нашого вхідного зображення після зміни розміру,

- `height`: цільова висота нашого вхідного зображення після зміни розміру,
- `inter`: необов'язковий параметр, який використовується для контролю того, який алгоритм інтерполяції використовується при зміні розміру.

Функція попереднього оброблення в рядку 12 вимагає одного аргументу – вхідного зображення, яке ми хочемо попередньо обробити.

Рядки 15 і 16 попередньо обробляють зображення, змінюючи його розмір до фіксованого розміру ширини та висоти, які ми потім повертаємо до функції виклику.

Знову ж таки, цей препроцесор за визначенням дуже базовий – все, що ми робимо, – це прийняти вхідне зображення, змінити його розмір до фіксованого виміру, а потім повернути. Однак у поєднанні з завантажувачем набору зображень далі цей препроцесор дозволить нам швидко завантажувати та попередньо обробляти набір даних з диска, дозволяючи швидко рухатись через наш конвеєр класифікації зображень і переходити до більш важливих аспектів, таких як фактичний класифікатор.

Створення завантажувача зображень

Тепер, коли наш `SimplePreprocessor` визначений, перейдемо до `SimpleDatasetLoader`:

```
# імпорт необхідних пакетів                                   #1
import numpy as np                                            #2
import cv2                                                    #3
import os                                                     #4
                                                              #5
class SimpleDatasetLoader:                                    #6
    def __init__(self, preprocessors=None):                   #7
        # зберігаємо препроцесор зображення                   #8
        self.preprocessors = preprocessors                    #9
                                                              #10
        # якщо препроцесор рівний None, ініціалізуємо їх як   #11
        # порожній список                                     #12
```

```
        if self.preprocessors is None:                          #13
            self.preprocessors = []                             #14
                                                                #15
```

Рядки 2-4 імпортують необхідні пакети Python: `numpy` для чисельної обробки, `cv2` для наших прив'язок OpenCV та `os`, щоб ми могли витягувати імена підкаталогів у шляхи до зображень.

Рядок 7 визначає конструктор для `SimpleDatasetLoader`, куди можемо додатково передати список попередніх процесорів зображень (наприклад, `SimplePreprocessor`), які можна послідовно застосувати до даного вхідного зображення. Важливо вказати ці препроцесори як список, а не як одне значення - бувають випадки, коли нам спочатку потрібно змінити розмір зображення до фіксованого розміру, потім виконати певне масштабування (наприклад, середнє віднімання), після чого перетворити масив зображень до формату, придатного для Keras. Кожен з цих препроцесорів може бути реалізований незалежно, що дозволяє ефективно застосовувати їх до зображення послідовно.

Тепер можемо перейти до методу завантаження, ядра `SimpleDatasetLoader`:

```
    def load(self, imagePaths, verbose=-1):                     #16
        # ініціалізуємо список ознак і міток                    #17
        data = []                                               #18
        labels = []                                             #19
                                                                #20
        # цикл по вхідних зображеннях                           #21
        for (i, imagePath) in enumerate(imagePaths):            #22
            # завантажуємо зображення і витягуємо мітку класу,  #23
            # враховуючи,що наш шлях має наступний формат:      #24
            # /path/to/dataset/{class}/{image}.jpg              #25
            image = cv2.imread(imagePath)                       #26
            label = imagePath.split(os.path.sep)[-2]            #27
                                                                #28
```

Наш метод завантаження вимагає єдиного параметра – `imagePaths`, тобто списку, в якому вказані шляхи до файлів зображень у наборі даних, що знаходяться на диску. Також можемо вказати значення для деталізації. Цей

“рівень деталізації” можна використовувати для друку оновлень на консолі, що дозволяє нам контролювати, скільки зображень обробив `SimpleDatasetLoader`.

Рядки 18 і 19 ініціалізують наш список даних (тобто самі зображення) разом з мітками, списком міток класів для наших зображень.

У рядку 22 ми починаємо цикл по кожному з вхідних зображень. Для кожного з цих зображень завантажуємо його з диска (рядок 26) і витягуємо мітку класу на основі шляху до файлу (рядок 27). Робимо припущення, що наші набори даних організовані на диску згідно з такою структурою каталогів:

```
/dataset_name/class/image.jpg
```

Ім'ям набору даних (`dataset_name`) може бути будь-яке ім'я набору даних, в даному випадку `animals`. Клас повинен мати назву мітки класу. Для нашого прикладу `class` – це `dog` (собака), `cat` (кішка) або `panda` (панда). Нарешті, `image.jpg` – це власне назва самого зображення.

Виходячи з цієї ієрархічної структури каталогів, можемо тримати наші набори даних охайними та організованими. Таким чином, можна впевнено вважати, що всі зображення всередині підкаталогу `dog` є прикладами собак. Подібним чином припускаємо, що всі зображення в каталозі `panda` містять приклади панд. Майже кожен набір даних, який розглядаємо, буде дотримуватись цієї ієрархічної структури дизайну каталогів – *настійно рекомендується робити те саме для своїх власних проєктів*.

Тепер, коли наше зображення завантажено з диска, можемо його попередньо обробити (при необхідності):

```
# перевіряємо, чи наші препроцесори не є None            #29
if self.preprocessors is not None:                         #30
    # цикл через препроцесор і застосування                #31
    # кожного до зображення                                #32
    for p in self.preprocessors:                           #33
        image = p.preprocess(image)                        #34
                                                           #35
# розглядаємо наше оброблене зображення як "вектор ознак"
# шляхом оновлення списку даних з наступними мітками     #37
```

```
        data.append(image)                                         #38
        labels.append(label)                                       #39
                                                                   #40
```

Рядок 30 робить швидку перевірку, щоб переконатися, що наші препроцесори не є `None`. Якщо перевірка проходить, прокручуємо кожен з препроцесорів в рядку 33 і послідовно застосовуємо їх до зображення в рядку 34 – ця дія дозволяє нам сформувати *ланцюжок препроцесорів*, який можна застосувати до кожного зображення в наборі даних.

Після попереднього оброблення зображення оновлюємо дані та списки міток, відповідно (рядки 38 та 39).

Наш останній блок коду просто обробляє оновлення друку на нашій консолі, а потім повертає кортеж: набір даних та міток до функції виклику:

```
        # показувати оновлення кожного «багатослівного» зображення
        if verbose > 0 and i > 0 and (i + 1) % verbose == 0:    #42
            print("[INFO] processed {}/{}".format(i + 1,        #43
                len(imagePaths)))                              #44
    # повернення кортежу даних і міток                          #45
    return (np.array(data), np.array(labels))                  #47
```

Як бачите, наш завантажувач набору даних простий за своїм дизайном; однак це дає можливість з легкістю застосовувати будь-яку кількість препроцесорів зображень до *кожного* зображення в наборі даних. Єдине застереження цього навантажувача наборів даних полягає в тому, що він передбачає, що всі зображення в наборі даних можуть одночасно поміститися в основну пам’ять.

Для наборів даних, які занадто великі, щоб вміститися в оперативній пам’яті нашої системи, треба буде розробити більш складний завантажувач наборів даних.

k-NN: простий класифікатор

Класифікатор k-найближчих сусідів *сьогодні* є найпростішим алгоритмом машинного навчання та класифікації зображень. Насправді це *настільки просто*,

що насправді нічого не «навчає». Натомість цей алгоритм безпосередньо покладається на відстань між векторами об’єктів (а це, в нашому випадку, вихідна інтенсивність пікселів RGB-зображень).

Простіше кажучи, алгоритм k-NN класифікує невідомі точки даних, знаходячи *найпоширеніший клас* серед *k найближчих прикладів*. Кожна точка даних у *k* найближчих точках даних надає голос, і виграє категорія з найбільшою кількістю голосів, як показано на рис.8.2. Враховуючи наш набір даних про собак, котів, панд, як можемо класифікувати зображення, позначене червоним?

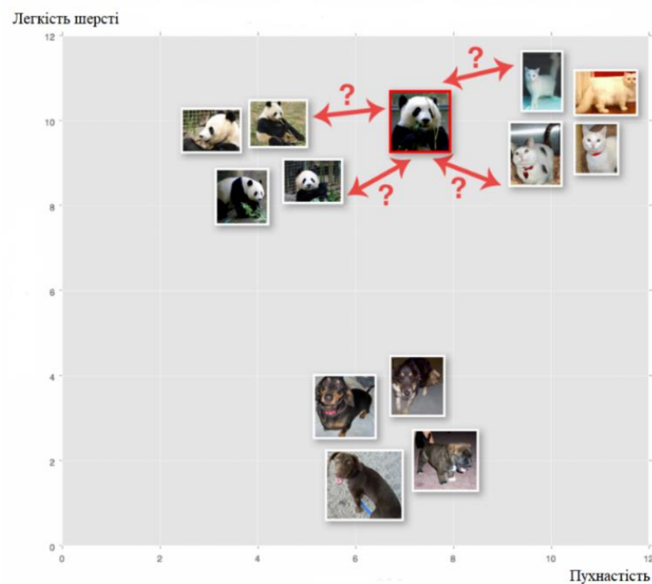

Рисунок 8.2 – Як класифікувати зображення, позначене червоним?

Для того, щоб алгоритм k-NN працював, робиться основне припущення, що зображення з подібним візуальним вмістом *лежать близько* в *n*-мірному просторі. Можемо побачити три категорії зображень, позначених як *dogs*, *cats* та *pandas*, відповідно. У цьому придуманому прикладі ми побудували графік «пухнастості» шерсті тварини вздовж *осі x* та «легкості» шерсті вздовж *осі y*. Кожна з точок даних про тварин згрупована відносно близько в нашому *n*-вимірному просторі. Це означає, що відстань між двома зображеннями *cats* *набагато менша*, ніж відстань між *cat* і *dog*.

Однак, щоб застосувати класифікатор k-NN, нам спочатку треба вибрати метрику відстані або функцію подібності. Загальним вибором є евклідова відстань (яку часто називають L2-відстань):

$$d(\boldsymbol{p},\boldsymbol{q}) = \sqrt{\sum_{i=1}^{N}(q_i - p_i)^2}$$

Однак, можуть використовуватися й інші показники відстані, такі як блок Манхеттен/місто (який часто називають L1-відстань):

$$d(\boldsymbol{p},\boldsymbol{q}) = \sum_{i=1}^{N}|q_i - p_i|$$

Насправді, можемо використовувати будь-яку метрику відстані/функцію подібності, яка найбільше відповідає нашим даним (і дає найкращі результати класифікації). Однак до кінця даної ЛР будемо використовувати найпопулярнішу метрику відстані: евклідову відстань.

Приклад працюючого k-NN

Вже зрозумілі принципи алгоритму k-NN. Ми знаємо, що він покладається на відстань між векторами/зображеннями об'єктів, щоб зробити класифікацію. І ми знаємо, що для обчислення цих відстаней потрібна функція відстані/подібності.

Але як насправді *зробити* класифікацію? Щоб відповісти на це питання, давайте розглянемо рис.8.3. Маємо набір даних про три типи тварин – собак, котів та панд – і ми побудували набори на основі пухнастості та легкості шерсті цих тварин.

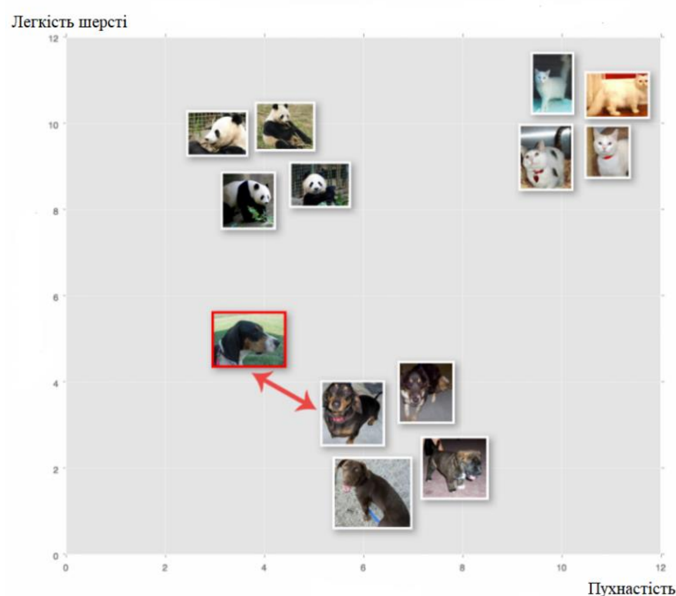

Рисунок 8.3 – Невідоме зображення (червоний колір) у наборі даних

Ми вставили невідоме зображення (виділене червоним кольором) у набір даних, а потім використали відстань між невідомою твариною та набором тварин для класифікації.

Також вставили «невідому тварину», яку намагаємося класифікувати, використовуючи лише одного сусіда (тобто k = 1). У цьому випадку найближчою твариною до вхідного зображення є точка даних собаки; таким чином наше вхідне зображення слід класифікувати як собаку.

Давайте спробуємо ще одну «невідому тварину» (рис.8.4). У трійці найкращих результатів ми виявили двох котів та одну панду. Оскільки категорія котів має найбільшу кількість голосів, ми класифікуємо наше вхідне зображення, як кішку, лише цього разу скористалися $k = 3$, а не просто $k = 1$. Оскільки є два зображення кота, які ближче до вхідного зображення, ніж одне зображення панди, позначимо це вхідне зображення як `cat`.

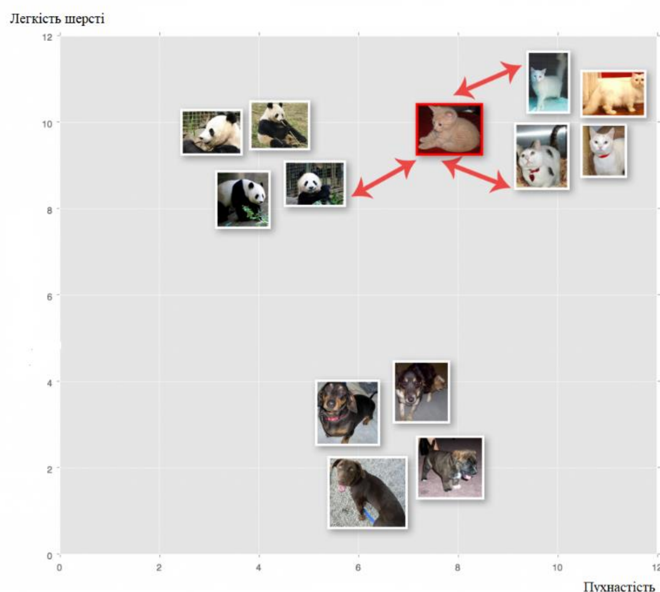

Рисунок 8.4 – Класифікація іншої тварини

Можемо продовжувати виконувати цей процес для різних значень k, але незалежно від того, наскільки великим або малим стає k, принцип залишається незмінним – категорія з найбільшою кількістю голосів у k найближчих навчальних балах виграє і використовується як позначка для точка вхідних даних.

Примітка. У разі пов'язаних класів алгоритм k-NN вибирає навмання одну з міток цих класів.

Гіперпараметри k-NN

Існує два чітких гіперпараметри, якими займаємось під час запуску алгоритму k-NN. Перше очевидне: значення *k*. Яке оптимальне значення *k*? Якщо воно занадто мале (наприклад, $k = 1$), тоді отримуємо ефективність, але стаємо сприйнятливими до шуму та неточних даних. Однак, якщо *k* занадто велике, тоді ризикуємо *надмірно згладити* результати нашої класифікації та збільшити упередженість.

Другим параметром, який слід враховувати, є фактична метрика відстані. Чи є найкращим вибором евклідова дистанція? А як щодо відстані Манхеттен?

Далі навчимо наш класифікатор k-NN на наборі даних Animals та оцінимо модель на нашому тестовому наборі. рекомендується пограти з різними значеннями *k* разом з різними показниками відстані, спостерігаючи, як змінюється продуктивність.

8.3. Реалізація k-NN

Метою цього розділу є тренування класифікатора k-NN щодо *інтенсивності пікселів* необроблених даних з набору даних Animals та використання його результатів для класифікації невідомих зображень тварин.

- **Крок 1. Збирання набору даних**. Набори даних Animals складаються всього з 3000 зображень: по 1000 зображень на клас собак, котів та панд, відповідно. Кожне зображення представлене у просторі RGB-кольору. Обробимо попередньо кожне зображення, змінивши його розмір до 32×32 пікселів. Беручи до уваги три RGB-канали, зменшення розміру зображення означає, що кожне зображення в наборі даних буде представлене $32 \times 32 \times 3 = 3072$ цілими числами.
- **Крок 2. Розділення набору даних**. Для нашого простого прикладу будемо використовувати розбиття даних на дві частини. Одна частина для

тренування, а друга – для тестування. Залишимо перевірку набору для налаштування гіперпараметрів як вправу для студентів.

- **Крок 3. Тренування класифікатора**. Наш класифікатор k-NN пройде навчання на основі інтенсивності вихідних пікселів зображень у наборі для тренування.
- **Крок 4. Оцінювання.** Після того, як наш класифікатор k-NN пройде навчання, зможемо оцінити його ефективність на тестовому наборі.

Давайте підемо далі. Для цього створимо новий файл, назвемо його `knn.py` і вставимо такий код:

```
# імпорт необхідних пакетів                                          #1
from sklearn.neighbors import KNeighborsClassifier                   #2
from sklearn.preprocessing import LabelEncoder                       #3
from sklearn.model_selection import train_test_split                 #4
from sklearn.metrics import classification_report                    #5
from pyimagesearch.preprocessing import SimplePreprocessor           #6
from pyimagesearch.datasets import SimpleDatasetLoader               #7
from imutils import paths                                            #8
import argparse                                                      #9
                                                                     #10
```

Рядки 2-9 імпортують необхідні пакети Python. Найважливіші з них, на які слід звернути увагу:

- Рядок 2: `KNeighborsClassifier` – це наша реалізація алгоритму k-NN, надана бібліотекою scikit-learn.
- Рядок 3: `LabelEncoder`, допоміжна утиліта для перетворення міток, представлених у вигляді рядків, у цілі числа, де є одне унікальне ціле число на мітку класу (звичайна практика при застосуванні машинного навчання).
- Рядок 4: Імпортуємо функцію `train_test_split`, яка є зручною функцією, що допомагає нам створювати розподіли даних на тренінгові та тестові.

- Рядок 5: Функція `classification_report` – це ще одна утиліта, яка використовується, щоб допомогти нам оцінити ефективність отриманого класифікатора та вивести добре відформатовану таблицю результатів на консоль.

Також можемо побачити наші реалізації SimplePreprocessor та SimpleDatasetLoader, імпортовані в рядках 6 та 7, відповідно.

Розберемо аргументи нашого командного рядка:

```
# будуємо аргумент синтаксично і аналізуємо аргумент            #11
ap = argparse.ArgumentParser()                                  #12
ap.add_argument("-d", "--dataset", required=True,               #13
   help="path to input dataset")                                #14
ap.add_argument("-k", "--neighbors", type=int, default=1,       #15
   help="# of nearest neighbors for classification")            #16
ap.add_argument("-j", "--jobs", type=int, default=-1,           #17
   help="# of jobs for k-NN distance (-1 uses all cores)")      #18
args = vars(ap.parse_args())                                    #19
                                                                #20
```

Наш сценарій вимагає одного аргументу командного рядка, за яким слідують два необов’язкові, кожен з яких розглянуто нижче:

- `--dataset`: Шлях до того місця на диску, де знаходиться набір вхідних зображень.
- `--neighbors`: Необов’язковий аргумент, кількість сусідів *k*, що застосовується при використанні алгоритму k-NN.
- `--jobs`: Необов’язковий аргумент, кількість одночасних завдань, які потрібно виконати при обчисленні відстані між точкою вхідних даних та тренувальним набором. Значення -1 використовуватиме всі доступні ядра процесора.

Тепер, коли аргументи командного рядка проаналізовані, можемо захопити шляхи до файлів зображень у наборі даних з подальшим завантаженням та попередньою обробкою (Крок 1 у конвеєрі класифікації):

```
# захоплюємо список зображень, які будемо описувати            #21
```

```
print("[INFO] loading images...")                                   #22
imagePaths = list(paths.list_images(args["dataset"]))               #23
                                                                    #24
# ініціалізуємо препроцесор зображення, завантажуємо набір          #25
# даних з диска,і переформуємо матрицю даних                        #26
sp = SimplePreprocessor(32, 32)                                     #27
sdl = SimpleDatasetLoader(preprocessors=[sp])                       #28
(data, labels) = sdl.load(imagePaths, verbose=500)                  #29
data = data.reshape((data.shape[0], 3072))                          #30
                                                                    #31
# показуємо інформацію про використання пам'яті зображеннями        #32
print("[INFO] features matrix: {:.1f}MB".format(                    #33
   data.nbytes / (1024 * 1024.0)))                                  #34
                                                                    #35
```

Рядок 23 захоплює шляхи до файлів всіх зображень у наборі даних. Потім ініціалізуємо `SimplePreprocessor`, який використовується для зміни розміру кожного зображення до 32 × 32 пікселів в рядку 27.

`SimpleDatasetLoader` ініціалізується в рядку 28, надаючи нашому екземпляру `SimplePreprocessor` аргумент (маючи на увазі, що `sp` буде застосовано до *кожного зображення* в наборі даних).

Виклик `.load` в рядку 29 завантажує наш фактичний набір зображень з диска. Цей метод повертає 2 кортежі даних (кожне зображення розміром до 32 × 32 пікселів) разом із мітками для кожного зображення.

Після завантаження зображень з диска масив даних NumPy має `.shape` (форму) (3000, 32, 32, 3). Це означає, що в наборі даних є 3000 зображень, кожне 32 × 32 пікселів з 3 каналами.

Однак, щоб застосувати алгоритм k-NN, треба «вирівняти» наші зображення з тривимірного подання до єдиного списку інтенсивності пікселів. Цього досягаємо тим, що у рядку 30 викликаємо метод `.reshape` для масиву даних NumPy, згладжуючи зображення 32 × 32 × 3 у масив з формою (3000, 3072). Фактичні дані зображення зовсім не змінилися - зображення просто представлені у вигляді списку з 3000 записів, кожен з розміром 3072 (32 × 32 × 3

= 3072). Щоб продемонструвати, скільки пам'яті потрібно для зберігання цих 3000 зображень у пам'яті, рядки 33 і 34 обчислюють кількість байтів, що займає масив, а потім перетворює число в мегабайти.

Далі побудуємо наші навчальні та тестові набори (Крок 2 у нашому конвеєрі):

```
# кодуємо мітки як цілі числа                                     #36
le = LabelEncoder()                                               #37
labels = le.fit_transform(labels)                                 #38
                                                                  #39
# розділяємо дані на навчальні та тестувальні, використовуючи   #40
# 75% даних для навчання, а решту 25% для тестування            #41
(trainX, testX, trainY, testY) = train_test_split(data, labels,#42
   test_size=0.25, random_state=42)                               #43
                                                                  #44
```

Рядки 37 і 38 перетворюють наші мітки (представлені у вигляді рядкових змінних) у цілі числа, де у нас є *одне унікальне ціле число* на клас. Це перетворення дозволяє зіставити клас *cat* з цілим числом 0, клас *dog* з цілим числом 1, а клас *panda* – з цілим числом 2. Багато алгоритмів машинного навчання припускають, що мітки класів кодуються цілими числами, тому важливо, щоб ми взяли за звичку виконувати цей крок.

Обчислення наших навчальних та тестових частин обробляється функцією `train_test_split` в рядках 42 і 43. Тут розділяємо дані та мітки на два унікальних набори: 75% даних для навчання та 25% для тестування.

Зазвичай використовується змінна *X* для посилання на набір даних, який містить точки даних, які будемо використовувати для навчання та тестування, тоді як *y* посилається на мітки класів. Тому використовуємо змінні `trainX` та `testX` для посилання на *приклади для навчання та тестування* відповідно. Змінні `trainY` та `testY` – це наші *мітки для навчання та тестування*. Надалі ви зустрінете ці загальні позначення на ЛР та в інших книгах, курсах та навчальних посібниках.

Нарешті, можемо створити наш класифікатор k-NN та оцінити його (Кроки 3 та 4 у конвеєрі класифікації зображень):

```
# навчання та оцінка класифікатора k-NN за інтенсивністю пікселів
print("[INFO] evaluating k-NN classifier...")                          #46
model = KNeighborsClassifier(n_neighbors=args["neighbors"],            #47
   n_jobs=args["jobs"])                                                #48
model.fit(trainX, trainY)                                              #49
print(classification_report(testY, model.predict(testX),              #50
   target_names=le.classes_))                                          #51
```

Рядки 47 і 48 ініціалізують клас `KNeighborsClassifier`. Виклик методу `.fit` в рядку 49 "готує" класифікатор, хоча фактичного "навчання" тут не відбувається – модель k-NN просто зберігає дані `trainX` та `trainY` внутрішньо, щоб можна було створювати прогнози при тестуванні шляхом обчислення відстані між вхідними даними та даними `trainX`.

Рядки 50 і 51 оцінюють наш класифікатор за допомогою функції `classification_report`. Тут нам потрібно надати мітки класу `testY`, *мітки передбачених класів* з нашої моделі та, за бажанням, імена міток класів (тобто, “dog”, “cat”, “panda”).

Результати роботи k-NN

Щоб запустити класифікатор k-NN, виконаємо таку команду:

```
python knn.py --dataset ../datasets/animals
```

Ми повинні побачити результати, подібні до наступних (рис.8.5):

```
[INFO] loading images...
[INFO] processed 500/3000
[INFO] processed 1000/3000
[INFO] processed 1500/3000
[INFO] processed 2000/3000
[INFO] processed 2500/3000
[INFO] processed 3000/3000
[INFO] features matrix: 8.8MB
[INFO] evaluating k-NN classifier...
              precision    recall  f1-score   support

        cats       0.37      0.52      0.43       239
        dogs       0.35      0.43      0.39       249
       panda       0.70      0.28      0.40       262

    accuracy                           0.41       750
   macro avg       0.47      0.41      0.41       750
weighted avg       0.48      0.41      0.40       750
```

Рисунок 8.5 – Результати роботи класифікатора k-NN

Звернемо увагу, що наша матриця функцій використовує лише 8,8 МБ пам’яті для 3000 зображень, кожне розміром 32 × 32 × 3 – цей набір даних можна легко зберігати в пам'яті на сучасних машинах без проблем.

Оцінюючи наш класифікатор, бачимо, що ми отримали 48% точності, що непогано для класифікатора, який взагалі не виконує жодного справжнього "навчання", враховуючи, що ймовірність випадкового вгадування правильної відповіді становить 1/3.

Однак цікаво перевірити точність кожної з міток класу. Клас "panda" був правильно класифікований в 70% випадків, ймовірно, через те, що панди в основному чорно-білі, і, отже, ці зображення лежать ближче одне до одного в нашому 3072-розмірному просторі.

Собаки та коти отримують значно нижчу точність класифікації – 35% та 37%, відповідно. Ці результати можна пояснити тим, що собаки та коти можуть мати дуже схожі відтінки шерсті, а колір їхньої шерсті не можна використовувати для їх розділення. Фоновий шум (наприклад, трава на задньому дворі, колір дивана, на якому відпочиває тварина тощо) також може «заплутати» алгоритм k-NN, оскільки він не в змозі вивчити будь-які закономірності, які розділяють ці види. Ця плутанина є одним із основних недоліків алгоритму k-NN: хоча він простий, але не може вчитися на даних.

Однією з головних переваг алгоритму k-NN є те, що його надзвичайно просто реалізувати та зрозуміти. Крім того, класифікатор не займає абсолютно ніякого часу, щоб тренуватися, оскільки все, що нам потрібно зробити, – це зберігати наші точки даних для подальшого обчислення відстаней до них та отримання остаточної класифікації.

Однак, ми платимо часом за цю простоту під час класифікації. Класифікація нової точки тестування вимагає порівняння з кожною окремою точкою даних з наших навчальних даних, яка масштабує $O(N)$, що робить обчислювально необмеженою роботу для великих наборів даних.

Можемо подолати ці витрати часу, використовуючи алгоритми приблизного найближчого сусіда (ANN). Тим не менш, у багатьох випадках

використання алгоритму k-NN варте докладених зусиль та невеликих втрат точності. Така поведінка, на відміну від більшості алгоритмів машинного навчання (та всіх нейронних мереж), де ми витрачаємо велику кількість часу заздалегідь навчаючи нашу модель для отримання високої точності, і, в свою чергу, дозволяє отримати *дуже швидкі* класифікації під час тестування.

Нарешті, алгоритм k-NN більше підходить для об’єктів з маловимірних параметрами (а зображення – не такі). Відстані у об'єктів з велико просторовими параметрами часто не інтуїтивні.

Важливо також зазначити, що алгоритм k-NN насправді нічого не «вчить» – алгоритм не може зробити себе розумнішим, якщо допускає помилки; це просто покладання на відстані в *n*-мірному просторі для виконання класифікації.

З огляду на ці мінуси, навіщо намагатися навіть вивчати алгоритм k-NN? Причина в тому, що алгоритм простий. Його легко зрозуміти. І найголовніше, це дає нам базову лінію, де ми можемо використовувати порівняння нейронних мереж та згорткових нейронних мереж під час подальшого навчання.

8.4. Завдання

1. Повторити розглянутий алгоритм k-NN, але для інших наборів зображень, які зможете завантажити з Інтернету.
2. Оформити протокол виконання ЛР та завантажити його у відповідну папку.

8.5. Зміст звіту

1. Сценарій на Python з коментарями.
2. Результати роботи класифікатора
2. Висновки за результатами виконання роботи.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

8.6. Контрольні питання

1. Яка основна перевага алгоритму k-NN?
2. Які параметри має метод `.reshape` NumPy?.

Лабораторна робота 9

ВИКОРИСТАННЯ МІКРОКОМП'ЮТЕРА RASPBERRY PI ДЛЯ МАШИННОГО НАВЧАННЯ

Мета роботи: Вивчити, як використовувати набір інструментів OpenVINO з OpenCV для прискореного глибокого навчання на RPi.

Зміст. Розглядаються первинні налаштування ПЗ, запуск на мікрокомп'ютері навченої моделі машинного навчання.

9.1. Загальні відомості

RPi має якісне апаратне забезпечення та спільноту підтримки, створену навколо пристрою.

Сьогодні для машинного навчання використовують дві модифікації: RPi 3 B+ і RPi 4 B. Для порівняння на рис.9.1 наведені параметри обох моделей.

Червоним кольором виділені основні відмінності між платами. Збільшення оперативної пам'яті та новий її тип в RPi 4 B є основним фактором використання даної плати для машинного навчання. Згідно експериментальних даних, RPi 4 B дозволяє прискорити оброблення зображень в 3 рази, в порівнянні з RPi 3 B+ (до 25 кадрів в секунду).

Тим не менш, для глибокого навчання поточне апаратне забезпечення RPi за своєю суттю обмежене в ресурсах, але ми отримаємо більше, ніж кілька FPS, бо використаємо Movidius NCS від Intel, який дозволяє швидше отримати результат за допомогою співпроцесора глибокого навчання, який підключається до роз'єму USB.

Перша версія API, яка постачалася разом із флешками, працювала добре та демонструвала потужність Myriad, але залишала бажати кращого. Потім було випущено Movidius APIv2. Він був легшим/надійнішим, ніж APIv1, але також мав достатню частку проблем.

Параметри	Raspberry Pi 4 B	Raspberry Pi 3 B+
Дата випуску	24 червня 2019	14 березня 2018
Тип SoC (Процесор)	Broadcom BCM2711	Broadcom BCM2837B0
Тип ядра	Cortex-A72 64-bit (ARMv8)	Cortex-A53 64-bit (ARMv8)
Кількість ядер	Quad-Core	
GPU	VideoCore VI	VideoCore IV
Мультимедіа	H.265 decode (4Kp60) H.264 decode (1080p60) H.264 encode (1080p30) OpenGL ES 1.1, 2.0, 3.0 Graphics	H.264, MPEG-4 decode (1080p30) H.264 encode (1080p30) OpenGL ES 1.1, 2.0 Graphics
Тактова частота CPU	1.5 GHz	1.4 GHz
Постійна пам'ять	microSD	
Оперативна пам'ять	LPDDR4 1GB, 2GB або 4GB	LPDDR2 1GB
Ethernet	True Gigabit Ethernet	Gigabit over USB 2.0 (Max 300Mbps)
Порт USB	2 x USB 3.0 + 2 x USB 2.0	4 x USB 2.0
HDMI	2 x micro HDMI support Dual Display	1 x full size HDMI
WiFi	802.11 b/g/n/ac (2.4 ГГц + 5 ГГц)	
Bluetooth	5.0 + BLE	4.2 + BLE
Антена	PCB Antenna	
GPIO	40 виводів	
Операційна система	Raspbian (> 24 June 2019)	Raspbian (> March 2018)
Розміри	85 мм х 56 мм	
Вхід живлення	5V via USB Type C (upto 3A) 5V via GPIO header (upto 3A) Power over Ethernet, вимагає PoE HAT	5V via USB Micro B (upto 2.5A) 5V via GPIO header (upto 3A) Power over Ethernet, вимагає PoE HAT

Рисунок 9.1 – Порівняння параметрів Raspberry Pi 4 B і Raspberry Pi 3 B+

«Флешка» USB 3 від Intel (рис.9.2), яка реалізує нейронна мережу для ПК та таких окремих плат, як RPi. Надзвичайно прискорює тензорну арифметику.

В основі модуля візуальний процесор Intel® Movidius™ Myriad™ X, який має 16 векторних 128 бітних LIW-ядер SHAVE (Streaming Hybrid Architecture Vector Engine). Myriad X має апаратну підтримку кодування 4k-відео зі швидкістю до 60 кадрів за сек. Максимальна теоретична продуктивність Movidius Myriad X складає 4 Терафлопси, при цьому енергоспоживання SoC не більше 1 Вт.

Рисунок 9.2 – Intel Neural Compute Stick 2

Підтримуються інфраструктури: TensorFlow і Caffe. Сумісні операційні системи: Ubuntu 16.04.3 LTS (64-розрядна), CentOS 7.4 (64-розрядна) і Windows 10 (64-розрядна). Також Intel Neural Compute Stick 2 підтримує Open Visual Inference & Neural Network Optimization (OpenVINO). Важливо, що можна об’єднати кілька «флешок» Intel® NCS 2 в одну платформу для масштабування продуктивності.

В таблиці 9.1 показані деякі моделі нейронних мереж та порівняльні швидкості оброблення зображення в кадрах за секунду (FPS). Для навчання найбільш ефективним рішенням за доступною ціною буде комбінація мікрокомп’ютера Raspberry Pi 4 з Intel Neural Compute Stick 2 та фреймворком TF.

Таблиця 9.1 – Порівняння ефективності мікрокомп’ютерних платформ

Model	Framework	Raspberry Pi (use TF-Lite)	Raspberry Pi (our NCNN)	Raspberry Pi Intel Neural Stick 2	Raspberry Pi Google Coral USB	JeVois	Jetson Nano	Google Coral
EfficientNet-B0 (224x224)	TensorFlow	14.6 FPS (Pi 3) 25.8 FPS (Pi 4)	-	95 FPS (Pi 3) 180 FPS (Pi 4)	105 FPS (Pi 3) 200 FPS (Pi 4)	-	216 FPS	200 FPS
ResNet-50 (244x244)	TensorFlow	2.4 FPS (Pi 3) 4.3 FPS (Pi 4)	1.7 FPS (Pi 3) 3 FPS (Pi 4)	16 FPS (Pi 3) 60 FPS (Pi 4)	10 FPS (Pi 3) 18.8 FPS (Pi 4)	-	36 FPS	18.8 FPS
MobileNet-v2 (300x300)	TensorFlow	4.4 FPS (Pi 3) 8 FPS (Pi 4)	8 FPS (Pi 3) 8.9 FPS (Pi 4)	30 FPS (Pi 3)	46 FPS (Pi 3)	30 FPS	64 FPS	130 FPS
SSD Mobilenet-V2 (300-300)	TensorFlow	2.6 FPS (Pi 3) 4.7 FPS (Pi 4)	3.7 FPS (Pi 3) 5.8 FPS (Pi 4)	11 FPS (Pi 3) 41 FPS (Pi 4)	17 FPS (Pi 3) 55 FPS (Pi 4)	-	39 FPS	48 FPS
Binary model (300x300)	XNOR	6.8 FPS (Pi 3) 12.5 FPS (Pi 4)	-	-	-	-	-	-
Inception V4 (299x299)	PyTorch	-	-	-	3 FPS (Pi 3)	-	11 FPS	9 FPS
Tiny YOLO V3 (416x416)	Darknet	0.5 FPS (Pi 3) 1 FPS (Pi 4)	1.1 FPS (Pi 3) 1.9 FPS (Pi 4)	-	-	2.2 FPS	25 FPS	-
OpenPose (256x256)	Caffe	-	-	5 FPS (Pi 3)	-	-	14 FPS	-
Super Resolution (481x321)	PyTorch	-	-	0.6 FPS (Pi 3)	-	-	15 FPS	-
VGG-19 (224x224)	MXNet	0.5 FPS (Pi 3) 1 FPS (Pi 4)	-	5 FPS	-	-	10 FPS	-
Unet (1x512x512)	Caffe	-	-	5 FPS	-	-	18 FPS	-

Тепер працювати з Movidius NCS, особливо з OpenCV, стало легше, ніж будь-коли. Перехід Intel на підтримку апаратного забезпечення Movidius із програмним забезпеченням OpenVINO зробило Movidius надзвичайно простим у використанні – просто треба встановити цільовий процесор (один виклик функції) і дозволити оптимізованому для OpenVINO OpenCV виконувати решту.

Ми розглянемо три основні теми:

1. Дізнаємося, що таке OpenVINO і наскільки це бажана зміна парадигми для RPi.
2. Вивчимо, як встановити OpenCV і OpenVINO на наш RPi.
3. Розробимо сценарій виявлення об’єктів у реальному часі за допомогою OpenVINO, OpenCV і Movidius NCS.

Що таке OpenVINO?

Набір інструментів Intel OpenVINO оптимізує наші програми комп’ютерного зору для обладнання Intel, наприклад Movidius NCS. Значне прискорення виявлення об’єктів у реальному часі за допомогою OpenVINO та OpenCV за допомогою RPi та Movidius NCS.

OpenVINO підтримує процесори Intel, графічні процесори, FPGA та VPU. Бібліотеки глибокого навчання, на які ми звикли покладатися, такі як TensorFlow тощо, підтримуються OpenVINO.

Intel OpenVINO Toolkit підтримує процесори Intel, графічні процесори, FPGA та VPU. TensorFlow, Caffe, mxnet і модуль DNN OpenCV оптимізовані та прискорені для обладнання Intel. Лінія блоків обробки зору Movidius (VPU) підтримується OpenVINO та добре поєднується з RPi.

Intel навіть оптимізувала модуль OpenCV DNN для підтримки апаратного забезпечення глибокого навчання. Насправді багато новіших розумних камер використовують апаратне забезпечення Intel разом із набором інструментів OpenVINO. OpenVINO – це периферійні обчислення та Інтернет речей у найкращому вигляді – він дозволяє пристроям з обмеженими ресурсами, таким як RPi, працювати з співпроцесором Movidius для виконання глибокого навчання зі швидкостями, корисними для реальних програм.

Далі встановимо OpenVINO на RPi, щоб його можна було використовувати з Movidius VPU.

9.2. Встановлення OpenVINO OpenCV на Raspberry Pi

Розглянемо підготовку та всі кроки, необхідні для встановлення оптимізованого OpenCV і OpenVINO на RPi.

Апаратне забезпечення, припущення та передумови

Припускаємо, що у нас є таке обладнання:

• Raspberry 4B або 3B+ (під керуванням Raspbian Buster)

• Movidius NCS 2 (або Movidius NCS 1)

• PiCamera V2 (або веб-камера USB)

• Картка microSD на 32 ГБ з оновленою ОС

• Екран HDMI + клавіатура/миша (принаймні для початкової конфігурації WiFi)

• Джерело живлення 5 ВЮ бажано не менше, ніж на 2,5 А.

Коли будемо готові, вставляємо картку microSD у наш RPi та завантажуємо його. Вводимо свої облікові дані WiFi та вмикаємо SSH, VNC та інтерфейс камери.

Нам знадобиться одне з наступного:

• Фізичний доступ до нашого RPi, щоб ми могли відкривати термінал і виконувати команди.

• Віддалений доступ через SSH або VNC.

Більша частина цієї ЛР виконується через SSH, але якщо є доступ до терміналу, то можна скористатися ним, або VNC.

Крок 1. **Розширяємо файлову систему на RPi**

Щоб почати роботу з OpenVINO, запустимо RPi та відкриємо з'єднання SSH (або скористайтеся робочим столом ОС за допомогою клавіатури + миші та запустимо термінал).

Завжди рекомендується спочатку перевірити, чи наша файлова система використовує весь доступний простір на карті microSD.

Щоб перевірити використання дискового простору, виконаємо

```
df -h
```

у нашому терміналі та перевіримо результат:

```
Filesystem Size Used Avail Use% Mounted on
/dev/root 30G 4.2G 24G 15% /
devtmpfs 434M 0 434M 0% /dev
tmpfs 438M 0 438M 0% /dev/shm
tmpfs 438M 12M 427M 3% /run
tmpfs 5.0M 4.0K 5.0M 1% /run/lock
tmpfs 438M 0 438M 0% /sys/fs/cgroup
/dev/mmcblk0p1 42M 21M 21M 51% /boot
tmpfs 88M 0 88M 0% /run/user/1000
```

В даному випадку, файлова система ОС була автоматично розширена, щоб включити всі 32 ГБ карти micro-SD. Це позначається тим фактом, що розмір становить 30 ГБ (майже 32 ГБ). Якщо бачимо, що використовується не весь обсяг карти пам'яті, то відкриємо конфігурацію Raspberry Pi у нашому терміналі:

```
sudo raspi-config
```

А потім виберемо пункт меню «Advanced Options» (рис.9.3):

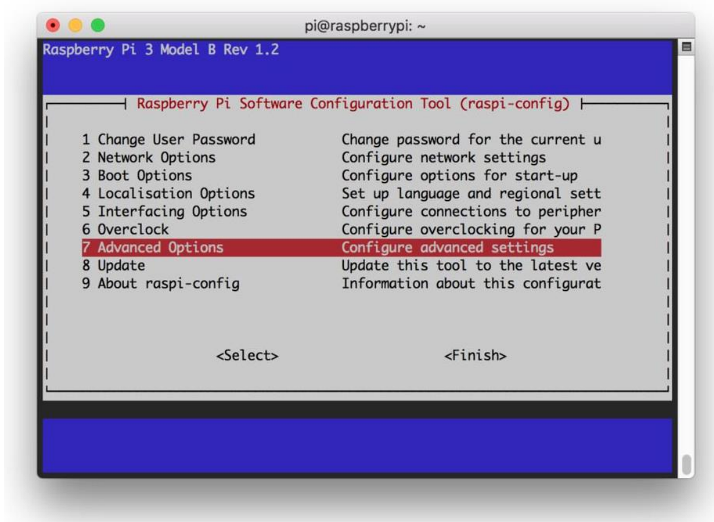

Рисунок 9.3 – Вибір «Advanced Options»

Вибір «Advanced Options» у меню `raspi-config` для розширення файлової системи Raspbian на нашому RPi є важливим перед встановленням OpenVINO та OpenCV. Далі ми фактично розширимо файлову систему (рис.9.4):

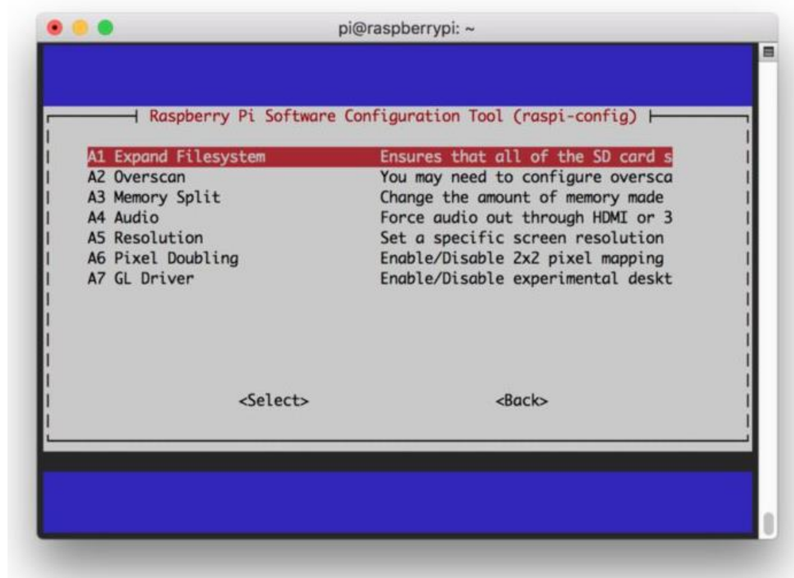

Рисунок 9.4 – Розширення файлової системи

Меню RPi «Expand Filesystem» дозволяє нам скористатися всією пам'яттю нашої флеш-карти. Це надасть простір, необхідний для встановлення OpenVINO, OpenCV та інших пакетів. Коли з'явиться відповідний запит, виберемо перший варіант «***A1. Expand File System***», натиснемо Enter на клавіатурі, стрілку вниз до кнопки <**Finish**>, а потім перезавантажимо RPi – це нам буде запропоноване. Крім того, ми можете перезавантажитися з терміналу:

```
sudo reboot
```

Переконаємось, виконавши знову команду

```
df -h
```

що файлова система розширена.

Крок 2. Звільняємо місце на Raspberry Pi

Один із простих способів отримати більше місця на Raspberry Pi – видалити LibreOffice і Wolfram Engine, щоб звільнити місце на Pi:

```
sudo apt purge wolfram-engine
sudo apt purge libreoffice*
sudo apt clean
```

```
sudo apt autoremove
```

Після видалення Wolfram Engine і LibreOffice можемо повернути майже 1 ГБ!

Крок 3. Встановлюємо залежності OpenVINO + OpenCV на наш RPi

Цей крок показує деякі залежності, які встановлюємо в кожній системі OpenCV. Хоча незабаром побачимо, що OpenVINO вже скомпільовано, рекомендується все одно встановити ці пакети на випадок, якщо нам у майбутньому доведеться скомпілювати OpenCV з нуля.

Давайте оновимо нашу систему:

```
sudo apt update && sudo apt upgrade -y
```

А потім встановимо інструменти розробника, включаючи CMake:

```
sudo apt install build-essential cmake unzip pkg-config
```

Настав час інсталювати добірку бібліотек зображень і відео – вони є ключовими для роботи з файлами зображень і відео:

```
sudo apt install libjpeg-dev libpng-dev libtiff-dev
sudo apt install libavcodec-dev libavformat-dev libswscale-dev
libv4l-dev
sudo apt install libxvidcore-dev libx264-dev
```

Звідти давайте встановимо GTK, наш GUI бекенд:

```
sudo apt install libgtk-3-dev
```

А тепер давайте встановимо пакет, який може допомогти зменшити попередження GTK:

```
sudo apt install libcanberra-gtk*
```

Зірочка означає, що ми візьмемо GTK для ARM. Це потрібно. Тепер нам потрібні два пакети, які містять числову оптимізацію для OpenCV:

```
sudo apt install libatlas-base-dev gfortran
```

І, нарешті, давайте встановимо заголовки розробки Python 3:

```
sudo apt install python3-dev
```

Після того, як ми встановили всі ці передумови, можемо переходити до наступного кроку.

Крок 4. Завантажуємо та розпаковуємо OpenVINO для Raspberry Pi

Завантажимо та інсталюємо набір інструментів OpenVINO для програм комп'ютерного бачення RPi та Movidius. Наші інструкції зі встановлення в основному базуються на посібнику Intel Raspberry Pi OpenVINO. Також будемо використовувати віртуальні середовища.

Наш наступний крок – завантажити OpenVINO.

Давайте перейдемо до нашої домашньої папки та створимо новий каталог:

```
cd ~
```

Звідти перейдемо і завантажимо OpenVINO Toolkit через `wget`:

```
sudo wget
https://storage.openvinotoolkit.org/repositories/openvino/packages
/2022.3/linux/l_openvino_toolkit_debian9_2022.3.0.9052.9752fafe8eb
_arm64.tgz -O openvino_2022.3.0.tgz
```

Після того, як ми успішно завантажили набір інструментів OpenVINO, можемо розархівувати його за допомогою такої команди:

```
sudo tar -xf openvino_2022.3.0.tgz
sudo mv l_openvino_toolkit_debian9_2022.3.0.9052.9752fafe8eb_arm64
/opt/intel/openvino_2022.3.0
```

Крок 5. Налаштовуємо OpenVINO на Raspberry Pi

Використаємо `nano`, щоб редагувати наш `~/.bashrc`. Ми додаємо рядок для завантаження OpenVINO `setupvars.sh` щоразу, коли викликаємо термінал RPi. Відкриваємо файл:

```
sudo nano ~/.bashrc
```

Прокрутимо вниз і додамо такі рядки:

```
# OpenVINO
source ~/openvino/bin/setupvars.sh
```

Збережемо і вийдемо із текстового редактора `nano`.

Тоді продовжимо і `source` наш `~/.bashrc` файл:

```
source ~/.bashrc
```

Крок 6. Налаштуємо правила USB для Movidius NCS і OpenVINO на RPi

OpenVINO вимагає встановлення власних правил USB. Це досить просто, тож почнемо. Спочатку введемо таку команду, щоб додати поточного користувача до групи «користувачів» Raspbian:

```
sudo usermod -a -G users "$(whoami)"
```

Потім вийдемо із системи та увійдемо знову. Якщо використовуємо SSH, можемо вводити

```
exit
```

а потім повторно встановити підключення SSH. Перезавантаження також доступне за допомогою

```
sudo reboot now
```

Повернувшись до свого терміналу, запустимо такий сценарій, щоб установити правила USB:

```
cd ~
sh openvino/install_dependencies/install_NCS_udev_rules.sh
```

Крок 7. Створимо віртуальне середовище OpenVINO на RPi

Встановимо pip, менеджер пакетів Python. Для цього просто введемо наступне у своєму терміналі:

```
wget https://bootstrap.pypa.io/get-pip.py
sudo python3 get-pip.py
```

Будемо використовувати віртуальні середовища для розробки на Python за допомогою OpenCV і OpenVINO. Віртуальні середовища дозволять нам ізольовано запускати незалежні, ізольовані середовища Python у системі. Сьогодні налаштуємо лише одне середовище, але можемо легко створити середовище для кожного проєкту. Давайте встановимо тепер `virtualenv` і `virtualenvwrapper` – вони дозволяють віртуальні середовища Python:

```
sudo pip install virtualenv virtualenvwrapper
sudo rm -rf ~/get-pip.py ~/.cache/pip
```

Щоб завершити встановлення цих інструментів, нам потрібно оновити наш `~/.bashrc` знову:

```
sudo nano ~/.bashrc
```

Потім додаємо такі рядки:

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
VIRTUALENVWRAPPER_ENV_BIN_DIR=bin
```

Наш профіль RPi `~/.bashrc` оновлено для підтримки OpenVINO та `virtualenvwrapper`. Тепер зможемо створити віртуальне середовище для пакетів Python. Крім того, можемо додати рядки безпосередньо за допомогою команд `bash`:

```
echo -e "\n# virtualenv and virtualenvwrapper" >> ~/.bashrc
echo "export WORKON_HOME=$HOME/.virtualenvs" >> ~/.bashrc
echo "export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3" >>
~/.bashrc
echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.bashrc
echo "VIRTUALENVWRAPPER_ENV_BIN_DIR=bin" >> ~/.bashrc
```

Далі джерело профілю `~/.bashrc`:

```
source ~/.bashrc
```

Тепер створимо віртуальне середовище для зберігання OpenVINO, OpenCV і пов’язаних пакетів:

```
mkvirtualenv openvino -p python3
```

Ця команда просто створює віртуальне середовище Python 3 під назвою openvino.

Крок 8. Встановимо пакети у середовище OpenVINO

Встановимо кілька пакетів, необхідних для нашого демонстраційного сценарію:

```
workon openvino
pip install numpy
pip install "picamera[array]"
pip install imutils
```

Тепер, коли ми встановили ці пакунки в віртуальному середовищі openvino, вони доступні лише в навколишньому середовищі openvino. Це наша відокремлена область для роботи над проектами OpenVINO (тут ми використовуємо віртуальні середовища Python, тому не ризикуємо зіпсувати нашу системну інсталяцію Python).

Додаткові пакети для Caffe, TensorFlow і mxnet можна встановити через файл requirements.txt за допомогою pip. Ми можете прочитати більше про це за цим посиланням на документацію Intel. Це не обов’язково для даної ЛР.

Крок 9. Перевіряємо інсталяцію OpenVINO на Raspberry Pi

Зробимо швидкий тест на працездатність, щоб побачити, чи готовий OpenCV до роботи, перш ніж спробувати приклад OpenVINO. Відкриємо термінал і виконаємо такі дії:

```
workon openvino
source ~/openvino/bin/setupvars.sh
python
>>> import cv2
>>> cv2.__version__
'4.2.0-openvino'
>>> exit()
```

Перша команда активує наше віртуальне середовище OpenVINO. Друга команда налаштовує Movidius NCS з OpenVINO і є дуже важливою. Звідти ми запускаємо двійковий файл Python 3 у середовищі та імпортуємо OpenCV.

Версія OpenCV вказує, що це оптимізована інсталяція OpenVINO!

Рекомендується створити сценарій оболонки для запуску середовища OpenVINO.

Відкриємо новий файл під назвою `start_openvino.sh` і помістимо його у свій ~/ каталог. Вставимо такі рядки:

```
#!/bin/bash
echo "Starting Python 3.7 with OpenCV-OpenVINO 4.2.0 bindings..."
source ~/openvino/bin/setupvars.sh
workon openvino
```

Збережемо і закриємо файл. З цього моменту ми можемо активувати своє середовище OpenVINO за допомогою однієї простої команди (на відміну від двох команд, як у попередньому кроці):

```
source ~/start_openvino.sh
Starting Python 3.7 with OpenCV-OpenVINO 4.2.0 bindings...
```

9.3. Виявлення об’єктів у реальному часі за допомогою RPi та OpenVINO

Встановлення OpenVINO було досить простим і навіть не вимагало компіляції OpenCV. Тепер давайте запустимо роботу Movidius Neural Compute Stick за допомогою OpenVINO. Для порівняння ми запустимо детектор об’єктів MobileNet SSD з Movidius і без нього, щоб порівняти FPS. Ми порівняємо значення з попередніми результатами використання Movidius NCS APIv1.

Структура проєкту

Після того, як ми розвернули zip, можемо використати команду `tree` для перевірки каталогу проєкту:

```
tree

├── MobileNetSSD_deploy.caffemodel
├── MobileNetSSD_deploy.prototxt
├── openvino_real_time_object_detection.py
└── real_time_object_detection.py
0 directories, 3 files
```

Наші файли детектора об’єктів MobileNet SSD містять файли .caffemodel і .prototxt.txt. Вони попередньо підготовлені (ми не будемо навчати MobileNet SSD сьогодні). Ми збираємося переглянути openvino_real_time_object_detection.py і порівняти його з вихідним сценарієм виявлення об’єктів у реальному часі (real_time_object_detection.py).

Щоб продемонструвати потужність OpenVINO на Raspberry Pi з Movidius, ми збираємося виконати виявлення об’єктів глибокого навчання в реальному часі.

Співпроцесор Movidius/Myriad виконає фактичне глибоке навчання, зменшуючи навантаження на ЦП RPi. Ми й надалі використовуватимемо процесор RPi для обробки результатів і вказування Movidius, що робити, але ми залишаємо за собою висновок глибокого навчання для Myriad, оскільки його апаратне забезпечення оптимізовано та розроблено для глибокого навчання.

OpenVINO з OpenCV дозволяє нам вказати процесор для результату під час використання модуля OpenCV «DNN».

Насправді, для використання процесора Movidius NCS Myriad потрібен лише один рядок коду (як правило). Пропустимо деталі й натомість продемонструємо потужність OpenVINO. Сьогодні ми додаємо лише один рядок коду, який виконує обчислення (і коментар + порожній рядок). Таким чином, нова загальна кількість рядків коду становить 103 без використання попереднього складного Movidius APIv1 (215 рядків коду).

Давайте дізнаємося про зміни, необхідні для адаптації API OpenVINO до OpenCV і Movidius. Відкриємо файл з назвою openvino_real_time_object_detection.py і вставимо наступні рядки:

```
# імпорт необхідних пакетів                                          #1
from imutils.video import VideoStream                                 #2
from imutils.video import FPS                                         #3
import numpy as np                                                    #4
import argparse                                                       #5
```

```
import imutils                                                      #6
import time                                                         #7
import cv2                                                          #8
                                                                    #9
# побудувати розбір аргументів і розібрати аргументи               #10
ap = argparse.ArgumentParser()                                      #11
ap.add_argument("-p", "--prototxt", required=True,                  #12
   help="path to Caffe 'deploy' prototxt file")                     #13
ap.add_argument("-m", "--model", required=True,                     #14
   help="path to Caffe pre-trained model")                          #15
ap.add_argument("-c", "--confidence", type=float, default=0.2,      #16
   help="minimum probability to filter weak detections")            #17
ap.add_argument("-u", "--movidius", type=bool, default=0,           #18
   help="boolean indicating if the Movidius should be used")        #19
args = vars(ap.parse_args())                                        #20
                                                                    #21
# ініціалізувати список міток класів, які MobileNet SSD виявляє,
# створити набір кольорів обмежувальної рамки для кожного класу#23
CLASSES = ["background", "aeroplane", "bicycle", "bird","boat",#24
   "bottle", "bus", "car", "cat", "chair", "cow","diningtable",#25
   "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
   "sofa", "train", "tvmonitor"]                                    #27
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))          #28
                                                                    #29
# завантажити нашу серіалізовану модель з диска                     #30
print("[INFO] loading model...")                                    #31
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"]) #32
                                                                    #33
# вказати цільовий пристрій як процесор Myriad на NCS               #34
net.setPreferableTarget(cv2.dnn.DNN_TARGET_MYRIAD)                  #35
                                                                    #36
# ініціалізувати відеопотік, дозволити датчику камери               #37
# розігрітися та ініціалізувати лічильник FPS                       #38
print("[INFO] starting video stream...")                            #39
vs = VideoStream(usePiCamera=True).start()                          #40
time.sleep(2.0)                                                     #41
```

```
fps = FPS().start()                                              #42
                                                                 #43
# цикл кадрів з відеопотоку                                      #44
while True:                                                      #45
    # взяти кадр із потокового відео, змінити його розмір,       #46
    # щоб мати максимальну ширину 400 пікселів                   #47
    frame = vs.read()                                            #48
    frame = imutils.resize(frame, width=400)                     #49
                                                                 #50
    # взяти розміри кадру та перетворити його на blob            #51
    (h, w) = frame.shape[:2]                                     #52
    blob = cv2.dnn.blobFromImage(frame, 0.007843, (300, 300),
127.5)                                                           #53
                                                                 #54
    # передати blob через мережу та отримати                     #55
    # виявлення та передбачення                                  #56
    net.setInput(blob)                                           #57
    detections = net.forward()                                   #58
                                                                 #59
    # цикл над виявленнями                                       #60
    for i in np.arange(0, detections.shape[2]):                  #61
        # отримати впевненість (тобто ймовірність),              #62
        # пов’язану з прогнозом                                  #63
        confidence = detections[0, 0, i, 2]                      #64
                                                                 #65
        # відфільтрувати слабкі виявлення, переконавшись, що     #66
        # `надійність` є більшою за мінімальну впевненість       #67
        if confidence > args["confidence"]:                      #68
            # видобути індекс мітки класу з`detections`,         #69
            # потім обчислити (x, y)-координати                  #70
            # обмежувальної рамки для об’єкта                    #71
            idx = int(detections[0, 0, i, 1])                    #72
            box = detections[0, 0, i, 3:7] * np.array([w, h, w,
h])                                                              #73
            (startX, startY, endX, endY) = box.astype("int")     #74
                                                                 #75
```

```
        # намалювати передбачення на кадрі                    #76
        label = "{}: {:.2f}%".format(CLASSES[idx],            #77
            confidence * 100)                                 #78
        cv2.rectangle(frame, (startX, startY), (endX, endY),
            COLORS[idx], 2)                                   #80
        y = startY - 15 if startY - 15 > 15 else startY + 15
        cv2.putText(frame, label, (startX, y),                #82
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)    #83
                                                              #84
    # показати вихідний кадр                                  #85
    cv2.imshow("Frame", frame)                                #86
    key = cv2.waitKey(1) & 0xFF                               #87
                                                              #88
    # якщо була натиснута клавіша `q`, вийти з циклу          #89
    if key == ord("q"):                                       #90
        break                                                 #91
                                                              #92
    # оновити лічильник FPS                                   #93
    fps.update()                                              #94
                                                              #95
# зупинити таймер і відобразити інформацію FPS                #96
fps.stop()                                                    #97
print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))    #98
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))         #99
                                                             #100
# зробити трохи прибирання                                   #101
cv2.destroyAllWindows()                                      #102
vs.stop()                                                    #103
```

Рядки 33-35 нові. Але лише один із цих рядків цікавий.

У рядку 35 ми повідомляємо модулю OpenCV DNN використовувати співпроцесор Myriad за допомогою `net.setPreferableTarget(cv2.dnn.DNN_TARGET_MYRIAD)`.

Процесор Myriad вбудовано в Movidius Neural Compute Stick. Ми можемо використовувати цей самий метод, якщо використовуємо OpenVINO + OpenCV

на пристрої з вбудованим чіпом Myriad (тобто без громіздкого USB-накопичувача).

Неймовірно, що для попереднього Movidius API потрібно було 215 рядків набагато складнішого коду, порівняно зі 103 рядками набагато легшого коду за допомогою OpenVINO. Ці відмінності в кількості рядків говорять самі за себе з точки зору зниження складності, часу та економії витрат на розробку, але які фактичні результати? Наскільки швидкий OpenVINO з Movidius? Дізнаємося далі.

Щоб запустити наш сценарій, спершу потрібно отримати «Завантаження», пов'язані з цією ЛР.

Розпакуємо zip і перейдемо до каталогу.

Активуємо своє віртуальне середовище за допомогою рекомендованого вище методу:

```
source ~/start_openvino.sh
Starting Python 3.7 with OpenCV-OpenVINO 4.2.0 bindings...
```

Щоб виконати виявлення об'єктів за допомогою OpenVINO, просто виконаємо таку команду:

```
python openvino_real_time_object_detection.py
--prototxt MobileNetSSD_deploy.prototxt \
--model MobileNetSSD_deploy.caffemodel
[INFO] loading model...
[INFO] starting video stream...
[INFO] elasped time: 55.35
[INFO] approx. FPS: 8.31
```

Як бачимо, ми досягаємо 8,31 FPS протягом приблизно однієї хвилини. На рис. 9.5 зібрані додаткові еталонні порівняння детектора об'єктів глибокого навчання MobileNet SSD за допомогою OpenVINO з Movidius Neural Compute Stick.за допомогою MobileNet SSD.

Real-time object detection with **OpenVINO** and Movidius

	Pi 3B+, CPU-only	Pi 3B, NCS1, **APIv1**	Pi 3B+, NCS1, **OpenVINO**	Pi 3B+, NCS2, **OpenVINO**
MobileNet SSD (display on)	0.63 FPS	3.37 FPS	5.88 FPS	**8.31 FPS**
MobileNet SSD (display off)	0.64 FPS	4.39 FPS	6.08 FPS	**8.37 FPS**

Рисунок 9.5 – Еталонні порівняння ефективності

OpenVINO та Movidius NCS 2 дуже швидкі, це величезне прискорення порівняно з попередніми версіями. Дивно, що результати у 8 разів кращі, ніж при використанні лише ЦП RPi 3B+ (без співпроцесора Movidius). Дві крайні праві колонки (світло-блакитні колонки 3 і 4) показують порівняння OpenVINO між NCS1 і NCS2. Зауважимо, що статистика у 2-му стовпці стосується RPi 3B (а не 3B+). Експеримент був зроблений в 2018 році з використанням попереднього API та попереднього обладнання RPi.

9.4. Завдання

1. Ознайомитися з процесом встановлення та налаштування програмного забезпечення (ПЗ) для використання Neural Compute Stick 2 з RPi 4.
2. Знайти на сайті розробників Neural Compute Stick інформацію про оновлення ПЗ.
3. В протоколі коротко описати послідовність встановлення та налаштування оновленого ПЗ.
4. Протокол завантажити у відповідну папку на сайті Moodle.

9.5. Зміст звіту

1. Сценарій на Python для кожного завдання.
2. Скріншоти результатів роботи сценарію.
3. Висновки за результатами виконання роботи.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

9.6. Контрольні питання

1. Наскільки прискорює оброблення зображень RPi 4 B в порівнянні з RPi 3 B+?
2. Яка максимальна теоретична продуктивність Movidius Myriad X?
3. Які функції реалізує OpenVINO?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Базова література:

1. Могильний С.Б. Мікрокомп'ютер Raspberry Pi – інструмент дослідника. – К.: Талком, 2014. – 340 с. (Електронна версія http://isearch.kiev.ua/uk/book/1850-microcomputer-raspberry-pi-tool-researcher)
2. Могильний С.Б. Вбудовані системи програмно-апаратних комплексів обробки інформації: Лабораторний практикум [Електронний ресурс]: навчальний посібник для здобувачів ступеня бакалавра за спеціальністю 172 «Електронні комунікації та радіотехніка» / С.Б.Могильний; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 3,74 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2023. – 121 с. – Назва з екрана.
3. Яковенко А.В. Основи програмування. Python. Частина 1 [Електронний ресурс]: підручник для студ. спеціальності 122 «Комп'ютерні науки». – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с. – Назва з екрана.
4. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М., 2020. 180 с.
5. Копей В.Б. Мова програмування Python для інженерів і науковців: Навчальний посібник. Івано-Франківськ: ІФНТУНГ, 2019. 274с.
6. Креневич А.П. Python у прикладах і задачах. Частина 1. Структурне програмування. Навчальний посібник із дисципліни "Інформатика та програмування" – К.: ВПЦ "Київський Університет", 2017. – 206 с.

Додаткова література:

1. Simon Monk. Raspberry Pi Cookbook. – O'REILLY, 2016. – 510 p.
2. Stewart Watkiss. Learn Electronics with Raspberry Pi. – Apress, 2016. – 300 p.
3. Alex Bradbury, Ben Everard. Learning Python with Raspberry Pi. – Wiley, 2013. – 288 p.
4. Tim Cox. Raspberry Pi Cookbook for Python Programmers. – Packt Publishing, 2014. – 402 p.

5. Adrian Rosebrock. Your First Image Classifier: Using k-NN to Classify Images. [Електронний ресурс] https://pyimagesearch.com/2021/04/17/your-first-image-classifier-using-k-nn-to-classify-images/. – Назва з екрана.
6. Adrian Rosebrock. OpenVINO, OpenCV, and Movidius NCS on the Raspberry Pi. [Електронний ресурс] https://pyimagesearch.com/2019/04/08/openvino-opencv-and-movidius-ncs-on-the-raspberry-pi/. – Назва з екрана.
7. Adrian Rosebrock. Keras Tutorial: How to get started with Keras, Deep Learning, and Python. [Електронний ресурс] https://pyimagesearch.com/2018/09/10/keras-tutorial-how-to-get-started-with-keras-deep-learning-and-python/. – Назва з екрана.
8. John C. Shovic. Raspberry Pi IoT Projects: Prototyping Experiments for Makers. – Washington, USA.: Liberty Lake, 2016, – 253 p.
9. Eben Upton, Gareth Halfacree. John. Raspberry Pi® User Guide, 4th Edition. – Chichester, West Sussex, United Kingdom.: Wiley & Sons Ltd, 2016. – 315 p.
10. Sams Teach Yourself Python Programming for Raspberry Pi in 24 Hours, Second Edition, Pearson Education, Inc., 2016. – 1760 c.

Інформаційні ресурси Інтернету:

1. Сайт Академії Mikrotik:
   - https://mikrotik.kpi.ua/index.php/courses-list/category-raspberry
   - https://mikrotik.kpi.ua/index.php/courses-list/category-python
2. Персональний сайт викладача: – http://isearch.kiev.ua/
3. Сайт дистанційного навчання на платформі Moodle Академії Mikrotik: – http://iot.kpi.ua/lms/
4. Платформа дистанційного навчання «Сікорський»: – https://www.sikorsky-distance.org/