

Міністерство освіти і науки України  
Національний університет «Запорізька політехніка»

**МЕТОДИЧНІ ВКАЗІВКИ**

до виконання лабораторних робіт  
з дисципліни

**«Інженерія прикладних інтелектуальних застосунків»**

для студентів

спеціальностей 121 «Інженерія програмного забезпечення» та  
122 «Комп'ютерні науки»

2023

Методичні вказівки до виконання лабораторних робіт з дисципліни “Інженерія прикладних інтелектуальних застосунків” для студентів спеціальностей 121 «Інженерія програмного забезпечення» та 122 «Комп’ютерні науки» / А.О. Олійник, С. Д. Леощенко, Є.М. Федорченко. – Запоріжжя: НУ «Запорізька політехніка», 2023. – 105 с.

Автори: А. О. Олійник, д.т.н., професор  
С. Д. Леощенко, доктор філософії, ст. викладач  
Є. М. Федорченко, ст. викладач

Рецензент: В.М. Льовкін, к.т.н., доцент

Відповідальний  
за випуск: С. О. Субботін, д.т.н., професор

Затверджено  
на засіданні кафедри  
програмних засобів

Протокол № 5  
від “ ” серпня 2023 р.

## ЗМІСТ

<b>Вступ .....</b>	<b>5</b>
<b>1 Лабораторна робота № 1 Обробка часових рядів.....</b>	<b>6</b>
1.1 Мета роботи .....	6
1.2 Короткі теоретичні відомості .....	6
1.3 Приклад прогнозування на основі даних, поданих у вигляді часового ряду .....	16
1.4 Завдання на лабораторну роботу.....	38
1.5 Зміст звіту.....	39
1.6 Контрольні запитання .....	39
<b>2 Лабораторна робота № 2 Комп'ютерний зір. Класифікація та розпізнавання зображень .....</b>	<b>41</b>
2.1 Мета роботи .....	41
2.2 Короткі теоретичні відомості .....	41
2.2.1 Розпізнавання зображень .....	41
2.2.2 Виявлення об'єктів на графічних зображеннях .....	49
2.3 Приклад розробки програмного забезпечення для обробки зображень .....	49
2.3.1 Розробка програмного забезпечення для розпізнавання графічних об'єктів .....	50
2.3.2 Розробка програмного забезпечення для пошуку об'єктів на графічних зображеннях .....	55
2.4 Завдання на лабораторну роботу.....	58
2.5 Зміст звіту.....	59
2.6 Контрольні запитання .....	59
<b>3 Лабораторна робота № 3 Обробка природної мови та текстової інформації.....</b>	<b>61</b>
3.1 Мета роботи .....	61
3.2 Короткі теоретичні відомості .....	61
3.2.1 Класифікація текстів .....	62
3.2.2 Генерація текстової інформації .....	65
3.3 Приклад розробки програмного забезпечення для обробки природної мови .....	67
3.3.1 Розробка програмного забезпечення для класифікації текстів ...	67
3.3.2 Розробка програмного забезпечення для генерації текстів .....	74

3.4 Завдання на лабораторну роботу.....	84
3.5 Зміст звіту.....	86
3.6 Контрольні запитання .....	86
<b>4 Лабораторна робота № 4 Розпізнавання звуку .....</b>	<b>87</b>
4.1 Мета роботи .....	87
4.2 Короткі теоретичні відомості .....	87
4.3 Приклад розробки програмного забезпечення для обробки аудіо-даних.....	90
4.4 Завдання на лабораторну роботу.....	100
4.5 Зміст звіту.....	101
4.6 Контрольні запитання .....	101
<b>Література.....</b>	<b>103</b>

## ВСТУП

Дане видання призначене для вивчення та практичного освоєння студентами усіх форм навчання основ інженерії прикладних інтелектуальних застосунків.

Відповідно до графіка студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою. У цих методичних вказівках використовуються матеріали, посилання на які наведено у переліку літератури. Звичайно, в дані методичні вказівки неможливо було внести весь матеріал, необхідний для виконання та захисту лабораторних робіт. Тому тут містяться основні, базові теоретичні відомості, необхідні для виконання лабораторних робіт. Таким чином для виконання лабораторної роботи та при підготовці до її захисту необхідно ознайомитись з конспектом лекцій та проробити весь матеріал, наведений в переліку рекомендованої літератури. При цьому не варто обмежуватись лише наведеним списком.

Для одержання заліку з кожної роботи студент здає викладачу оформлений звіт, а також демонструє на екрані комп'ютера результати виконання лабораторної роботи.

Звіт має містити:

- титульний аркуш;
- тему та мету роботи;
- завдання до роботи;
- лаконічний опис теоретичних відомостей;
- результати виконання лабораторної роботи;
- змістовний аналіз отриманих результатів та висновки.

Звіт виконують на білому папері формату А4 (210 × 297 мм) або подають в електронному вигляді.

Під час співбесіди при захисті лабораторної роботи студент повинний виявити знання про зміст роботи та методи виконання кожного етапу роботи, а також вміти продемонструвати результати роботи на конкретних прикладах. Студент повинний вміти правильно аналізувати отримані результати. Для самоперевірки при підготовці до виконання і захисту роботи студент повинен відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

# 1 ЛАБОРАТОРНА РОБОТА № 1

## ОБРОБКА ЧАСОВИХ РЯДІВ

### 1.1 Мета роботи

Навчитися реалізовувати програмні проекти у сфері інженерії програмного забезпечення, пов'язані з необхідністю обробки часових послідовностей даних.

### 1.2 Короткі теоретичні відомості

Часто вхідні дані представляються у вигляді **часових послідовностей (time series)** даних, які являють собою ряд точок даних, проіндексованих у часовому порядку. Найчастіше часовий ряд – це набір даних, виміряних у послідовні однакові моменти часу (послідовність даних у дискретний час). Прикладами часових рядів є висоти океанських припливів через деякий час, кількість сонячних плям, щоденне кінцеве значення промислового індексу Доу-Джонса, курси валют або акцій, медичні дані (дані кардіограми) та ін. [1]–[3].

Часові ряди **використовуються** для:

- обробки сигналів технічних об'єктів;
- моделювання поширення хвороб;
- аналізу динаміки цін на товарних, фондових та інших ринках;
- розпізнавання образів;
- прогнозування погоди;
- розробки іншого програмного забезпечення, пов'язаного з аналізом часових послідовностей.

Для аналізу часових послідовностей використовується техніка **прогнозування часових рядів (Time Series Forecasting)** – це техніка, яка використовує історичні та поточні дані для прогнозування майбутніх значень протягом певного періоду часу або певної точки в майбутньому. Аналізуючи дані, які ми зберігаємо в минулому, ми можемо приймати обґрунтовані рішення, які можуть спрямовувати нашу бізнес-стратегію та допомагати нам зрозуміти майбутні тенденції.

Оскільки прогнозування — це не строго визначений метод, а скоріше комбінація методів аналізу даних, аналітики та дослідники даних повинні враховувати обмеження, які мають моделі прогнозу-

вання, а також самі дані. Тобто необхідно обґрунтовано обрати метод прогнозування часових послідовностей, який якнайкраще підходить для розв'язання конкретної прикладної задачі. Тому важливим кроком при вирішенні задач, пов'язаних з прогнозуванням часових рядів, є **аналіз предметної області** для розуміння даних. Занурюючись у проблемну область, розробник може легше відрізнити випадкові коливання від стабільних і постійних тенденцій в історичних даних. Це стане в нагоді під час налаштування моделі прогнозування для створення найкращих прогнозів і навіть при розгляді того, який метод прогнозування використовувати.

Використовуючи аналіз часових рядів, необхідно враховувати деякі обмеження даних. Поширені проблеми включають узагальнення з одного джерела даних і труднощі в отриманні відповідних вимірювань і точному визначенні правильної моделі для представлення даних. Тому часто до синтезу прогнозуючої моделі виконують **попередню обробку даних** (стиснення даних, обробка даних з пропущеними значеннями, нормалізація даних та інші перетворення).

Існує чимало факторів, пов'язаних із прогнозуванням часових рядів, але найважливіші з них є такі:

- об'єм даних;
- якість даних;
- сезонність;
- тренди (тенденції);
- несподівані події.

Об'єм даних, ймовірно, є найважливішим фактором (за умови, що дані точні). Хорошим емпіричним правилом було б, що чим більше даних ми маємо, тим краще наша модель створюватиме прогнози. Це також полегшує нашій моделі розрізнення тенденцій і шуму в даних.

Якість даних передбачає деякі базові вимоги, такі як відсутність дублікатів, пропусків, стандартизований формат даних, а також те, щоб дані збиралися послідовно або через регулярні проміжки часу.

Сезонність означає, що існують чіткі періоди часу, коли дані містять постійні відхилення. Наприклад, якби інтернет-магазин проаналізував свою історію продажів, було б очевидно, що святковий сезон призводить до збільшення обсягу продажів.

Тренди (тенденції) – це, мабуть, найважливіша інформація, яку необхідно отримати із необроблених даних. Вони вказують, чи буде

змінна в часовому ряді збільшуватися або зменшуватися в даний період.

Неочікувані події (іноді їх також називають шумом або порушеннями) завжди можуть статися, і ми повинні враховувати це під час створення моделі прогнозування. Вони представляють шум в історичних даних, і вони також непередбачувані.

У наш час використовуються різні **групи методів прогнозування часових рядів**:

- декомпозиція часових рядів;
- регресійні моделі часових рядів;
- експоненціальне згладжування;
- моделі ARIMA;
- нейронні мережі.

Часто використовуються різні методи, аналізуються їх результати, після чого використовується один найкращий або група методів чи певна комбінація методів (гібридний підхід).

**Декомпозиція часових рядів** — це метод явного моделювання даних як комбінації компонентів сезонного (seasonal), трендового (trend), циклічного (cycle) й залишкового (remainder) компонентів замість моделювання за допомогою часових залежностей і автокореляції. Цей підхід можна використовувати або як окремий метод для прогнозування часових рядів, або як перший крок до кращого розуміння набору даних на етапі аналізу предметної області.

Використовуючи модель декомпозиції, необхідно спрогнозувати майбутні значення для кожного з компонентів, наведених вище, а потім додати ці прогнози разом, щоб отримати загальний прогноз.

Декомпозиція часових рядів відноситься до техніки, яка розкладає дані часових рядів на деякі компоненти. Як правило, до таких компонентів належать такі:

- тренд (trend);
- цикл (cycle);
- сезонність (seasonal);
- залишок (residual, remainder).

Приклад декомпозиції наявних даних (observed) на тренд (trend), сезонність (seasonal) та залишок (residual) наведено на рис. 1.1.



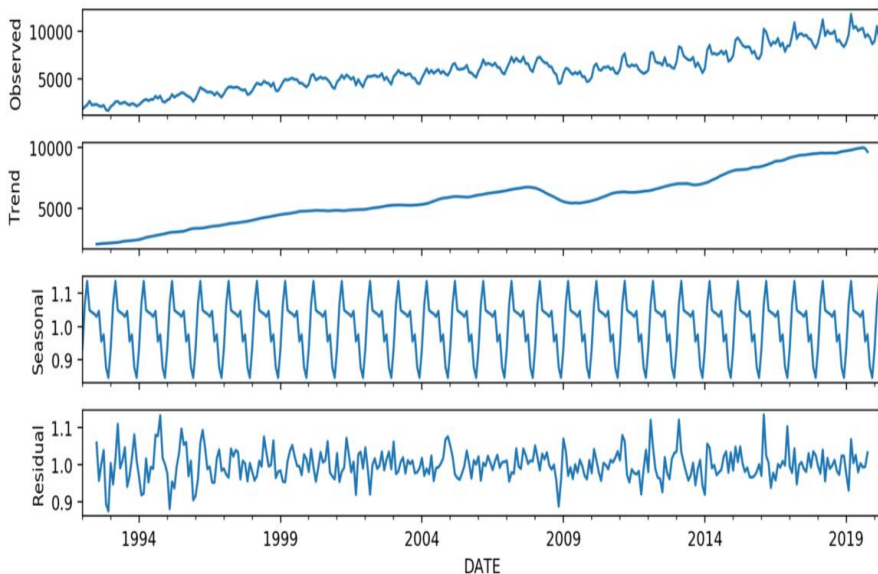


Рисунок 1.1 – Приклад розкладання наявних даних (observed) на тренд (trend), сезонність (seasonal) та залишок (residual) [2]

Приклад взято з джерела [2], в якому детально розписано цей підхід, а також наведено програмний код для його реалізації.

Декомпозиція на основі швидкості зміни значень часового ряду є важливою технікою, коли мова йде про аналіз сезонних коригувань. Техніка створює кілька компонентних рядів, які, об'єднуючи (з використанням додавання та множення), призводять до початкового часового ряду. Кожен із компонентів має певну характеристику або тип поведінки, і зазвичай вони включають:

- $T_t$ : компонент тренду в момент часу  $t$ , який описує довгострокову прогресію часового ряду. Тенденція наявна, коли спостерігається послідовне збільшення або зменшення в напрямку даних. Компонент тренду (тенденції) не обмежений лінійною функцією;

- $C_t$ : циклічна складова в момент часу  $t$  – відображає повторювані, але неперіодичні коливання. Тривалість цих коливань залежить від характеру часового ряду;

–  $S_t$ : сезонна складова в момент часу  $t$  – відображає сезонність (сезонні коливання). Таку сезонну модель можна знайти в часових рядах, на які впливають сезонні фактори. Сезонність зазвичай виникає у фіксований і відомий період (наприклад, сезон відпусток);

–  $I_t$ : нерегулярний компонент (або «шум») у момент часу  $t$  – представляє випадкові та нерегулярні впливи. Його також можна вважати рештою часового ряду після видалення інших компонентів.

Об'єднання цих компонентів може виконуватися за адитивною чи мультиплікативною технікою.

Адитивна декомпозиція означає, що дані часових рядів є функцією суми своїх компонентів, тому модель можна представити наступним рівнянням:

$$y_t = T_t + C_t + S_t + I_t,$$

де  $y_t$  – дані часових рядів,  $T_t$  – компонент тренду,  $C_t$  – циклічний компонент,  $S_t$  – сезонний компонент,  $I_t$  – залишок.

Мультиплікативна декомпозиція визначає дані часових рядів як функцію добутку його компонентів:

$$y_t = T_t \cdot C_t \cdot S_t \cdot I_t.$$

Якщо величина сезонної складової є динамічною та змінюється з часом, то модель ряду варто будувати як мультиплікативну. Якщо сезонна складова постійна, ряд є адитивним.

Деякі методи поєднують компоненти тренду та циклу в один компонент циклу тренду. Його можна назвати компонентом тренду, навіть якщо він містить видимі властивості циклу.

**Регресійні моделі часових рядів** відносяться до статистичного підходу прогнозування майбутніх значень на основі історичних даних. Прогнозна змінна також називається регресивною, залежною або вихідною змінною. Вхідні змінні іноді називають регресорами, незалежними або пояснювальними змінними. Алгоритми регресії намагаються обчислити лінію (не обов'язково пряму), яка найкраще підходить для заданого набору даних. Наприклад, алгоритм лінійної регресії може спробувати мінімізувати суму квадратів різниць між спостережуваним значенням і прогнозованим значенням, щоб знайти найкращу відповідність.

Регресійна модель, що описує лінійну залежність між змінною прогнозу  $y$  і простою змінною предиктора  $x$ , може бути подана таким чином:

$$y_t = \beta_0 + \beta_1 x_t + \varepsilon_t,$$

де коефіцієнти  $\beta_0$  та  $\beta_1$  позначають точку перетину та нахил лінії.

Важливо відзначити, що спостереження не ідеально вирівнюються на прямій лінії, а дещо розкидані навколо неї. Кожне зі спостережень  $y_t$  складається із систематичного компонента моделі  $\beta_0 + \beta_1 x_t$  і компонента помилки  $\varepsilon_t$ . Компонент помилки охоплює будь-які відхилення від прямолінійної моделі.

Лінійна модель дуже обмежена в апроксимації базових функцій, тому інші моделі регресії можуть бути більш ефективними.

**Експоненціальне згладжування** (Exponential Smoothing) являє собою емпіричне правило для згладжування даних часових рядів за допомогою функції експоненціального вікна (рис. 1.2). У той час як метод простого ковзного середнього зважає історичні дані однаково, щоб зробити прогнози щодо майбутнього, експоненціальне згладжування використовує експоненціальні функції для обчислення зменшення ваг з часом.

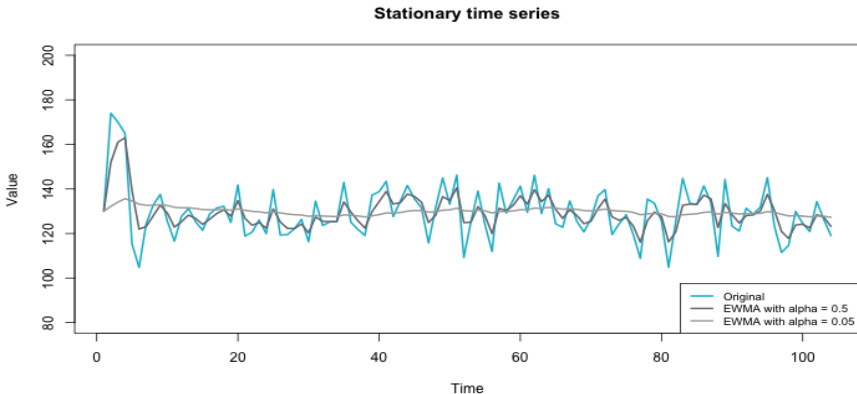


Рисунок 1.2 – Приклад експоненціального згладжування [3]

Існують різні типи експоненціального згладжування, які включають просте експоненціальне згладжування, потрійне експоненціаль-

не згладжування (також відоме як метод Холта-Уінтерса) та інші методи.

**Модель ARIMA** (AutoRegressive Integrated Moving Average) — це метод прогнозування, який поєднує модель авторегресії та модель ковзного середнього. Авторегресія використовує спостереження з попередніх часових кроків, щоб передбачити майбутні значення за допомогою рівняння регресії. Авторегресійна модель використовує лінійну комбінацію минулих значень змінних для створення прогнозів.

Модель SARIMA (Seasonal ARIMA) є розширенням моделі ARIMA. Це досягається шляхом додавання лінійної комбінації минулих сезонних значень і помилок прогнозу.

Відзначимо, що вище розглянуті методи, які відносяться до класичних методів прогнозування часових рядів, мають деякі обмеження, зокрема:

- робота з повними даними. Відсутні або пошкоджені дані зазвичай не підтримуються;
- фокус на лінійних залежностях: припущення лінійної залежності виключає більш складні типи залежностей;
- робота з одновимірними даними: багато реальних проблем мають кілька входних змінних, що ускладнює застосування таких методів;
- фокус на на одноетапних прогнозах: багато реальних проблем вимагають прогнозів із довгим часовим горизонтом.

Моделі на основі **нейронних мереж** можуть бути ефективними для розв'язання більш складних проблем прогнозування часових рядів із кількома входними змінними, складними нелінійними зв'язками та відсутніми даними.

Нейронні мережі здатні досить добре апроксимувати будь-яку безперервну функцію для прогнозування часових рядів. У той час як класичні методи, як правило, припускають лінійне співвідношення між входами та виходами, нейронні мережі не зв'язані цим обмеженням. Вони здатні апроксимувати будь-яку нелінійну функцію без попереднього знання про властивості рядів даних.

Нейронні мережі, такі як багаточарові персептрони, пропонують численні переваги, зокрема:

- стійкість до шуму. Нейронні мережі не тільки стійкі до шуму, коли йдеться про вхідні дані, але також стійкі до функції відображення. Це може стати в нагоді під час роботи з даними, які містять відсутні значення;

- підтримка нелінійних залежностей: нейронні мережі не пов'язані з сильними припущеннями та жорсткою функцією відображення. Вони здатні постійно вивчати нові лінійні та нелінійні зв'язки;

- багатовимірні вхідні дані (Multivariate inputs): багатовимірне прогнозування підтримується, оскільки можлива різна кількість вхідних ознак (змінних) при синтезі нейромережевої моделі;

- багатокрокові прогнози (Multi-step forecasts): кількість вихідних значень також є змінною.

Для прогнозування числових рядів доцільно використовувати такі типи нейронних мереж:

- багатошарові перцептрони (Multilayer Perceptrons, MLP);
- згорткові нейронні мережі (Convolutional Neural Networks);
- рекурентні нейронні мережі (Recurrent Neural Networks).

**Багатошарові перцептрони** (Multilayer Perceptrons, MLP) можна використовувати для моделювання проблем прогнозування однофакторних та багатфакторних часових рядів. Найпростіша модель MLP має один прихований шар вузлів і вихідний рівень, який використовується для прогнозування. Створення простої моделі типу MLP з використанням бібліотеки tensorflow та мови Python можна виконати таким чином:

```
model = Sequential()
model.add(Dense(25, activation='relu', input_dim=n_steps))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

У прикладі створюється модель model з одним прихованим шаром, який містить 25 нейронів. Кількість вхідних ознак у навчальній вибірці дорівнює n\_steps. Функція активації activation='relu' – функція випрямленої лінійної одиниці, яка повертає значення вхідного параметру у випадку, якщо воно більше нуля, в іншому випадку повертає нульове значення. Можна використовувати й інші функції активації, наприклад, sigmoid (сигмоїдна), softmax, softplus, softsign, exponential, tanh, selu, elu та інші. Другий шар є вихідним та містить один нейрон model.add(Dense(1)).

Складність використання MLP для прогнозування часових рядів полягає в підготовці даних. Зокрема, спостереження відставання повинні бути зведені у вектори ознак.

Оскільки однофакторні часові ряди – це набір даних, що складається з однієї серії спостережень із часовим упорядкуванням, і модель потрібна для навчання на основі серії минулих спостережень, щоб передбачити наступне значення в послідовності. Перш ніж можна буде змодельовати однофакторний ряд, його необхідно підготувати.

Модель MLP дізнається функцію, яка відображає послідовність минулих спостережень як вхідні дані для вихідного спостереження. Таким чином, послідовність спостережень повинна бути перетворена в кілька прикладів (екземплярів), з яких модель може навчатися.

Для цього можна розділити послідовність на кілька шаблонів з вхідними та вихідними значеннями, які називаються зразками, де декілька (наприклад, п'ять) часових кроків використовуються як вхідні дані, а один (наприклад, шостий) часовий крок використовується як вихідні дані для однокрокового прогнозу, який вивчається.

Приклад синтезу багатошарових персептронів для однофакторних та багатофакторних даних, а також для однокрокового та для багатокрокового прогнозування наведено у джерелі [4].

**Згорткові нейронні мережі** (Convolutional Neural Networks, CNN) – це тип нейронних мереж, розроблених для ефективної обробки даних у вигляді зображень. Вони довели ефективність у вирішенні складних проблем комп'ютерного зору, досягаючи досить ефективних результатів у таких завданнях, як класифікація зображень, і надаючи компонент у гібридних моделях для абсолютно нових проблем, таких як локалізація об'єктів, підписи до зображень тощо.

Вони досягають цього, працюючи безпосередньо з необробленими даними, такими як необроблені значення пікселів, замість специфічних для домену або ручних функцій, отриманих із необроблених даних. Потім модель дізнається, як автоматично витягувати функції з необроблених даних, які безпосередньо корисні для вирішення проблеми. При використанні CNN характеристики зображень виділяються незалежно від того, як вони відбуваються в даних, забезпечуючи так звану інваріантність перетворення.

Здатність CNN навчатися та автоматично витягувати характеристики з необроблених вхідних даних може бути застосована до задач прогнозування часових рядів. Послідовність спостережень можна роз-

глядати як одновимірне зображення, яке модель CNN може прочитати та розділити на найбільш помітні елементи.

CNN отримують переваги багатозарових персептронів для прогнозування часових рядів, а саме підтримку багатовимірних вхідних даних, багатовимірного виходу та вивчення довільних, але складних функціональних зв'язків, але не вимагають, щоб модель навчалася безпосередньо із спостережень затримок. Замість цього модель може дізнатися представлення з великої вхідної послідовності, яке є найбільш релевантним для проблеми прогнозування.

Створення згорткової нейронної мережі з використанням бібліотеки tensorflow та мови Python можна виконати таким чином:

```
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(n_steps, n_features)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Приклад використання згорткової нейронної мережі для однофакторних та багатфакторних даних, а також для однокрокового та для багатокрокового прогнозування наведено у джерелі [5].

**Рекурентні нейронні мережі** (Recurrent Neural Networks, RNN), такі як мережа довгострокової короткочасної пам'яті (Long Short-Term Memory network, LSTM), додають явну обробку порядку між спостереженнями під час вивчення функції відображення входів і виходів, чого не пропонують MLP або CNN. Це тип нейронної мережі, яка додає власну підтримку вхідних даних, що складаються з послідовностей спостережень.

Додавання послідовності є новим виміром для апроксимованої функції. Замість того, щоб відображати входи лише на виходи, мережа здатна до навчання функції відображення (mapping function) входів через певний час у вихід.

Ця здатність LSTM була використана з великим ефектом у складних проблемах обробки природної мови, таких як нейронний машинний переклад, де модель повинна вивчати складні взаємозв'язки між словами як у межах певної мови, так і між мовами під час перекладу з однієї мови на іншу. Цю можливість можна використовувати

для прогнозування часових рядів. На додаток до загальних переваг використання нейронних мереж для прогнозування часових рядів, рекурентні нейронні мережі також можуть автоматично вивчати часову залежність з даних. Тому найбільш відповідний контекст вхідних спостережень для очікуваного результату вивчається та може змінюватися динамічно.

У найпростішому випадку мережа показує одне спостереження за раз із послідовності, і вона може дізнатися, які спостереження вона бачила раніше є релевантними та наскільки вони релевантні для прогнозування. Модель вивчає відображення від входів до виходів і дізнається, який контекст із вхідної послідовності корисний для відображення, і може динамічно змінювати цей контекст за потреби.

Приклад використання рекурентної нейронної мережі LSTM для прогнозування часових рядів наведено у джерелі [6].

### **1.3 Приклад прогнозування на основі даних, поданих у вигляді часового ряду**

Як приклад, будемо використовувати програмні коди, подані у джерелі [7], а також дані з [8].

Цей набір даних містить 14 різних характеристик, зокрема, температура повітря, атмосферний тиск та вологість. Вони збиралися кожні 10 хвилин, починаючи з 2003 року. Для аналізу будемо використовувати лише дані, зібрані в період з 2009 по 2016 рік [7], [8].

Програмне забезпечення буде розроблюватися з використанням бібліотеки tensorflow та мови Python. Для цього будуть необхідними такі модулі та бібліотеки:

```
import os
import datetime

import IPython
import IPython.display
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
```



Для завантаження набору даних за певною адресою можна використати такий програмний код:

```
zip_path = tf.keras.utils.get_file(
origin='https://storage.googleapis.com/tensorflow/tf-
keras-datasets/jena_climate_2009_2016.csv.zip',
    fname='jena_climate_2009_2016.csv.zip',
    extract=True)
csv_path, _ = os.path.splitext(zip_path)
```

У випадку завантаження з файлу необроблених даних (файл TS\_01.csv) можна скористатися таким кодом:

```
df = pd.read_csv('TS_01.csv')
```

Виконаємо попередню обробку даних. Будемо розглядати лише погодинні прогнози, тому виділимо з даних 10-хвилинні інтервали на одногодинні:

```
df = pd.read_csv(csv_path)
# Slice [start:stop:step], starting from index 5 take
every 6th record.
df = df[5::6]
```

```
date_time = pd.to_datetime(df.pop('Date Time'), for-
mat='%d.%m.%Y %H:%M:%S')
```

Відобразимо деякі дані:

```
print("Фрагмент скороченої вибірки (залишено кожен 6-й
рядок базової):")
print(df.head())
```

Побудуємо графіки деяких з часових рядів, поданих у вхідному наборі даних:

```
plot_cols = ['T (degC)', 'p (mbar)', 'rho (g/m**3)']
plot_features = df[plot_cols]
plot_features.index = date_time
_ = plot_features.plot(subplots=True)

plot_features = df[plot_cols][:480]
plot_features.index = date_time[:480]
_ = plot_features.plot(subplots=True)
plt.show()
```

Відобразимо статистику вхідного набору даних:

```
df.describe().transpose()
```

Далі наведено приклади способів попередньої обробки даних, які є корисними для завантаженого набору даних. При обробці інших наборів можуть застосовуватися інші підходи до попередньої обробки.

Зі статистики досліджуваного набору видно, що у стовпцях швидкості вітру (wv (m/s)) та поривів вітру (max. wv (m/s)) мінімальне значення є меншим за нуль (-9999). Очевидно, що це є помилкою, оскільки значення швидкості вітру (або поривів вітру) повинно бути не менше нуля. Тому замінимо відповідні значення на нулі:

```
wv = df['wv (m/s)']
bad_wv = (wv < 0)
wv[bad_wv] = 0.0
```

```
max_wv = df['max. wv (m/s)']
bad_max_wv = (max_wv < 0)
max_wv[bad_max_wv] = 0.0
```

Останній стовпець даних, wd (deg), вказує напрям вітру в градусах. Кути не підходять для введення у модель, оскільки : 360° та 0° повинні розташовуватися близько один до одного (оскільки по суті це одне й те саме значення) і плавно переходити один до одного. Крім того, напрямок не має значення, якщо вітер не дме. Тому для легшої інтерпретації моделі перетворимо стовпці напрямку та швидкості вітру на вектор вітру:

```
wv = df.pop('wv (m/s)')
max_wv = df.pop('max. wv (m/s)')
```

```
# Перетворення на радіани.
wd_rad = df.pop('wd (deg)')*np.pi / 180
```

```
# Розрахунок компонентів x та y вектору швидкості
df['Wx'] = wv*np.cos(wd_rad)
df['Wy'] = wv*np.sin(wd_rad)
```

```
# Розрахунок компонентів x and y вектору пориву вітру
df['max Wx'] = max_wv*np.cos(wd_rad)
df['max Wy'] = max_wv*np.sin(wd_rad)
```

Виконаємо також перетворення днів та років у періодичний вигляд для можливості використання цих параметрів як вхідних аргументів для побудови прогностичної моделі:

```
timestamp_s = date_time.map(pd.Timestamp.timestamp)
day = 24*60*60
year = (365.2425)*day
df['Day sin'] = np.sin(timestamp_s * (2 * np.pi / day))
df['Day cos'] = np.cos(timestamp_s * (2 * np.pi / day))
df['Year sin'] = np.sin(timestamp_s * (2 * np.pi /
year))
df['Year cos'] = np.cos(timestamp_s * (2 * np.pi /
year))
```

Також можна видалити зайві стовбці, наприклад, нульовий, які можуть автоматично додаватися при перетворенні даних з одного формату на інший:

```
df=df.drop(df.columns[[0]], axis=1)
```

Ще раз відзначимо, що для попередньої обробки даних застосовуються різні методи та підходи в залежності від особливостей конкретного набору вхідних даних.

Після попередньої обробки даних виконаємо їх перетворення на три набори: **навчальні, перевірочні та тестові дані**. Для цього будемо використовувати поділ (70%, 20%, 10%) для навчальних, перевірочних та тестових наборів. При цьому такий розподіл даних гарантує, що результати перевірки/тестування будуть більш реалістичними, оскільки оцінюються на основі даних, зібраних після навчання моделі.

```
column_indices = {name: i for i, name in enumerate(df.columns)}
```

```
n = len(df)
train_df = df[0:int(n*0.7)]
val_df = df[int(n*0.7):int(n*0.9)]
test_df = df[int(n*0.9)::]

num_features = df.shape[1]

print("Навчальна вибірка (70%):")
print(train_df)
print("Перевірочна вибірка (20%):")
```

```
print(val_df)
print("Тестова вибірка (10%):")
print(test_df)
```

**Нормалізація даних.** Перед навчанням прогностичної моделі, наприклад, на основі нейронної мережі, важливо масштабувати значення вхідних параметрів. Нормалізація є поширеним способом масштабування, який полягає у відніманні середнього значення і поділу на стандартне відхилення кожної ознаки.

Середнє значення та стандартне відхилення слід обчислювати тільки з використанням навчальних даних, щоб моделі не мали доступу до значень у перевірочних та тестових наборах.

```
train_mean = train_df.mean()
train_std = train_df.std()

train_df = (train_df - train_mean) / train_std
val_df = (val_df - train_mean) / train_std
test_df = (test_df - train_mean) / train_std
```

Після попередніх перетворень даних необхідно створити функції для створення **вікон даних** на основі наявних часових послідовностей. Саме вік послідовних вибірок з даних і будуть вхідними даними для синтезу прогностичних моделей на основі часових рядів.

Основні особливості вікон даних такі: ширина (width, кількість часових кроків), зміщення (offset) між даними, які функції використовуються як вхідні дані, які як вихідні або те й інше. Розробимо програмні функції для створення вікон даних (рис. 1.3) для можливості подальшого синтезу прогностичних моделей для прогнозування з одним виходом або кількома виходами, а також з одним або декількома часовими кроками.

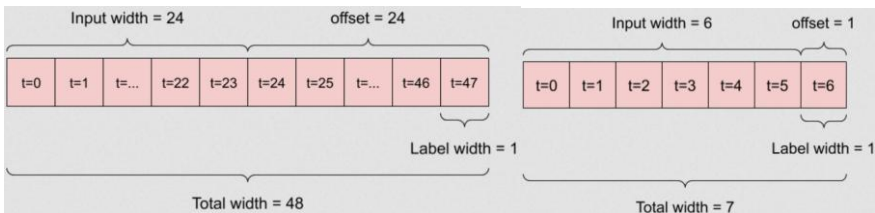


Рисунок 1.3 – Приклад вікон даних

На рис.1.3 зліва зображено приклад вікна даних для виконання одного прогнозу на 24 години вперед, враховуючи 24-годинну історію попередніх даних. На рис.1.3 справа зображено приклад вікна даних для синтезу моделі, яка робить прогноз на одну годину вперед, враховуючи шість годин історичних даних. Фактично кожне таке вікно буде являти собою один з багатьох екземплярів даних для синтезу прогностичної моделі.

Нижче наведено клас WindowGenerator, який може:

- обробляти індекси та зсуви, як показано на рис.1.3;
- розділити вікна даних на пари вхідних та вихідних значень (features, labels);
- побудувати вміст створених вікон;
- ефективно генерувати пакети (набори) цих вікон із навчальних, оціночних та тестових даних.

```
class WindowGenerator():
    def __init__(self, input_width, label_width, shift,
                  train_df=train_df, val_df=val_df,
                  test_df=test_df,
                  label_columns=None):
        # Store the raw data.
        self.train_df = train_df
        self.val_df = val_df
        self.test_df = test_df

        # Work out the label column indices.
        self.label_columns = label_columns
        if label_columns is not None:
            self.label_columns_indices = {name: i for i, name
in
                                         enumerate(
ate(label_columns))
            self.column_indices = {name: i for i, name in
                                         enumerate(train_df.columns)}

        # Work out the window parameters.
        self.input_width = input_width
        self.label_width = label_width
        self.shift = shift

        self.total_window_size = input_width + shift
```

```

        self.input_slice = slice(0, input_width)
        self.input_indices =
np.arange(self.total_window_size)[self.input_slice]

        self.label_start = self.total_window_size -
self.label_width
        self.labels_slice = slice(self.label_start, None)
        self.label_indices =
np.arange(self.total_window_size)[self.labels_slice]

    def __repr__(self):
        return '\n'.join([
            f'Total window size: {self.total_window_size}',
            f'Input indices: {self.input_indices}',
            f'Label indices: {self.label_indices}',
            f'Label column name(s): {self.label_columns}'])

```

Метод `__init__` класу `WindowGenerator` включає всю необхідну логіку для індексів введення та міток. Він також приймає навчальні, оціночні та тестові набори даних як вхідні дані.

Враховуючи список послідовних вхідних даних, метод `split_window` перетворює їх у вікно вхідних даних та вікно міток:

```

def split_window(self, features):
    inputs = features[:, self.input_slice, :]
    labels = features[:, self.labels_slice, :]
    if self.label_columns is not None:
        labels = tf.stack(
            [labels[:, :, self.column_indices[name]] for
name in self.label_columns],
            axis=-1)

    # Slicing doesn't preserve static shape information,
    so set the shapes
    # manually. This way the `tf.data.Datasets` are easi-
    er to inspect.
    inputs.set_shape([None, self.input_width, None])
    labels.set_shape([None, self.label_width, None])

    return inputs, labels

```

```
WindowGenerator.split_window = split_window
```

Метод `make_dataset` візьме часовий ряд `DataFrame` і перетворить його на `tf.data.Dataset` з пар (`input_window`, `label_window`) за допомогою функції `tf.keras.utils.timeseries_dataset_from_array`:

```
def make_dataset(self, data):
    data = np.array(data, dtype=np.float32)
    ds = tf.keras.utils.timeseries_dataset_from_array(
        data=data,
        targets=None,
        sequence_length=self.total_window_size,
        sequence_stride=1,
        shuffle=True,
        batch_size=32,)

    ds = ds.map(self.split_window)

    return ds
```

```
WindowGenerator.make_dataset = make_dataset
```

Додамо властивості доступу до множин даних як `tf.data.Dataset` за допомогою методу `make_dataset`, який було визначено раніше:

```
@property
def train(self):
    return self.make_dataset(self.train_df)

@property
def val(self):
    return self.make_dataset(self.val_df)

@property
def test(self):
    return self.make_dataset(self.test_df)

@property
def example(self):
    result = getattr(self, '_example', None)
    if result is None:
```

```

        result = next(iter(self.train))
        self._example = result
    return result

```

```

WindowGenerator.train = train
WindowGenerator.val = val
WindowGenerator.test = test
WindowGenerator.example = example

```

Для відображення результатів у вигляді графіків створимо метод `plot`.

```

def plot(self, model=None, plot_col='T (degC)',
max_subplots=3):
    inputs, labels = self.example
    plt.figure(figsize=(12, 8))
    plot_col_index = self.column_indices[plot_col]
    max_n = min(max_subplots, len(inputs))
    for n in range(max_n):
        plt.subplot(max_n, 1, n+1)
        plt.ylabel(f'{plot_col} [normed]')
        plt.plot(self.input_indices, inputs[n, :,
plot_col_index],
                label='Inputs', marker='.', zorder=-10)

        if self.label_columns:
            label_col_index =
self.label_columns_indices.get(plot_col, None)
        else:
            label_col_index = plot_col_index

        if label_col_index is None:
            continue

        plt.scatter(self.label_indices, labels[n, :, la-
bel_col_index],
                    edgecolors='k', label='Labels',
c='#2ca02c', s=64)
        if model is not None:
            predictions = model(inputs)
            plt.scatter(self.label_indices, predictions[n, :,
label_col_index],

```



```

        marker='X', edgecolors='k', la-
bel='Predictions',
        c='#ff7f0e', s=64)

    if n == 0:
        plt.legend()

    plt.xlabel('Time [h]')
    plt.show()

WindowGenerator.plot = plot

```

Після розробки класу та методів для створення вікон даних виконаємо **розробку функцій для прогнозування часових рядів на основі різних моделей**.

Найпростіша модель, яку можна побудувати для такого типу даних, – це **базова модель (Baseline)**, яка **передбачає значення однієї функції** – один часовий крок (одна година) у майбутнє, ґрунтуючись тільки на поточних умовах.

Будемо будувати модель для прогнозування значення  $T$  (degC) (стовпець 'T (degC)' множини даних  $df$ ) на одну годину вперед. Отже ця модель просто повертає поточну температуру як прогноз, прогножуючи «без змін».

Для створення базової моделі спочатку виконаємо налаштування об'єкту `WindowGenerator` для створення однокрокових пар (`input`, `label`)

```

single_step_window = WindowGenerator(
    input_width=1, label_width=1, shift=1,
    label_columns=['T (degC)'])

```

Клас моделі `Baseline` наведено нижче:

```

class Baseline(tf.keras.Model):
    def __init__(self, label_index=None):
        super().__init__()
        self.label_index = label_index

    def call(self, inputs):
        if self.label_index is None:
            return inputs

```

```
result = inputs[:, :, self.label_index]
return result[:, :, tf.newaxis]
```

Створимо екземпляр `baseline` класу `Baseline` та оцінимо модель:

```
baseline = Baseline(label_index=column_indices['T'
(degC')])
```

```
baseline.compile(loss=tf.losses.MeanSquaredError(),
                 metrics=
[tf.metrics.MeanAbsoluteError()])
```

```
val_performance = {}
performance = {}
val_performance['Baseline'] = base-
line.evaluate(single_step_window.val)
performance['Baseline'] = base-
line.evaluate(single_step_window.test, verbose=0)
```

Вище наведено приклад роботи з моделлю `Baseline` лише для одного екземпляра, поданого у вигляді одного вікна даних `single_step_window`. Створимо більшу кількість вікон за допомогою класу `WindowGenerator` та згенеруємо вікна на 24 години послідовних вхідних даних та міток за раз (рис.1.4).

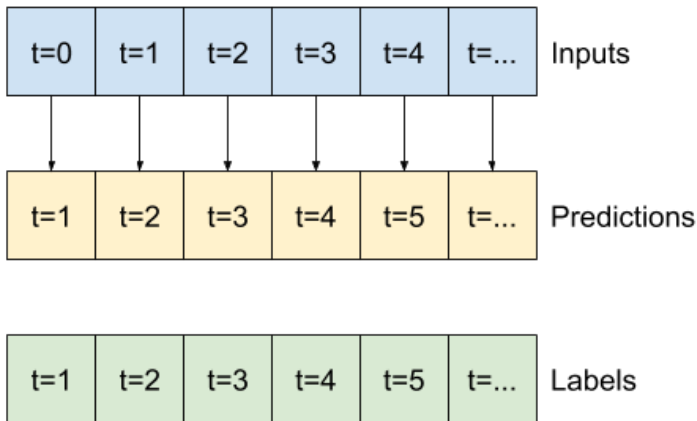


Рисунок 1.4 – Приклад вікон даних `wide_window`

Нова змінна `wide_window` не змінює спосіб роботи моделі. Модель, як і раніше, робить прогнози на одну годину вперед на основі одного вхідного часового кроку:

```
wide_window = WindowGenerator(
    input_width=24, label_width=24, shift=1,
    label_columns=['T (degC)'])

print(wide_window)
```

Як результат генерації вікон отримаємо таке:

```
Total window size: 25
Input indices: [ 0  1  2  3  4  5  6  7  8  9 10 11 12
13 14 15 16 17 18 19 20 21 22 23]
Label indices: [ 1  2  3  4  5  6  7  8  9 10 11 12 13
14 15 16 17 18 19 20 21 22 23 24]
Label column name(s): ['T (degC)']
```

Розширене вікно `wide_window` можна передати безпосередньо в ту саму базову модель `Baseline` без будь-яких змін коду. Це можливо, тому що входи та мітки мають однакову кількість часових кроків, а базова лінія просто перенаправляє вхід на вихід:

```
print('Input shape:', wide_window.example[0].shape)
print('Output shape:', baseline(wide_window.example[0]).shape)
```

На рис.1.5 наведено результати прогнозування часового ряду 'T (degC)' за допомогою моделі `Baseline`. Для виведення графіку необхідно використати команду

```
wide_window.plot(baseline)
```

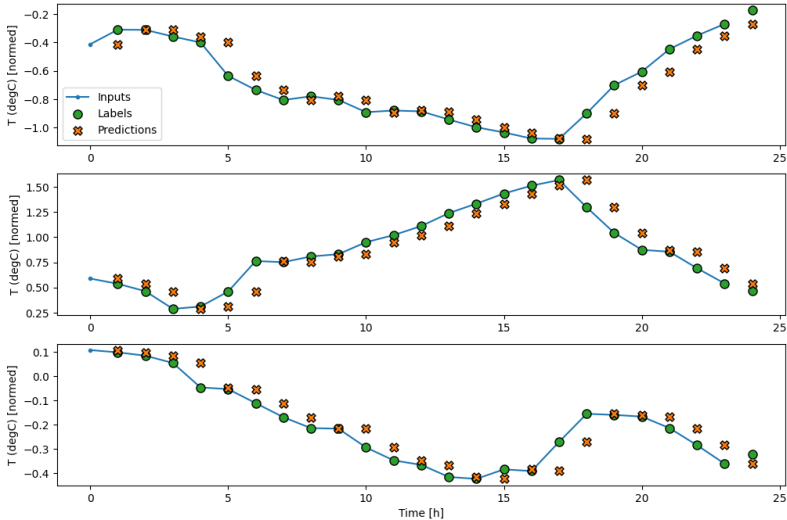


Рисунок 1.5 – Результати прогнозування за допомогою моделі Baseline

На наведених вище графіках трьох прикладів одноетапна модель Baseline працює протягом 24 годин:

- синя лінія Inputs показує вхідну температуру на кожному часовому інтервалі;
- зелені точки Labels показують цільове значення прогнозу (реальне значення вихідного параметру). Діапазон міток зміщений на 1 крок щодо входів;
- помаранчеві хрести Predictions – це прогнози моделі для кожного вихідного часового кроку. Якби модель передбачала ідеально, прогнози попадали б прямо в Labels.

Також досить простою моделлю є **лінійна модель**, яка забезпечує лінійне перетворення між входом і виходом.

Шар `tf.keras.layers.Dense` без набору функцій активації `activation` є лінійною моделлю. Шар перетворює тільки останню вісь даних (`batch, time, inputs`) в (`batch, time, units`); він застосовується незалежно до кожного елемента по осях `batch` і `time`.

Створення лінійної моделі можна виконати таким чином:

```
linear = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1)
])
```

У рядку коду, наведеному вище, відбувається створення лінійної моделі `linear`, яка поки що не навчена за конкретним набором даних.

Для навчання моделі (не тільки лінійної) створимо функцію, що забезпечує оптимальний підбір її параметрів в залежності від даних навчальної вибірки (вікон даних).

```
MAX_EPOCHS = 20
```

```
def compile_and_fit(model, window, patience=2):
    early_stopping =
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                     patience=patience,
                                     mode='min')

    model.compile(loss=tf.losses.MeanSquaredError(),
                  optimizer=tf.optimizers.Adam(),
                  metrics=[tf.metrics.MeanAbsoluteError()])

    history = model.fit(window.train, epochs=MAX_EPOCHS,
                        validation_data=window.val,
                        callbacks=[early_stopping])

    return history
```

Цю функцію будемо використовувати також для навчання інших моделей.

Виконаємо навчання лінійної моделі, а також оцінимо її ефективність:

```
history = compile_and_fit(linear, single_step_window)

val_performance['Linear'] = line-
ar.evaluate(single_step_window.val)
performance['Linear'] = line-
ar.evaluate(single_step_window.test, verbose=0)
```

Як і базову модель `baseline`, лінійну модель можна викликати для широких пакетів вікон `wide_window`. За такого використання модель робить набір незалежних прогнозів на послідовних часових кроках. Між прогнозами на кожному часовому етапі немає взаємодій (рис.1.6).

```
print('Input shape:', wide_window.example[0].shape)
print('Output shape:',
baseline(wide_window.example[0]).shape)
```

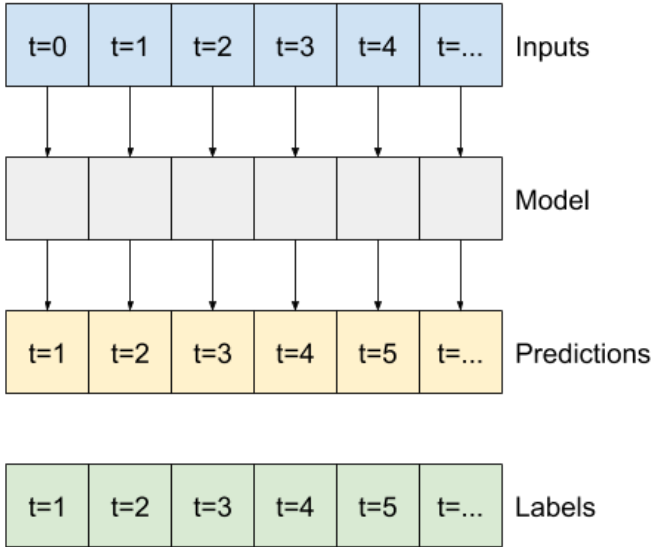


Рисунок 1.6 – Функціонування лінійної моделі

На рис.1.7 наведено результати прогнозування для широкого вікна даних `wide_window` за допомогою лінійної моделі. Як видно, в багатьох випадках прогноз явно кращий, ніж просто повернення вхідної температури (як у моделі `baseline`), але в деяких випадках він гірший.

```
wide_window.plot(linear)
```

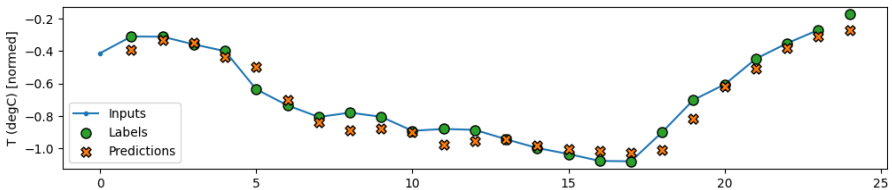


Рисунок 1.7 – Результати прогнозування для широкого вікна даних `wide_window` за допомогою лінійної моделі

Модель типу *багатошарового перцептрон* може бути створена таким чином:

```
dense = tf.keras.Sequential([
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=1)
])
```

Навчання моделі та виведення результатів її роботи:

```
history = compile_and_fit(dense, single_step_window)
```

```
val_performance['Dense'] =
dense.evaluate(single_step_window.val)
performance['Dense'] =
dense.evaluate(single_step_window.test, verbose=0)
```

Прогнозування на декілька кроків вперед на основі попередніх даних (рис. 1.8) може бути виконано за допомогою створення відповідних вікон даних.

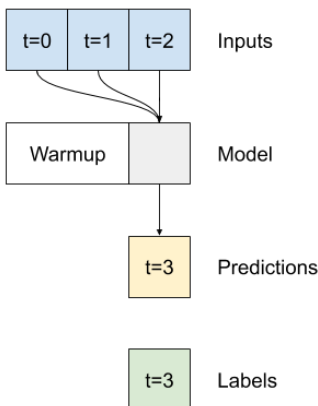


Рисунок 1.8 – Прогнозування на декілька кроків вперед

Створимо екземпляр класу WindowGenerator, який створюватиме пакети тригодинних вхідних даних та одногодинних міток.

```
CONV_WIDTH = 3
conv_window = WindowGenerator(
    input_width=CONV_WIDTH,
    label_width=1,
```

```
shift=1,
label_columns=['T (degC)'])
```

Створимо багатошарову нейромодель `multi_step_dense` на основі вікна з кількома вхідними кроками, додавши `tf.keras.layers.Flatten` як перший шар моделі:

```
multi_step_dense = tf.keras.Sequential([
    # Shape: (time, features) => (time*features)
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1),
    # Add back the time dimension.
    # Shape: (outputs) => (1, outputs)
    tf.keras.layers.Reshape([1, -1]),
])
```

Навчання багатошарової моделі `multi_step_dense` та виведення результатів її роботи:

```
history = compile_and_fit(multi_step_dense,
conv_window)

val_performance['Multi step dense'] = mul-
ti_step_dense.evaluate(conv_window.val)
performance['Multi step dense'] = mul-
ti_step_dense.evaluate(conv_window.test, verbose=0)
```

Ефективним засобом прогнозування часових рядів є *згорткові нейронні мережі*.

Шар згортки (`tf.keras.layers.Conv1D`) також використовує кілька часових кроків як вхідні дані для кожного прогнозу.

Нижче представлена та ж модель, що і `multi_step_dense`, переписана за допомогою згортки.

Зверніть увагу на зміни:

- `tf.keras.layers.Flatten` та перший `tf.keras.layers.Dense` замінюються `tf.keras.layers.Conv1D`;

- `tf.keras.layers.Reshape` більше не потрібний, оскільки згортка зберігає вісь часу у своїх вихідних даних.

```
conv_model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(filters=32,
```



```

        kernel_size=(CONV_WIDTH, ),
        activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1),
])

```

Навчання згорткової моделі `conv_model` та виведення результатів її роботи:

```
history = compile_and_fit(conv_model, conv_window)
```

```

val_performance['Conv'] =
conv_model.evaluate(conv_window.val)
performance['Conv'] =
conv_model.evaluate(conv_window.test, verbose=0)

```

Різниця між цією `conv_model` та моделлю `multi_step_dense` полягає в тому, що `conv_model` можна запускати на входах будь-якої довжини. Згортковий шар застосовується до ковзного вікна вхідних даних (рис. 1.9).

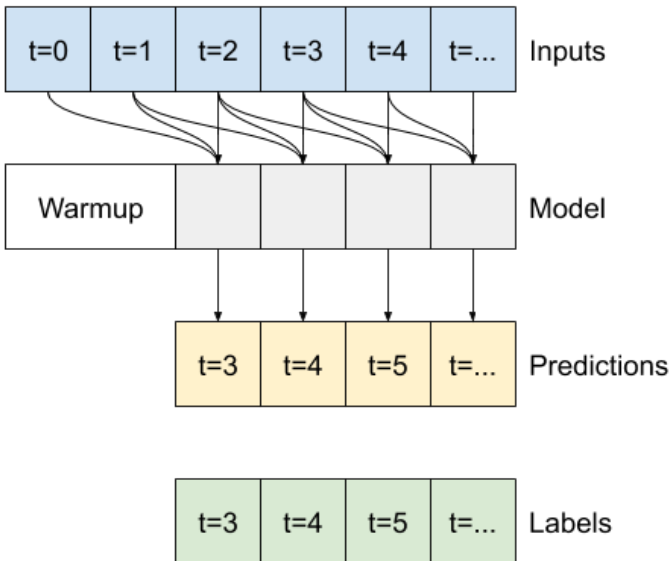


Рисунок 1.9 – Прогнозування на декілька кроків вперед за допомогою згорткової нейронної мережі

Створимо об'єкт класу WindowGenerator для генерації широких вікон з кількома додатковими часовими кроками введення, щоб довжина мітки та прогнозу збігалася:

```

LABEL_WIDTH = 24
INPUT_WIDTH = LABEL_WIDTH + (CONV_WIDTH - 1)
wide_conv_window = WindowGenerator(
    input_width=INPUT_WIDTH,
    label_width=LABEL_WIDTH,
    shift=1,
    label_columns=['T (degC)'])

print(wide_conv_window)

```

Виконаємо прогнозування за допомогою моделі conv\_model на основі вікна даних wide\_conv\_window. Кожен прогноз тут ґрунтується на 3 попередніх часових кроках (рис. 1.10).  
wide\_conv\_window.plot(conv\_model)

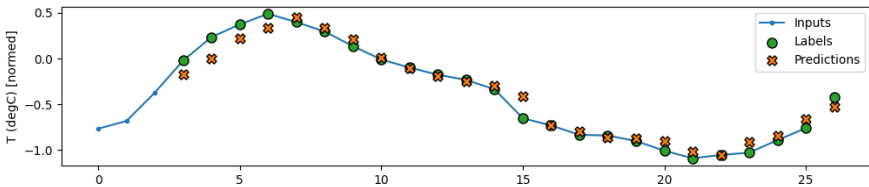


Рисунок 1.10 – Результати прогнозування для широкого вікна даних wide\_window за допомогою згорткової мережі

**Рекурентна нейронна мережа (RNN)** – це тип нейронної мережі, що добре підходить для даних часових рядів. RNN обробляють тимчасовий ряд крок за кроком, зберігаючи внутрішній стан від кроку до кроку.

Для прогнозування будемо використовувати шар RNN під назвою Long Short-Term Memory (tf.keras.layers.LSTM).

Важливим аргументом при конструюванні у Keras для всіх шарів RNN, таких як tf.keras.layers.LSTM, є аргумент return\_sequences. Цей параметр може налаштувати шар одним із двох способів.

Якщо return\_sequences=False, шар повертає лише вихідні дані останнього часового кроку, даючи моделі час, щоб прогріти свій внутрішній стан, перш ніж робити один прогноз (рис. 1.11).

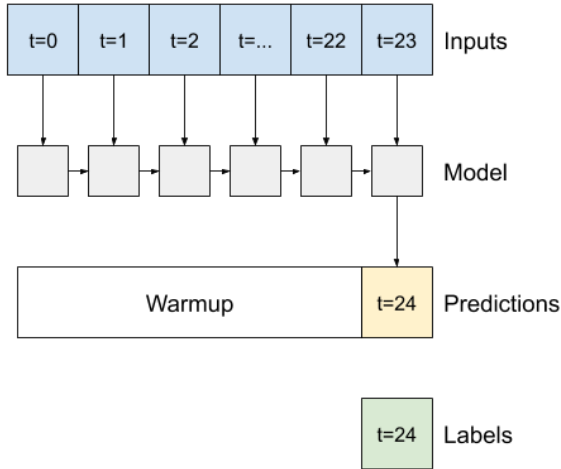


Рисунок 1.11 – Прогнозування за допомогою LSTM при `return_sequences=False`

Якщо `return_sequences=True`, шар повертає результат для кожного входу (рис.1.12). Це корисно для укладання шарів RNN та навчання моделі на кількох часових кроках одночасно.

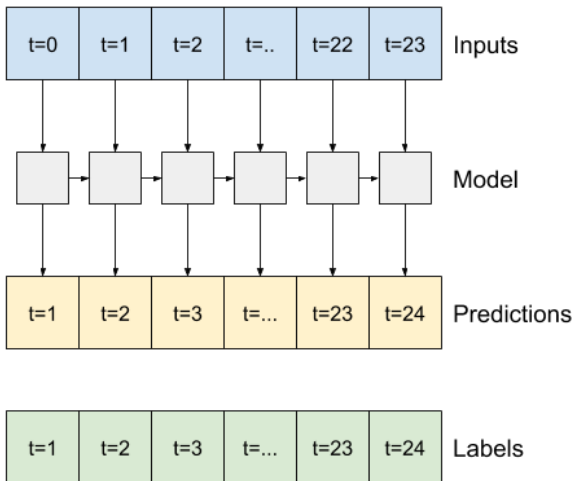


Рисунок 1.12 – Прогнозування за допомогою LSTM при `return_sequences= True`

За допомогою параметру `return_sequences=True` модель можна навчати даних за 24 години за раз:

```
lstm_model = tf.keras.models.Sequential([
    # Shape [batch, time, features] => [batch, time,
lstm_units]
    tf.keras.layers.LSTM(32, return_sequences=True),
    # Shape => [batch, time, features]
    tf.keras.layers.Dense(units=1)
])
```

```
history = compile_and_fit(lstm_model, wide_window)
val_performance['LSTM'] =
lstm_model.evaluate(wide_window.val)
performance['LSTM'] =
lstm_model.evaluate(wide_window.test, verbose=0)
```

```
wide_window.plot(lstm_model)
```

З метою аналізу отриманих результатів виконаємо побудову гістограми (рис. 1.13), що відображає результати прогнозування (точність на тестових наборах даних).

```
x = np.arange(len(performance))
width = 0.3
metric_name = 'mean_absolute_error'
metric_index =
lstm_model.metrics_names.index('mean_absolute_error')
val_mae = [v[metric_index] for v in
val_performance.values()]
test_mae = [v[metric_index] for v in perfor-
mance.values()]

plt.ylabel('mean_absolute_error [T (degC), normal-
ized]')
plt.bar(x - 0.17, val_mae, width, label='Validation')
plt.bar(x + 0.17, test_mae, width, label='Test')
plt.xticks(ticks=x, labels=performance.keys(),
rotation=45)
_ = plt.legend()
```

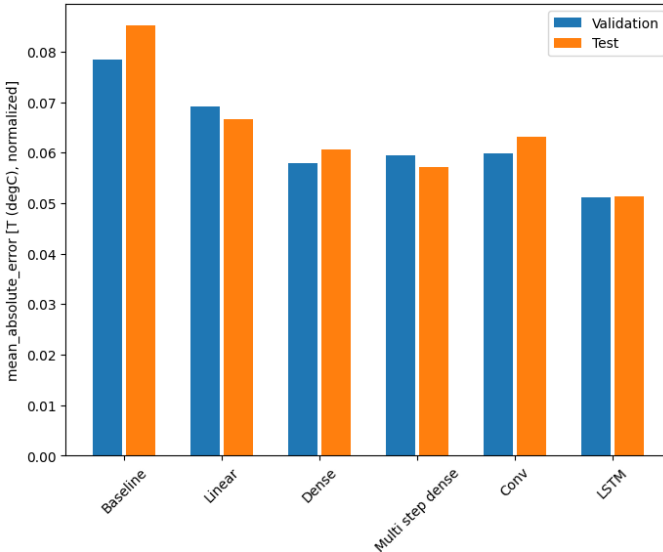


Рисунок 1.13 – Точність роботи різних методів на тестових наборах даних

Для текстового відображення точності різних моделей використовуємо такий програмний код:

```
for name, value in performance.items():
    print(f'{name:12s}: {value[1]:0.4f}')
```

Отримані результати:

```
Baseline      : 0.0852
Linear        : 0.0666
Dense         : 0.0573
Multi step dense: 0.0586
Conv          : 0.0577
LSTM          : 0.0518
```

Як видно, моделі на основі нейронних мереж показали кращі результати у порівнянні з моделлю Baseline та лінійною моделлю.

Для більш детального ознайомлення з побудовою прогностичних моделей з декількома виходами (Multi-output models) та багатокрокових моделей (Multi-step models) рекомендовано ознайомитися з джерелом [7].

## 1.4 Завдання на лабораторну роботу

1.4.1 Ознайомитися з основними теоретичними відомостями та рекомендованою літературою за темою роботи.

1.4.2 Обрати вхідні дані, подані у вигляді часових послідовностей, для подальшої їх обробки та аналізу.

Як вхідні дані можна обрати, наприклад, такі:

– дані з відомих репозиторіїв. Наприклад, з репозиторію UCI Machine Learning Repository:

<https://archive.ics.uci.edu/ml/datasets.php?format=&task=&att=&area=&numAtt=10to100&numIns=&type=ts&sort=nameUp&view=table>

– дані з предметної області, що є добре відомою для студента;  
– спеціально згенеровані вхідні дані;  
– будь-які інші дані, які подані у вигляді часових послідовностей.

1.4.3 Розробити програмне забезпечення для прогнозування часових рядів.

1.4.3.1 Для створення програмного забезпечення обрати мову програмування та інші засоби розробки (програмні бібліотеки та ін.). Студент не обмежується у виборі мови програмування та інших засоби розробки програмного забезпечення.

1.4.3.2 Проаналізувати обрані вхідні дані та предметну область, визначити вимоги до програмного забезпечення (формати подання вхідних даних, необхідність попередньої обробки даних, необхідність поділу множини даних на навчальну та тестову вибірки даних, використовувати математичні моделі для прогнозування часових рядів, необхідність подання результатів у вигляді графіків та ін.).

1.4.3.3 Здійснити проєктування архітектури програмного забезпечення для прогнозування часових рядів.

1.4.3.4 Виконати конструювання програмного забезпечення.

1.4.3.5 Виконати тестування розробленого програмного забезпечення. Здійснити прогнозування на основі обраної послідовності вхідних даних за допомогою різних методів (хоча б двох). Результати прогнозування подати у вигляді таблиці.

1.4.4. Зробити висновки за роботою.

1.4.5. Оформити звіт з роботи.

1.4.6. Відповісти на контрольні питання.

## 1.5 Зміст звіту

- назва і мета роботи;
- фрагмент вхідних даних та їх короткий опис;
- текст розробленого програмного забезпечення;
- результати аналізу вхідних даних за допомогою розробленого програмного забезпечення;
- висновки, що відображують результати виконання лабораторної роботи.

## 1.6 Контрольні запитання

- 1.6.1 Що таке часовий ряд?
- 1.6.2 Наведіть приклади даних, поданих у вигляді часових рядів.
- 1.6.3 У яких прикладних областях використовуються часові ряди?
- 1.6.4 Що таке прогнозування часових рядів?
- 1.6.5 З якою метою виконують попередню обробку даних до синтезу прогнозуючої моделі?
- 1.6.6 Які фактори впливають на якість прогнозування часових рядів?
- 1.6.7 Що таке тренди?
- 1.6.8 Які групи методів використовуються для прогнозування часових рядів?
- 1.6.9 Поясніть метод декомпозиції часових рядів.
- 1.6.10 Як будуються регресійні моделі часових рядів?
- 1.6.11 Що таке експоненціальне згладжування?
- 1.6.12 Модель ARIMA для прогнозування часових рядів.
- 1.6.13 Які види нейронних мереж доцільно використовувати для прогнозування часових рядів?
- 1.6.14 Яким чином використовуються багатошарові перцептрони для прогнозування часових рядів?
- 1.6.15 Яким чином використовуються згорткові нейронні мережі для прогнозування часових рядів?
- 1.6.16 Яким чином використовуються рекурентні нейронні мережі для прогнозування часових рядів?
- 1.6.17 З якою метою виконується нормалізація даних при синтезі моделей для прогнозування часових рядів?

1.6.18 Що таке вікна даних? Як відбувається перетворення вхідного набору даних до формату, зручного для синтезу прогностичної моделі часового ряду?

1.6.19 Як програмно створити лінійну модель для прогнозування часових рядів?

1.6.20 Порівняйте ефективність застосування різних типів моделей для прогнозування часових рядів.



## 2 ЛАБОРАТОРНА РОБОТА № 2 КОМП'ЮТЕРНИЙ ЗІР. КЛАСИФІКАЦІЯ ТА РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

### 2.1 Мета роботи

Навчитися реалізовувати програмні проєкти у сфері інженерії програмного забезпечення, пов'язані з необхідністю обробки зображень.

### 2.2 Короткі теоретичні відомості

Однією з важливих галузей штучного інтелекту є комп'ютерне бачення (Computer Vision, CV). **Комп'ютерне бачення** (комп'ютерний зір) – це наука про комп'ютери та програмні системи, які можуть розпізнавати та розуміти зображення та сцени. Комп'ютерне бачення дозволяє комп'ютеру імітувати людський зір і приймати рішення на основі побачених сцен [9], [10]. Комп'ютерне бачення також складається з різних аспектів, таких як розпізнавання зображень (image recognition), виявлення об'єктів (object detection), створення зображень (image generation), збільшення якості зображень за допомогою методів надроздільної здатності (image super-resolution) тощо [9].

#### 2.2.1 Розпізнавання зображень

**Розпізнавання зображень** стосується аналізу пікселів і шаблонів зображення з метою розпізнавання зображення як окремого об'єкту [10]. Поширеним прикладом розпізнавання зображень є оптичне розпізнавання символів. Сканер може ідентифікувати символи на зображенні, щоб перетворити текст із зображення в текстовий файл. За допомогою того самого процесу можна розпізнавати, наприклад, текст номерного знака на зображенні [10].

Для розроблення програмного забезпечення для розпізнавання зображень необхідно синтезувати розпізнавальну модель на основі наявних даних. Процес **створення моделі розпізнавання зображень** не відрізняється від процесу моделювання у машинному навчанні [10], основні етапи цього процесу наведено нижче.

Етап 1. Витягання піксельних функцій із зображення (Extract pixel features from an image).

На цьому етапі виконується оцифровка зображення з реального світу, внаслідок чого із зображення витягується велика кількість характеристик (ознак). Внаслідок цього отримується цифрове зображення, яке складається з пікسلів (рис.2.1). Кожен піксель представляється числом або набором чисел (діапазон цих чисел називається глибиною кольору або бітовою глибиною). Глибина кольору вказує на максимальну кількість потенційних кольорів, які можна використовувати в зображенні. У 8-бітному сірому зображенні кожен піксель має одне значення в діапазоні від 0 до 255. Більшість зображень використовують 24-бітний колір або вище. Зображення з роздільною здатністю  $1024 \times 768$  являє собою сітку з 1024 стовпцями та 768 рядками, яка, отже, містить  $1024 \times 768 = 0,78$  мегапікселя [10].

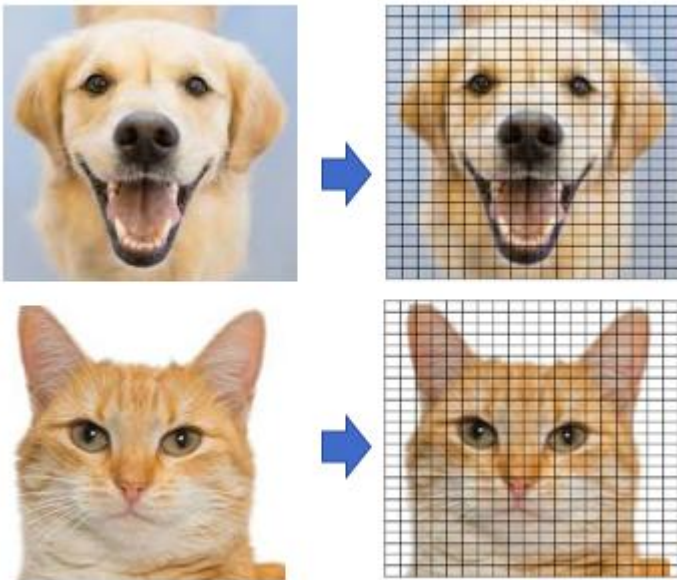


Рисунок 2.1 – Витягання піксельних функцій із зображення [10]

Етап 2. Підготовка множини зображень з мітками для навчання моделі (формування навчальної вибірки).

На цьому етапі створюється велика кількість зображень, що відносяться до різних категорій (міток, класів), наприклад, «собака»,

«кіт», «риба», «птах» та ін. (рис. 2.2). Чим більше зображень буде використано для кожної категорії, тим краще можна навчити модель визначати відповідну категорію (клас) зображення. На цьому етапі відомою є категорія, до якої належить кожне зображення [10]. Ці дані (зображення та категорії, до яких вони відносяться) будуть використовуватися на наступному етапі для навчання моделі.

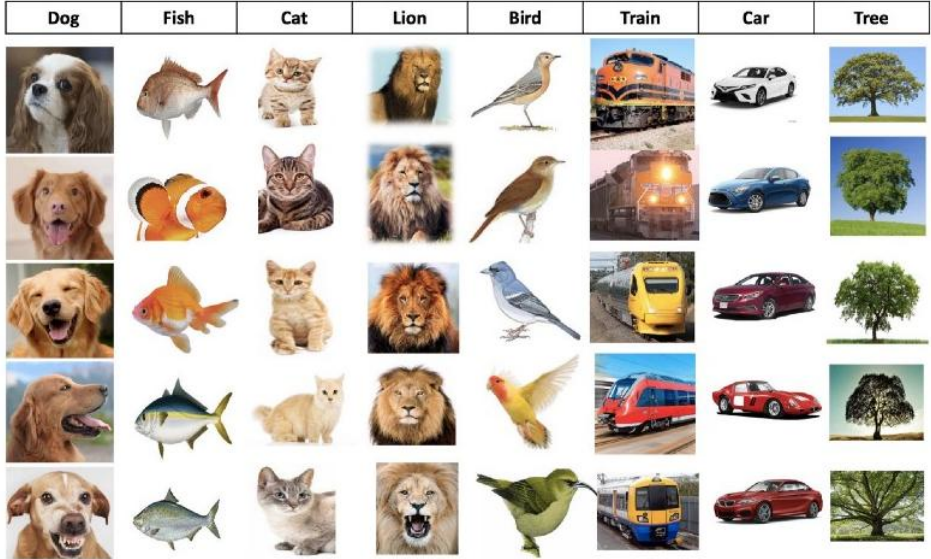


Рисунок 2.2 – Формування навчальної вибірки [10]

Етап 3. Моделювання (синтез розпізнавальної моделі).

На цьому етапі відбувається синтез моделі (підбір значень її параметрів) шляхом її навчання до класифікації наявних зображень (зображень, поданих у навчальній вибірці). На рис.2.3 показано, як модель навчається за допомогою попередньо позначених зображень (зображень, для яких відомими є категорії, класи). Зображення у витягнутих формах надходять на вхідну сторону моделі, а мітки (категорії, класи) – на вихідну. Мета тут полягає в тому, щоб навчити моделі (як правило, нейронні мережі) таким чином, щоб зображення зі своїми характеристиками, що надходять із вхідних даних, відповідало мітці праворуч [10].

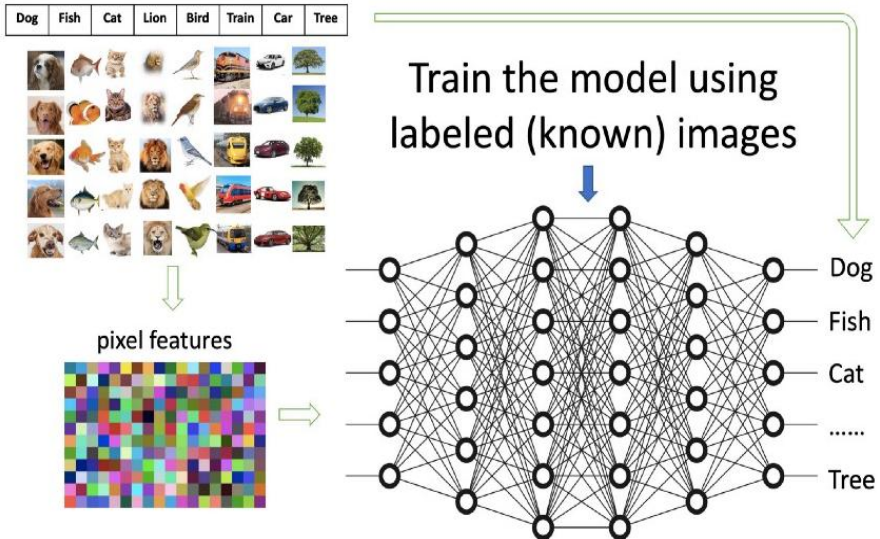


Рисунок 2.3 – Навчання розпізнавальної моделі [10]

Етап 4. Використання синтезованої моделі для розпізнавання нових зображень.

Коли модель навчена, її можна використовувати для розпізнавання невідомого зображення. На рис. 2.4 показано, що нове зображення розпізнається як зображення собаки [10].

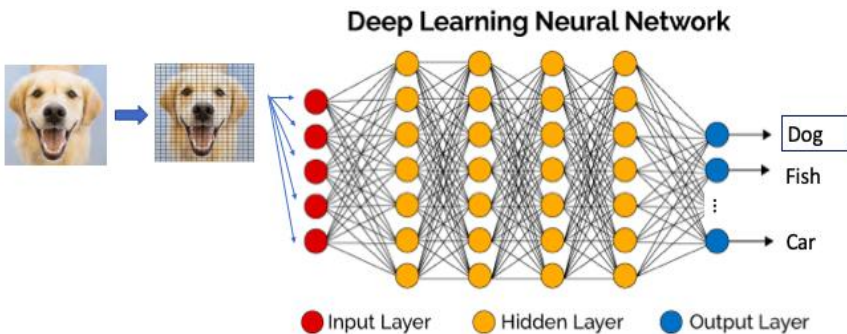


Рисунок 2.4 – Використання синтезованої моделі для розпізнавання нових зображень [10]

Як базис для створення розпізнавальних моделей ефективно застосовуються **згорткові нейронні мережі** (Convolution Neural Networks, CNN), які широко застосовуються для класифікації зображень, виявлення об'єктів або розпізнавання зображень (рис. 2.5) [10].

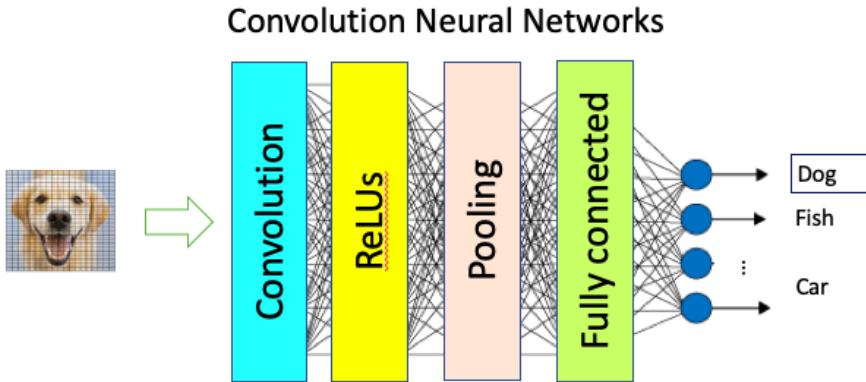


Рисунок 2.5 – Застосування згорткових нейронних мереж розпізнавання зображень [10]

У мережах CNN існує чотири основні типи шарів: згортка (convolution), ReLU, об'єднання (pooling) та повнозв'язані шари (fully connected), як показано на рис. 2.5. Для пояснення роботи цих шарів розглянемо приклад застосування згорткових нейронних мереж для розпізнавання зображень рукописних символів (чисел від 0 до 9) у вигляді зображень у тонах сірого вільної форми [10].

Шар згортки (convolution layer). Першим кроком, який виконують мережі CNN, є створення багатьох маленьких елементів, які називаються ознаками (features), як-от квадрати розміром 2x2. Кожна ознака характеризує певну форму вихідного зображення. При скануванні зображення за допомогою кожної ознаки розраховується відповідна оцінка частини зображення. Якщо збіг є ідеальним, у цієї частини зображення (наприклад, квадрату розміром 2x2), їй присвоюється у відповідність високий бал. Якщо збіг є низьким (або взагалі відсутній), оцінка відповідності також є низькою або нульовою. Цей процес отримання балів (scores) називається фільтрацією (filtering) [10].

На рис. 2.6 показано три функції. Кожна функція створює відфільтроване зображення з високими та низькими оцінками під час ска-

нування вихідного зображення. Наприклад, червоне поле знайшло чотири області на оригінальному зображенні, які ідеально збігаються з об'єктом, тому для цих чотирьох областей оцінки високі. Рожеві прямокутники – це області, які певною мірою збігаються. Акт перевірки кожного можливого збігу шляхом сканування вихідного зображення називається згорткою. Відфільтровані зображення складаються разом, щоб стати шаром згортки [10].

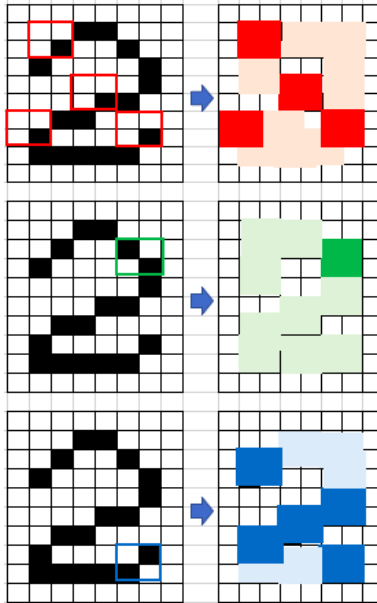


Рисунок 2.6 – Перетворення зображень за допомогою шару згортки [10]

Шар ReLU (Rectified Linear Unit) виправляє будь-яке від'ємне значення до нуля, щоб гарантувати, що нейронна мережа буде працювати коректно. Математично функція ReLU являє собою виправлену лінійну (кусково-лінійну) функцію активації, яка повертає вхідне значення без змін, якщо воно є додатнім, в іншому випадку повертається нульове значення [10].

Шар Max Pooling (максимального об'єднання) відповідає за зменшення розміру зображення. На рис. 2.7 вікно 2x2 сканує кожне з відфільтрованих зображень і призначає максимальне значення цього вік-

на  $2 \times 2$  рамці  $1 \times 1$  у новому зображенні. Як показано на рисунку, максимальне значення в першому вікні  $2 \times 2$  є високим балом (позначеним червоним кольором), тому високий бал призначається полю  $1 \times 1$  нового зображення. Поле  $2 \times 2$  переміщується до другого вікна, де є високий бал (червоний) і низький бал (рожевий), тому високий бал призначається наступному полю  $1 \times 1$ . Наступному полю призначається світло червоний колір. Після об'єднання створюється новий стек менших відфільтрованих зображень [10].

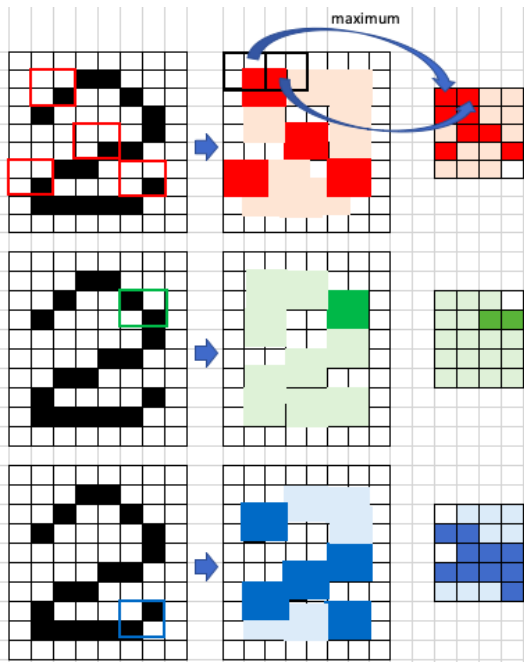


Рисунок 2.7 – Перетворення зображень за допомогою шару Max Pooling [10]

Повноз'язний шар (фінальний шар). На цьому шарі відбувається розділення менш відфільтрованих зображень та складання їх в один список, як показано на рис.2.8. Кожне значення в єдиному списку передбачає ймовірність для кожного з кінцевих значень 1, 2, ... і 0. Ця частина є такою ж, як вихідний шар у звичайних (багатошарових) нейронних мережах. У нашому прикладі цифра «2» отримує найвищу

загальну оцінку з усіх вузлів єдиного списку. Тому CNN розпізнає оригінальне зображення рукописного тексту як «2» [10].

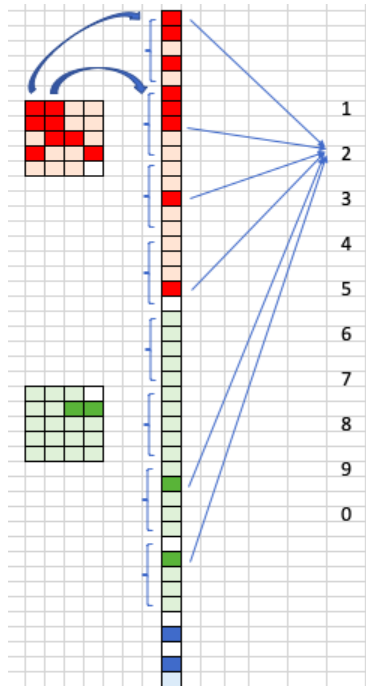


Рисунок 2.8 – Робота останнього, повноз'язного шару при розпізнаванні зображень [10]

Відзначимо, що багатошарові нейронні мережі складають вихідне зображення у список і перетворюють його на вхідний шар. Інформація між сусідніми пікселями при цьому може не зберігатися. На відміну від цього, CNN створює шар згортки, який зберігає інформацію про сусідні пікселі, тому розпізнавання за допомогою CNN є більш ефективним [10].

Окрім нейромоделей типу CNN для розпізнавання зображень також ефективно можуть використовуватися попередньо навчені глибокі моделі (deep learning models), які є у відкритому доступі, зокрема, Xception, VGG16, VGG19, ResNet50, InceptionV3, InceptionResNetV2, MobileNet, DenseNet, NASNet і MobileNetV2 [11].



Ці моделі можна використовувати для розпізнавання та прогнозування, виділення ознак і точного налаштування.

Також у відкритому доступі є досить велика база даних зображень ImageNet, яку можна завантажити для дослідницьких цілей при розв'язанні різних завдань обробки зображень [12].

### **2.2.2 Виявлення об'єктів на графічних зображеннях**

Виявлення об'єктів (object detection) означає здатність комп'ютерних і програмних систем знаходити об'єкти на зображенні/сцені та ідентифікувати кожен об'єкт. Виявлення об'єктів широко використовується для виявлення обличчя на зображеннях, виявлення транспортних засобів, підрахунку пішоходів, обробки вебзображень, у системах безпеки та програмних засобів для автомобілів без водія. Виявлення об'єктів можна використовувати багатьма способами в багатьох галузях практики [9].

Ранні реалізації виявлення об'єктів передбачали використання класичних алгоритмів, подібних до тих, що підтримуються в OpenCV, популярній бібліотеці комп'ютерного зору. Однак ці класичні алгоритми не могли досягти достатньої продуктивності для роботи в інших умовах. У наш час для виявлення об'єктів використовуються більш високоточні алгоритми і моделі, зокрема як R-CNN, Fast-RCNN, Faster-RCNN, RetinaNet, SSD і YOLO. Використання цих методів і алгоритмів, заснованих на нейронних мережах та глибокому навчанні, яке також базується на машинному навчанні, потребує великої кількості математичних розрахунків [9].

Тому розроблено багато програмних бібліотек, що дозволяють розробникам програмного забезпечення легко інтегрувати сучасні технології комп'ютерного зору у розроблювані програми, використовуючи лише кілька рядків коду. Роботу з такими бібліотеками описано, наприклад, у джерелах [9], [13]–[19].

## **2.3 Приклад розробки програмного забезпечення для обробки зображень**

Як приклади, будемо використовувати програмні коди, подані у джерелах [9], [13]–[19].

Програмне забезпечення буде розроблюватися з використанням бібліотеки tensorflow та мови Python.

### **2.3.1 Розробка програмного забезпечення для розпізнавання графічних об'єктів**

Розглянемо навчання простої згорткової нейронної мережі (CNN) для розпізнавання (класифікації) зображень з набору CIFAR [13]. Спочатку імпортуємо необхідні для роботи програмного забезпечення бібліотеки:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np
```

Виконаємо завантаження набору даних CIFAR10, який містить 60 000 кольорових зображень, кожне з яких відноситься до одного з 10 класів, по 6 000 зображень у кожному класі. Набір даних поділено на 50 000 навчальних зображень та 10 000 тестових зображень. Класи взаємовиключні і з-поміж них немає перетину.

```
(train_images, train_labels), (test_images,
test_labels) = datasets.cifar10.load_data()
```

Збережемо навчальні та тестові даних у файлах train.npz та test.npz для можливості їх подальшого завантаження

```
np.savez("train.npz", train_images=train_images,
train_labels=train_labels)
np.savez("test.npz", test_images=test_images,
test_labels=test_labels)
```

При необхідності повторного використання набору CIFAR10 завантажити збережені раніше дані можна таким чином:

```
data = np.load("train.npz")
train_images = data['train_images']
train_labels = data['train_labels']
```

```
data = np.load("test.npz")
test_images = data['test_images']
test_labels = data['test_labels']
```

Нормалізація піксельних значень у межах від 0 до 1:

```
train_images, test_images = train_images / 255.0,  
test_images / 255.0
```

Щоб переконатися, що набір даних виглядає правильно та є коректним, побудуємо перші 25 зображень із навчального набору та відобразимо назву класу під кожним зображенням:

```
class_names = ['airplane', 'automobile', 'bird', 'cat',  
'deer',  
               'dog', 'frog', 'horse', 'ship', 'truck']
```

```
plt.figure(figsize=(10,10))  
for i in range(25):  
    plt.subplot(5,5,i+1)  
    plt.xticks([])  
    plt.yticks([])  
    plt.grid(False)  
    plt.imshow(train_images[i])  
    plt.xlabel(class_names[train_labels[i][0]])  
plt.show()
```

В результаті отримаємо рисунок, на якому відображено перші 25 зображень з завантаженого набору (рис. 2.9).

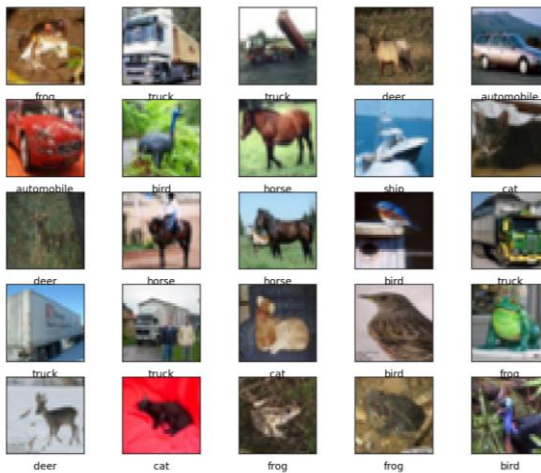


Рисунок 2.9 – Фрагмент вхідних даних

На наступному етапі виконаємо створення згорткової нейронної мережі.

Перші шість рядків коду визначають базу згортки за допомогою загального шаблону: стек шарів Conv2D і MaxPooling2D. Як вхідні дані CNN приймає тензори (набори даних) форми (висота\_зображення, ширина\_зображення, канали\_кольору). У цьому прикладі модель CNN буде обробляти вхідні дані форми input\_shape=(32, 32, 3), що є форматом зображень CIFAR (32 – висота, 32 – ширина, 3 – кількість кольорових каналів, у даному випадку три, оскільки використовується кольорова модель R,G,B).

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Виведемо архітектуру моделі на екран:

```
model.summary()
```

Як результат, отримаємо таку архітектуру моделі model:

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36928
Total params: 56,320		
Trainable params: 56,320		
Non-trainable params: 0		

Як видно, результатом кожного шару Conv2D і MaxPooling2D є тривимірний тензор форми (висота, ширина, канали). Розміри ширини

та висоти мають тенденцію до зменшення в міру заглиблення в мережу. Кількість вихідних каналів для кожного рівня Conv2D контролюється першим аргументом (наприклад, 32 або 64). Як правило, коли ширина та висота зменшуються, ви можете дозволити собі (з точки зору обчислень) додати більше вихідних каналів у кожен шар Conv2D.

Щоб завершити модель, необхідно передати останній вихідний тензор із згорткової бази (форми (4, 4, 64)) в один або декілька Dense-шарів (звичайних шарів нейронів, аналогічних тим, які використовуються у багатшарових нейромоделях) для виконання класифікації. Dense-шари приймають вектори як вхідні дані (які є одновимірними), тоді як поточний вихід є 3D-тензором. Тому спочатку необхідно звести (або розгорнути) 3D-вихід до одновимірного (за допомогою шару Flatten), а потім додати один або декілька Dense-шарів зверху. Набір даних CIFAR має 10 вихідних класів, тому на останньому шарі необхідно використовувати 10 виходів (10 нейронів):

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Використовуючи команду `model.summary()`, виведемо архітектуру моделі на екран:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65600
dense_1 (Dense)	(None, 10)	650

```
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

Як видно, останній вихідний тензор згортки (форми (4, 4, 64)) було зведено у вектори форми (1024) перед проходженням через два Dense-шари.

Після цього виконується компіляція на навчання згорткової нейронної мережі:

```
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from
_logits=True), metrics=['accuracy'])
```

```
history = model.fit(train_images, train_labels,
epochs=10, steps_per_epoch=100, valida-
tion_data=(test_images, test_labels))
```

На останньому етапі виконується оцінювання моделі та відображення результатів її роботи

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label =
'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
```

```
plt.show()
```

```
test_loss, test_acc = model.evaluate(test_images,
test_labels, verbose=2)
```

```
print("Точність моделі на тестових даних: ", test_acc)
```

Як результат, відображається точність моделі на тестових даних. Враховуючи стохастичний характер синтезу нейромоделі, точність при кожному запуску програми буде різною.

Також відображається графік залежності точності (Accuracy) розпізнавання зображень за допомогою моделі від номеру ітерації (епохи, Epoch) для навчальних та перевірочних даних (рис. 2.10).

Після синтезу моделі, її можна використовувати для класифікації зображень, що не входили у навчальну вибірку.

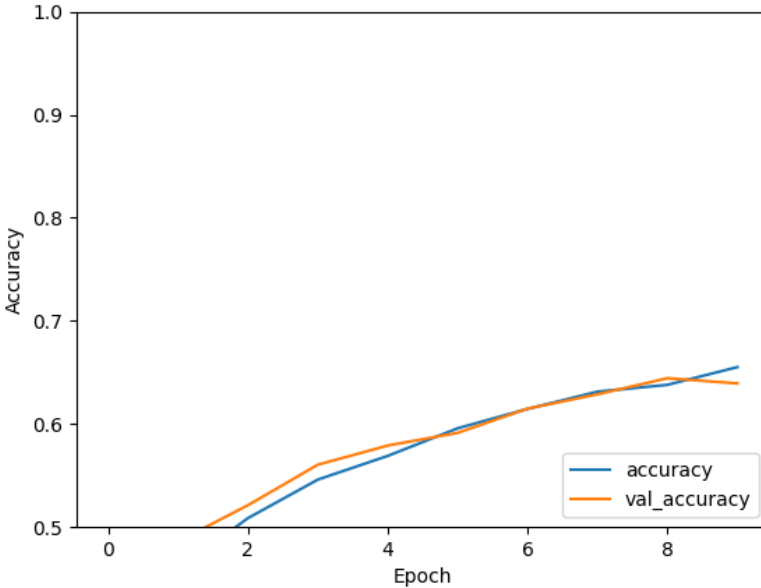


Рисунок 2.10 – Графік залежності точності розпізнавання зображень за допомогою синтезованої моделі від номеру ітерації

Також рекомендовано ознайомитися з іншим прикладом використання згорткової нейронної мережі для розпізнавання зображень, поданим у джерелі [14].

### 2.3.2 Розробка програмного забезпечення для пошуку об'єктів на графічних зображеннях

У наш час розроблено багато моделей, які програмно реалізовані та є загально доступними, для пошуку об'єктів (object detection) на графічних зображеннях. Задача синтезу таких моделей і доволі складною та вимагає багато часу та обчислювальних ресурсів комп'ютера, тому будемо використовувати вже існуючі програмні бібліотеки та моделі пошуку об'єктів.

Розробимо програмне забезпечення для пошуку об'єктів на графічних зображеннях за допомогою бібліотеки `opencv-python`. Отже для роботи програми необхідно встановити деякі бібліотеки (якщо цього не було зроблено раніше):

```
pip install opencv-python
pip install matplotlib
```

Завантажимо програмні модулі, які необхідні для роботи програми для пошуку облич на графічних зображеннях, а також очей на знайдених зображеннях:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
```

Відзначимо, що для роботи програми необхідно завантажити попередньо підготовлені класифікатори OpenCV, які містять інформацію для розпізнавання відповідних об'єктів (обличчя, очі та ін.) на зображеннях. У даному випадку заздалегідь підготовлені файли `haarcascade_frontalface_default.xml` та `haarcascade_eye.xml`, які знаходяться у відкритому доступі та містять інформацію для детекції облич на зображеннях, а також очей на обличчях, відповідно [18]:

```
face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.
xml')
eye_cascade =
cv2.CascadeClassifier('haarcascade_eye.xml')
```

Завантажимо зображення з файлу `photo.jpg` та виконаємо його перетворення тону сірого:

```
img = cv2.imread('photo.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Знаходимо обличчя на зображенні за допомогою класифікатора `face_cascade`:

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

Знаходимо очі на попередньо знайдених обличчях:

```
for (x,y,w,h) in faces:
    cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
```



```
cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)
```

Відображаємо результат:

```
cv2.imshow('img', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Після запуску програми виводиться зображення з файлу photo.jpg з позначеними на ньому обличчями та очима людей.

Зауважимо, що якість пошуку об'єктів на зображеннях залежить не тільки від самих зображень, а і від якості попередньо синтезованих детекторів.

Також програмне забезпечення для пошуку об'єктів на графічних зображеннях може бути розроблено за допомогою бібліотеки ImageAI [9], написаної на мові Python з використанням засобів бібліотеки tensorflow та деяких інших бібліотек Python.

Отже для роботи з бібліотекою ImageAI необхідно встановити деякі інші бібліотеки (якщо цього не було зроблено раніше):

```
pip install tensorflow
pip install TorchVision keras numpy pillow scipy h5py
matplotlib opencv-python keras-resnet
```

Після цього встановимо бібліотеку ImageAI:

```
pip install imageai --upgrade
```

Виконаємо завантаження файлу з вже синтезованою моделлю RetinaNet для пошуку об'єктів на графічних зображеннях [19]:

[https://github.com/OlafenwaMoses/ImageAI/releases/download/essentials-v5/resnet50\\_coco\\_best\\_v2.1.0.h5/](https://github.com/OlafenwaMoses/ImageAI/releases/download/essentials-v5/resnet50_coco_best_v2.1.0.h5/)

Створимо файл LR2\_imageDetection.py з таким програмним кодом:

```
from imageai.Detection import ObjectDetection

import os

execution_path = os.getcwd()

detector = ObjectDetection()
```

```

detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path ,
"resnet50_coco_best_v2.1.0.h5"))
detector.loadModel()
detections =
detector.detectObjectsFromImage(input_image=os.path.joi
n(execution_path , "image.jpg"),
output_image_path=os.path.join(execution_path ,
"imagenew.jpg"))

for eachObject in detections:
    print(eachObject["name"] , " : " , eachObject["percentage_probability"] )

```

Після запуску код у консолі друкуються результати роботи програми: назва виділеного об'єкту (name, наприклад, person, bus, car) та відсоток ймовірності (percentage\_probability), що виділений об'єкт відноситься до конкретного типу. Коли результат буде надруковано на консолі (програма виконає обробку заданого файлу), можна перейти до папки, у якій знаходиться програмний файл LR2\_imageDetection.py, де буде знаходитися нове зображення з виділеними на ньому об'єктами.

## 2.4 Завдання на лабораторну роботу

2.4.1 Ознайомитися з основними теоретичними відомостями та рекомендованою літературою за темою роботи.

2.4.2 Обрати вхідні дані, подані у графічних зображень, для подальшої їх обробки та аналізу.

Як вхідні дані можна обрати, наприклад, такі:

– дані з відомих репозиторіїв. Наприклад, з репозиторію TensorFlow [20]:

<https://www.tensorflow.org/datasets> ;

– дані з предметної області, що є добре відомою для студента;

– спеціально згенеровані вхідні дані;

– будь-які інші дані, які подані у вигляді графічних зображень.

2.4.3 Розробити програмне забезпечення для вирішення однієї з задач комп'ютерного зору (розпізнавання зображень, пошуку об'єктів на графічних зображеннях та ін.).

2.4.3.1 Для створення програмного забезпечення обрати мову програмування та інші засоби розробки (програмні бібліотеки та ін.). Студент не обмежується у виборі мови програмування та інших засоби розробки програмного забезпечення.

2.4.3.2 Проаналізувати обрані вхідні дані та предметну область, визначити вимоги до програмного забезпечення (формати подання вхідних даних, необхідність попередньої обробки даних, необхідність поділу множини даних на навчальну та тестову вибірки даних, використовувані математичні моделі для обробки зображень, необхідність подання результатів у вигляді графіків та ін.).

2.4.3.3 Здійснити проєктування архітектури програмного забезпечення для обробки зображень.

2.4.3.4 Виконати конструювання програмного забезпечення.

2.4.3.5 Виконати тестування розробленого програмного забезпечення. Подати результати тестування розробленого програмного забезпечення для обробки зображень у зручному для подальшого аналізу вигляді.

2.4.4. Зробити висновки за роботою.

2.4.5. Оформити звіт з роботи.

2.4.6. Відповісти на контрольні питання.

## **2.5 Зміст звіту**

- назва і мета роботи;
- фрагмент вхідних даних та їх короткий опис;
- текст розробленого програмного забезпечення;
- результати аналізу вхідних даних за допомогою розробленого програмного забезпечення;
- висновки, що відображують результати виконання лабораторної роботи.

## **2.6 Контрольні запитання**

- 2.6.1 Поясніть поняття «Комп'ютерне бачення».
- 2.6.2 Які задачі вирішує комп'ютерне бачення?
- 2.6.3 У чому полягає суть розпізнавання зображень?
- 2.6.4 Проаналізуйте етапи створення моделі розпізнавання зображень.

2.6.5 Яким чином відбувається формування навчальної вибірки для синтезу моделі для розпізнавання зображень?

2.6.6 У чому полягає етап синтезу розпізнавальної моделі для обробки зображень?

2.6.7 Як використовуються синтезовані моделі для розпізнавання нових зображень?

2.6.8 Які типи шарів нейронів використовуються у згорткових нейронних мережах?

2.6.9 Для чого призначений шар згортки у згорткових нейромоделях?

2.6.10 Яким чином будується останній шар згорткової нейронної мережі для розпізнавання зображень? Скільки нейронів містить такий шар?

2.6.11 Чому для розпізнавання зображень більш доцільно використовувати згорткові нейронні мережі, ніж багат шарові перцептрони?

2.6.12 Які типи попередньо навчених глибоких моделей можуть використовуватися для розпізнавання зображень?

2.6.13 Що таке виявлення (детекція) об'єктів на зображеннях?

2.6.14 У яких практичних застосуваннях використовується виявлення об'єктів на зображеннях?

2.6.15 Які моделі доцільно використовувати для виявлення об'єктів на зображеннях?

## **3 ЛАБОРАТОРНА РОБОТА № 3 ОБРОБКА ПРИРОДНОЇ МОВИ ТА ТЕКСТОВОЇ ІНФОРМАЦІЇ**

### **3.1 Мета роботи**

Навчитися реалізовувати програмні проєкти у сфері інженерії програмного забезпечення, пов'язані з необхідністю обробки текстової інформації.

### **3.2 Короткі теоретичні відомості**

Обробка природної мови (Natural Language Processing, NLP) – це технологія, яка допомагає комп'ютерам розуміти природну людську мову; галузь штучного інтелекту, яка має на меті надати машинам здатність читати, розуміти та виводити (генерувати) людську мову. До завдань обробки природної мови відносять визначення настроїв у тексті, машинний переклад, перевірку орфографії та інші завдання [21]. У процесі обробки природної мови часто використовуються методи лінгвістики, існує чотири етапи обробки мови: морфологія (спосіб творення слів і їх зв'язок з іншими словами), синтаксис (як ці слова поєднуються в реченні), семантика (як значення слів розкривається через граматику та лексичне значення), прагматика (значення слів у контексті).

У більшості випадків у процесі обробки природної мови її спочатку перетворюють на структуру, яку комп'ютер здатний прочитати. Наприклад, позначають частини мови у тексті, видаляють певні слова-заповнювачі, що не додають значного змісту тексту (зокрема, можуть видалятися артиклі та деякі інші слова, наприклад, а, the, to, etc.). Також попередньо над текстом можуть виконувати лематизацію (Lemmatization), стеммінг (Stemming), токенизацію (Tokenization) та інші дії.

Обробку природної мови можна використовувати як фільтр фейкових новин, для підтримки клієнтів та аналізу відгуків, аналізу настроїв у соціальних мережах, фільтрації електронної пошти (пошуку спаму) та ін. [21].

### 3.2.1 Класифікація текстів

Класифікація текстів – це техніка машинного навчання, яка призначає набір попередньо визначених категорій деякому тексту. Текстові класифікатори можна використовувати для організації, структурування та класифікації майже будь-якого типу тексту – від документів, медичних досліджень і файлів, а також у всьому Інтернеті [22].

Наприклад, нові статті можна впорядкувати за темами; розмови в чаті можна організувати за мовами; згадування брендів можна організовувати за настроями; і так далі [22].

Класифікація текстів є одним із основних завдань обробки природної мови з широким застосуванням, таким як аналіз настроїв, маркування тем, виявлення спаму та виявлення намірів.

Для створення програмного забезпечення класифікації текстів, як правило, необхідно виконати етапи, наведені нижче [23]-[24].

Етап 1. Отримання високоякісного набору даних. У випадку класифікації тексту використовуються алгоритми машинного навчання з учителем (supervised machine learning algorithms), тому для навчання моделі необхідні позначені даними (labeled data). Таким чином дані являють собою множину текстів, кожен з яких позначається певною категорією (міткою, класом, тегом). При цьому кожен текст (наприклад, якийсь коментар або пост у соцмережі) відноситься тільки до одного класу (наприклад, позитивний коментар або негативний коментар). До одного класу можуть відноситися багато текстів [23].

Етап 2. Фільтрація та обробка даних. Оскільки моделі машинного навчання можуть розуміти лише числові значення, для правильного розпізнавання текстових даних моделлю знадобиться токенізація (tokenization) та вбудовування слів (word embedding) у наданий текст.

**Токенізація** – це процес поділу текстових документів на менші частини, які називаються токенами. Токени можуть бути представлені як ціле слово, підслово або окремий символ. Наприклад, токенізувати слово «smarter» можна так:

- токен на рівні слова: smarter;
- токени на рівні підслів: smart-er;
- токени на рівні символів: s-m-a-r-t-e-r.

Токенізація важлива, оскільки моделі класифікації тексту можуть обробляти дані лише на рівні токенів і не можуть розуміти й обробляти цілі речення [23].

У процесі фільтрації алгоритми машинного та глибокого навчання можуть розуміти лише числові значення, тому на етапі попередньої обробки даних також виконується **вбудовування слів** (word embedding) у заданому наборі даних. Вбудовування слів — це процес перетворення деяких слів тексту у вектори дійсних значень, які можуть кодувати значення даного слова. Використовуються такі методи вбудовування слів [23]:

- Word2Vec — метод вбудовування слів без учителя (unsupervised), розроблений Google. Він використовує нейронні мережі для вивчення великих наборів текстових даних. Підхід Word2Vec перетворює кожне слово в заданий вектор;

- GloVe (Global Vector) — модель машинного навчання без учителя для отримання векторних представлень слів. Подібно до методу Word2Vec, алгоритм GloVe відображає слова у значущих просторах, де відстань між словами пов'язана із семантичною подібністю;

- TF-IDF (Term Frequency-Inverse Document Frequency) — це алгоритм вбудовування слів, який оцінює, наскільки важливим є слово в певному документі. TF-IDF присвоює кожному слову заданий бал, щоб вказати його важливість у наборі документів.

Подальша обробка даного необробленого набору даних буде потрібна для того, щоб модель легко засвоювала ці дані. Для цього на етапі попередньої обробки доцільно видалити непотрібні ознаки, відфільтрувати нульові та нескінченні значення тощо. Перетасування всього набору даних допоможе запобігти будь-яким упередженням на етапі навчання [23].

Етап 3. Поділ набору даних на навчальну і тестову множини даних. Часто навчальний набір даних складає 80% (або іншу частку) заданого набору, а тестові дані складають 20% від заданого та використовуються для перевірки алгоритму на точність [23].

В залежності від особливостей даних та програмної реалізації забезпечення для класифікації текстів, другий та третій етапи можуть мінятися місцями або об'єднуватися в один етап.

Етап 4. **Синтез моделі**. На цьому етапі обирається тип та структура моделі. Після цього виконується навчання моделі (підбір її параметрів) із навчальним набором даних. Алгоритм навчання налаштовує модель таким чином, щоб вона класифікувала надані тексти за різними категоріями з мінімальною помилкою [23].

Як базис для синтезу моделей для класифікації текстів можуть використовуватися такі [22]:

- наївний баєсів класифікатор (naive Bayes classifier) – ймовірнісний класифікатор, заснований на застосуванні теореми Байєса з сильними (наївними) припущеннями незалежності між ознаками. При використанні таких методів відбувається визначення ймовірності належності екземпляра (тексту, поданого у вибірці) до одного з заданих класів при (наївному) припущенні про незалежність змінних;

- модель на основі методу опорних векторів (Support vector machine, SVM). SVM намагається побудувати лінію або гіперплощину, яка розділяє простір на два підпростори, таким чином, щоб екземпляри (тексти) з окремих категорій (класів) були розділені частиною простору пошуку, яка є якомога більш ширшою. Таким чином, в одній частині простору пошуку опиняються екземпляри одного класу, в іншій – екземпляри іншого класу, а між цими частинами простору знаходиться роздільна гіперплощина, що максимально віддалена від екземплярів кожного з класів;

- моделі на основі глибокого навчання. Двома основними архітектурними моделями глибокого навчання для класифікації тексту є **згорткові нейронні мережі** (Convolutional Neural Networks, CNN) і **рекурентні нейронні мережі** (Recurrent Neural Networks, RNN). Алгоритми глибокого навчання вимагають набагато більше навчальних даних, ніж традиційні алгоритми машинного навчання, проте у них немає порогу для навчання з навчальних даних, а класифікатори стають кращими, чим більше даних ви їм надаєте.

Також можуть застосовуватися й інші моделі, наприклад, логістична регресія, моделі, засновані на правилах, гібридна модель та ін.

Етап 5. Тестування та перевірка працездатності моделі. На цьому етапі перевіряється якість моделі, використовуючи набір тестових даних, визначений на етапі 3. Тестовий набір даних проганяється через модель, внаслідок чого для кожного тексту з тестового набору отримуються значення відповідних класів або міток. Ці класи порівнюються з фактичними значеннями, на основі чого розраховується точність моделі на тестових даних. Для ефективного тестування моделі тестовий набір даних повинен містити нові тестові випадки (дані, що відрізняються від попереднього навчального набору даних) [23].

Як критерії оцінювання ефективності синтезованої моделі для класифікації текстів можуть використовуватися:



- точність (Assigasy) – відсоток текстів, які були класифіковані правильно (клас, до якого було віднесено текст за допомогою моделі, збігається з фактичним класом);

- помилка (Error) – відсоток текстів, які були класифіковані неправильно (клас, до якого було віднесено текст за допомогою моделі, не збігається з фактичним класом);

- правильність, влучність (Precision) – відсоток текстів, які класифікатор правильно відніс до певного класу, до загальної кількості текстів, які класифікатор відніс до цього класу;

- повнота, чутливість (Recall, sensitivity) – відсоток текстів, які класифікатор відніс (не обов’язково правильно) для певного класу, до загальної кількості текстів, які він мав віднести для даного класу.

- F1-критерій (F1 Score) – середнє гармонійне значення правильності та повноти.

Іноді, у випадку отримання незадовільних результатів тестування моделі (низька точність на тестових даних) виконують налаштування алгоритму машинного навчання. Для цього корегують різні гіперпараметри методу навчання моделі. Гіперпараметр — це параметр, значення якого контролює процес навчання моделі.

### 3.2.2 Генерація текстової інформації

Генерація текстової інформації відноситься до так званого генеративний штучного інтелекту, що дозволяє створювати новий текст, зображення, відео тощо на основі наявних вхідних даних. Генеративний штучний інтелект є одним із головних стратегічних технологічних трендів та має різноманітні застосування в іграх, рекламі, банківській справі, нагляді та охороні здоров’я [25].

Генерація текстової інформації є також галуззю обробки природної мови. Генерування тексту, як правило, виконується на основі моделей глибокого навчання. Як приклад, дослідники тренують генеративні змагальні мережі (Generative adversarial networks, GAN), які є генеративними моделями, що складаються з генератора та дискримінатора й використовуються для створення синтетичних результатів для генерації тексту [25].

Таким чином, для створення програмного забезпечення для генерації текстової інформації необхідно синтезувати відповідну модель, як правило, на основі архітектур глибокого навчання.

Для генерації тексту досить ефективно використовуються такі моделі [26]:

- рекурентні нейронні мережі (Recurrent Neural Networks, RNN);
- нейронні мережі довгокороткочасної пам'яті (Long Short Term Memory, LSTM) і контрольовані рекурентні блоки (Gated Recurrent Unit, GRU);
- двонаправлені рекурентні нейронні мережі (BiDirectional Recurrent Neural Networks, BRNN);
- згорткові нейронні мережі (Convolutional Neural Networks, CNN);
- варіаційні автокодери (Variational Auto-Encoders, VAE);
- генеративні змагальні мережі (Generative Adversarial Networks, GAN).

У наш час розроблено та вже навчено різні моделі штучного інтелекту, які можуть генерувати текст та є досить складними. Однією з таких моделей є GPT (Generative Pre-trained Transformer), або генеративний попередньо навчений трансформатор. Ця мовна модель, створена OpenAI, має різні моделі, включаючи GPT-3. GPT-3 – це досить велика та складна модель із понад 175 мільярдами параметрів, яка навчалася на різних джерелах даних, включаючи книги, статті та сховища коду, щоб створювати реалістичні та схожі на людські тексти. За допомогою GPT-3 можна створювати резюме, відповідати на запитання та робити переклади [25].

Іншим підходом до генерування тексту є моделі на основі шаблону (template-based model). На відміну від GPT-3, ці моделі не працюють незалежно, а проміжні кроки вимагають втручання людини.

У наш час розроблено різні інструменти генерації тексту, які створюють і надають готові шаблони для публікацій у блогах, соціальних мережах, електронних листів, описів продуктів, слоганів тощо.

Текстові генератори застосовується для [25]:

- створення контенту. Текстові генератори можна використовувати для створення будь-якого вмісту, що підтримує бізнес-функції. У маркетингу текстові генератори можуть використовуватися для створення повідомлень у блозі на основі ключових слів і бажаної довжини, створення описів товару на основі даних про його властивості та переваги, генерації постів у соціальних мережах, рекламних кампаніях. Також текстові генератори можуть використовуватися у засобах масової інформації для створення статей для регулярних подій, таких як спортивні матчі;

– узагальнення тексту – створення резюме довгих текстів. Наприклад, створення інформаційних бюлетенів, узагальнення внутрішніх документів компанії, допомога викладачам у підготовці навчального матеріалу шляхом надання їм узагальненого змісту джерел, допомога у огляді літератури у дослідницькому контексті та ін. Також текстові генератори можуть використовуватися у SEO-оптимізації щоб зробити статтю більш оптимізованою для пошукових систем, текстові генератори допомагають у процесі вибору заголовка, мета-опису та ключових слів статті. За допомогою цих інструментів можна виявити кластери тем, які найчастіше шукають, і кількість їхніх ключових слів, а також отримати URL-адреси з найкращим рейтингом для підвищення видимості SEO. Інструменти створення тексту можуть також надавати клієнтам підтримку у вигляді чат-бота в режимі реального часу, а також готувати персоналізовані відповіді служби підтримки клієнтів.

### **3.3 Приклад розробки програмного забезпечення для обробки природної мови**

Як приклади, будемо використовувати програмні коди, подані у джерелах [27]–[31]. Програмне забезпечення буде розроблюватися з використанням бібліотеки tensorflow та мови Python.

#### **3.3.1 Розробка програмного забезпечення для класифікації текстів**

Розглянемо процес створення програмного забезпечення на основі рекурентної нейронної мережі (Recurrent Neural Network, RNN) для класифікації текстів, що являють собою рецензії (позитивні або негативні) на різні фільми [27].

Імпортуємо модулі, необхідні для створення програмного забезпечення:

```
import numpy as np
import tensorflow_datasets as tfds
import tensorflow as tf

tfds.disable_progress_bar()
```

На першому етапі виконується завантаження набору даних. У цьому прикладі використовується набір даних IMDB ([ai.stanford.edu/~amaas/data/sentiment/](http://ai.stanford.edu/~amaas/data/sentiment/)) для бінарної класифікації текстів. Цей набір даних містить рецензії на фільми (відгуки про фільми), подані у вигляді текстових файлів, а також бінарні мітки полярності настрою (sentiment polarity labels) кожної рецензії. Тобто кожна рецензія (текст) у наборі даних IMDB відноситься до одного з двох класів: позитивний (pos) або негативний (neg) настрій. Набір IMDB містить 25 000 рецензій фільмів для навчання та 25 000 для тестування, також є додаткові немарковані дані для використання [27].

```
dataset, info = tfds.load('imdb_reviews',
                           with_info=True,
                           as_supervised=True)
train_dataset, test_dataset = dataset['train'], da-
taset['test']
print(train_dataset.element_spec)
```

Виведемо, як приклад, одну пару значень (текст та мітка) з завантаженого набору даних train\_dataset:

```
for example, label in train_dataset.take(1):
    print('text: ', example.numpy())
    print('label: ', label.numpy())
```

Як результат виводиться текст рецензії та відповідна мітка:

```
text:  b"This was an absolutely terrible movie. Don't
be lured in by Christopher Walken or Michael Ironside.
Both are great actors, but ... I could barely sit
through it."
label:  0
```

На другому етапі створюємо навчальну та тестову вибірки шляхом перемішування екземплярів (текстів), щоб завжди мати довільний порядок текстів, що подаються в мережу:

```
BUFFER_SIZE = 10000
BATCH_SIZE = 64
train_dataset =
train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).pr
efetch(tf.data.AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE)
.prefetch(tf.data.AUTOTUNE)
```

#виводимо перші три тексти у навчальній вибірці та відповідні їм мітки:

```
for example, label in train_dataset.take(1):
    print('texts: ', example.numpy()[:3])
    print()
    print('labels: ', label.numpy()[:3])
```

**На третьому етапі виконуємо попередню обробку текстових даних.** Для цього створюємо текстовий кодувальник для попередньої обробки текстових даних. Для цього у якості першого шару синтезованої моделі будемо використовувати шар **TextVectorization**.

```
VOCAB_SIZE = 1000
encoder = tf.keras.layers.TextVectorization(
    max_tokens=VOCAB_SIZE)
encoder.adapt(train_dataset.map(lambda text, label:
text))
```

Метод `.adapt` визначає словник шару. Виведемо перші 20 токенів. Після доповнення невідомими токенами ('[UNK]') вони сортуються за частотою:

```
vocab = np.array(encoder.get_vocabulary())
print(vocab[:20])
```

Як результат, на екрані отримаємо перші 20 токенів:

```
[' '[UNK]' 'the' 'and' 'a' 'of' 'to' 'is' 'in' 'it' 'i'
'this' 'that' 'br' 'was' 'as' 'for' 'with' 'movie' 'but']
```

Після встановлення словника шар може кодувати текст в індекси. Тензори (набори) індексів для кожного тексту у наборі даних доповнюються нулем до найдовшої послідовності в пакеті. Виведемо тензори індексів для перших трьох текстів у наборі даних:

```
encoded_example = encoder(example)[:3].numpy()
print(encoded_example)
```

Як результат можемо отримати, наприклад, такі набори індексів (оскільки набори даних кожного разу перемішуються, масиви індексів також будуть різними при кожному запуску програми):

```
[[ 10   1 442 ...   0   0   0]
 [ 45   2  61 ...   0   0   0]
 [  6  28   1 ...   0   0   0]]
```

Виведемо перші три тексти в оригінальному вигляді та поновленому вигляді (шляхом перетворення наборів числових індексів у текстові слова з словника):

```
for n in range(3):
    print("Original: ", example[n].numpy())
    print("Round-trip: ", "
".join(vocab[encoded_example[n]]))
    print()
```

Як результат отримаємо текстові рецензії в оригінальному (Original) то поновленому (Round-trip) вигляді. Як приклад, наведемо два варіанти однієї рецензії:

```
Original: b'I voted 3 for this movie because it looks great as
does all of Greenaways output. However it was his usual mix of
...'
Round-trip: i [UNK] 3 for this movie because it looks great as
does all of [UNK] [UNK] however it was his usual [UNK] of ...
```

На четвертому етапі відбувається створення моделі. Враховуючи, що першим шаром моделі є шар TextVectorization, створений на попередньому етапі як змінна encoder, на цьому етапі як перший шар задаємо змінну encoder:

```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        # Use masking to handle the variable sequence
        lengths
        mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64))
,
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

Архітектура рекурентної нейронної мережі для класифікації тексту подана на рис.3.1 [27].

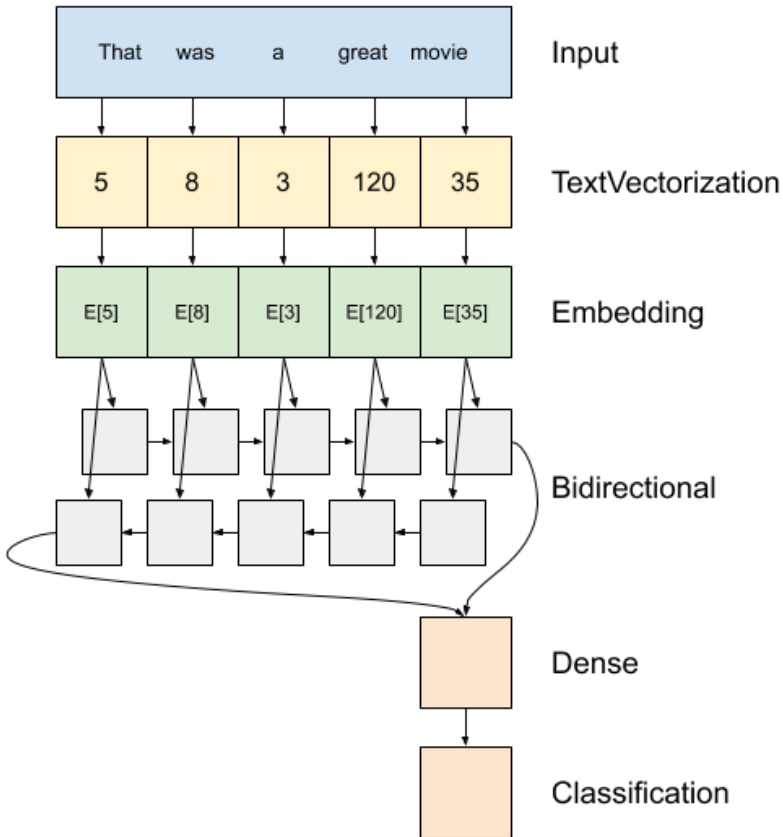


Рисунок 3.1 – Архітектура рекурентної нейронної мережі для класифікації тексту

Як видно з рисунку, перший шар являє собою кодувальник TextVectorization, який перетворює текст на послідовність індексів токенів. Після кодувальника йде шар вбудовування Embedding, який зберігає один вектор на кожне слово. При виклику він перетворює послідовності індексів слів на послідовності векторів. Ці вектори піддаються навчанню. Після навчання (за достатньої кількості даних) слова зі схожими значеннями часто мають схожі вектори. Цей індексний пошук є набагато ефективнішим, ніж еквівалентна операція прохо-

дження вектора з одноразовим кодуванням через шар `tf.keras.layers.Dense`.

Рекурентна нейронна мережа обробляє послідовні вхідні дані шляхом ітерації по елементам. RNN передають вихідні дані з одного часового кроку на свій вхід на наступному часовому кроці. Шар `tf.keras.layers.Bidirectional` також можна використовувати з шаром RNN, поширюючи таким чином вхідні дані вперед і назад через рівень RNN, а потім об'єднуючи кінцевий вихід. Основна перевага двонаправлених RNN (bidirectional RNN) полягає в тому, що сигнал від початку входу мережі не потрібно обробляти протягом кожного часового кроку, щоб вплинути на вихід. Основним недоліком двонаправлених RNN є те, що ви не можете ефективно передавати прогнози (вихідні значення нейронів), оскільки слова додаються в кінці.

Після того, як RNN перетворить послідовність в єдиний вектор, два `Dense`-шари (по 64 та 1 нейрону) виконують деяку остаточну обробку та перетворюють це векторне представлення в один вихід як результат класифікації.

Після створення моделі виконується її **компіляція**:

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer=tf.keras.optimizers.Adam(1e-4),
              metrics=['accuracy'])
```

Після цього здійснюється **навчання моделі** на основі навчальних даних `train_dataset`:

```
history = model.fit(train_dataset, epochs=10,
                   validation_data=test_dataset,
                   validation_steps=30)
```

Тестові дані (`test_dataset`) використовуються для оцінювання якості моделі (точності на тестових даних).

Після навчання моделі можна вивести її точність (Accuracy) та втрати (loss) на екран:

```
test_loss, test_acc = model.evaluate(test_dataset)

print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```



При цьому у якості функції втрат loss можуть використовувати-ся середньоквадратична помилка або інші (кросентропія, логарифмічна функція, ймовірнісна функція).

Створимо функцію для побудови графіків залежності точності синтезованої моделі від номеру ітерації процесу навчання:

```
import matplotlib.pyplot as plt

def plot_graphs(history, metric):
    plt.plot(history.history[metric])
    plt.plot(history.history['val_'+metric], '')
    plt.xlabel("Epochs")
    plt.ylabel(metric)
    plt.legend([metric, 'val_'+metric])
```

Побудуємо графіки залежності точності та втрат від номеру ітерації (рис. 3.2):

```
plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
plot_graphs(history, 'accuracy')
plt.ylim(None, 1)
plt.subplot(1, 2, 2)
plot_graphs(history, 'loss')
plt.ylim(0, None)
plt.show()
```

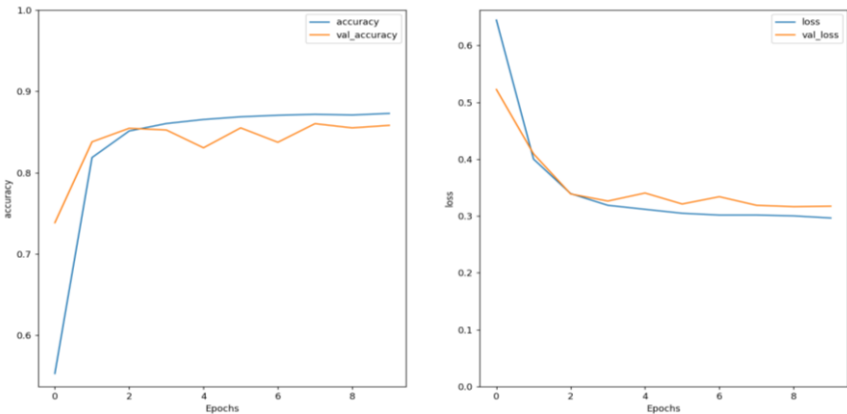


Рисунок 3.2 – Графіки залежності точності та втрат від номеру ітерації

Відзначимо, що враховуючи складність синтезованої моделі для класифікації текстової інформації, процес її побудови може зайняти доволі багато часу (порядка декількох годин).

Також рекомендовано ознайомитися з іншим прикладом класифікації текстової інформації, поданим у джерелі [28].

### 3.3.2 Розробка програмного забезпечення для генерації текстів

Розглянемо процес створення програмного забезпечення для генерації текстів на основі рекурентної нейронної мережі (Recurrent Neural Network, RNN) [29]. Генеруюча модель буде навчатися на наборі даних творів Шекспіра, дані будуть поділені на невеликі пакети тексту (кожен по 100 символів). Синтезована модель для генерації тексту буде здатна генерувати довгі послідовності тексту зі зв'язною структурою.

Отже, завдання синтезу моделі для генерування тексту буде полягати у тому, щоб враховуючи символ чи послідовність деяких символів, визначити, який наступний символ є найбільш вірогідним. Вхідними даними моделі буде послідовність символів, на основі якої модель буде прогнозувати вихідні дані – наступний символ кожному часовому кроці.

Імпортуємо TensorFlow та інші бібліотеки:

```
import tensorflow as tf

import numpy as np
import os
import time
```

На першому етапі завантажимо набір даних «Shakespeare» та виведемо деяку інформацію про нього:

```
path_to_file =
tf.keras.utils.get_file('shakespeare.txt',
'https://storage.googleapis.com/download.tensorflow.org
/data/shakespeare.txt')
```

```
text = open(path_to_file,
'rb').read().decode(encoding='utf-8')
# довжина тексту дорівнює кількості символів у ньому
```

```

print(f'Length of text: {len(text)} characters')

# виведення перших 250 символів тексту
print(text[:250])

# Унікальні символи у файлі
vocab = sorted(set(text))
print(f'{len(vocab)} unique characters: {vocab}')

```

**На другому етапі виконаємо попередню обробку навчального тексту.** Перед навчанням моделі необхідно перетворити рядки на числове подання. Шар `tf.keras.layers.StringLookup` може перетворювати кожен символ на числовий ідентифікатор. Для цього спочатку потрібно розділити текст на токени.

```

example_texts = ['abcdefg', 'xyz']
chars = tf.strings.unicode_split(example_texts, input_encoding='UTF-8')
print("chars =", chars)

```

```

#Створення шару tf.keras.layers.StringLookup, який
#перетворює токени на ідентифікатори символів:
ids_from_chars = tf.keras.layers.StringLookup(
    vocabulary=list(vocab), mask_token=None)

```

```

ids = ids_from_chars(chars)
print("ids=", ids)

```

Оскільки нам необхідно створити модель для генерації тексту, також важливо інвертувати це подання і відновити з нього рядки, що легко читаються. Для цього створюємо шар `chars_from_ids = tf.keras.layers.StringLookup`, який перетворює ідентифікатори символів на символи (відновлює символи з ідентифікаторів):

```

chars_from_ids = tf.keras.layers.StringLookup(
    vocabulary=ids_from_chars.get_vocabulary(),
    invert=True, mask_token=None)

```

```

chars = chars_from_ids(ids)
print("chars=", chars)

```

Також доцільно мати функцію об'єднання символів у строки:

```
def text_from_ids(ids):
    return tf.strings.reduce_join(chars_from_ids(ids),
axis=-1)
```

Після цього виконується **створення навчальних екземплярів та відповідних їм вихідних параметрів**. Для цього навчальний текст ділиться на приклади (екземпляри) послідовностей. Кожна вхідна послідовність міститиме `seq_length` символів із тексту. Для кожної вхідної послідовності відповідні виходи містять текст однакової довжини, крім зміщеного вправо одного символу. Отже, необхідно розбити текст на шматки довжиною `seq_length+1`. Наприклад, припустимо, що `seq_length` дорівнює 4, а наш текст – "Hello". Вхідна послідовність буде "Hell", а цільова послідовність "ello". Для цього спочатку використовуйте функцію `tf.data.Dataset.from_tensor_slices`, щоб перетворити текстовий вектор на потік індексів символів.

```
all_ids = ids_from_chars(tf.strings.unicode_split(text,
'UTF-8'))
print("all_ids=", all_ids)
```

```
ids_dataset =
tf.data.Dataset.from_tensor_slices(all_ids)
```

```
for ids in ids_dataset.take(10):
    print(chars_from_ids(ids).numpy().decode('utf-8'))
```

#Пакетний метод `batch` дозволяє конвертувати окремі символи `ids_dataset` в послідовності `sequences` потрібного розміру.

```
seq_length = 100
sequences = ids_dataset.batch(seq_length+1,
drop_remainder=True)
for seq in sequences.take(1):
    print(chars_from_ids(seq))
for seq in sequences.take(5):
    print(text_from_ids(seq).numpy())
```

Для навчання знадобиться набір даних у вигляді пар (`input`, `label`), де `input` та `label` є послідовностями. На кожному часовому кроці введенням є поточний символ, а міткою є наступний символ. Створимо функцію, яка приймає послідовність символів як вхідні дані, дуб-

лює та зсуває її, щоб вирівняти вхідні дані та мітку для кожного часового кроку:

```
def split_input_target(sequence):
    input_text = sequence[:-1]
    target_text = sequence[1:]
    return input_text, target_text
```

### **Формування масиву даних dataset:**

```
dataset = sequences.map(split_input_target)
for input_example, target_example in dataset.take(1):
    print("Input :",
          text_from_ids(input_example).numpy())
    print("Target:",
          text_from_ids(target_example).numpy())
```

До того, як вводити навчальні дані у модель, їх необхідно перетасувати та запакувати в пакети.

```
BATCH_SIZE = 64
BUFFER_SIZE = 10000
```

```
dataset = (
    dataset
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE, drop_remainder=True)
    .prefetch(tf.data.experimental.AUTOTUNE))

print("dataset=", dataset)
```

На третьому етапі здійснюється синтез моделі для генерації тексту. Модель складається з трьох шарів:

- `tf.keras.layers.Embedding` – вхідний шар. Навчальна таблиця, яка буде відображати кожен ідентифікатор символу у вектор з розмірами `embedding_dim` ;

- `tf.keras.layers.GRU` – тип RNN з кількістю нейронів, що дорівнює `rnn_units` (тут також можна використовувати шар LSTM);

- `tf.keras.layers.Dense` – вихідний шар із кількістю виходів, що дорівнює `vocab_size` . Він виводить один логіт (логарифмічна ймовірність) для кожного символу у словнику. Чим більша ця величина для конкретного символу, тим більшою є ймовірність того, що він буде виходом мережі.

```

# довжина словнику у шарі Embedding
vocab_size = len(vocab)

# The embedding dimension
embedding_dim = 256

# кількість RNN нейронів
rnn_units = 1024

class MyModel(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim,
rnn_units):
        super().__init__(self)
        self.embedding =
tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(rnn_units,
re-
turn_sequences=True,
return_state=True)
        self.dense = tf.keras.layers.Dense(vocab_size)

    def call(self, inputs, states=None, re-
turn_state=False, training=False):
        x = inputs
        x = self.embedding(x, training=training)
        if states is None:
            states = self.gru.get_initial_state(x)
        x, states = self.gru(x, initial_state=states,
training=training)
        x = self.dense(x, training=training)

        if return_state:
            return x, states
        else:
            return x

model = MyModel(
    vocab_size=vocab_size,
    embedding_dim=embedding_dim,
    rnn_units=rnn_units)

```

Для кожного символу модель шукає його подання (embedding), запускає GRU на один часовий крок із поданням як вхідними даними і застосовує dense-шар для генерації логітів, що передбачають логарифмічну ймовірність наступного символу (рис. 3.3).

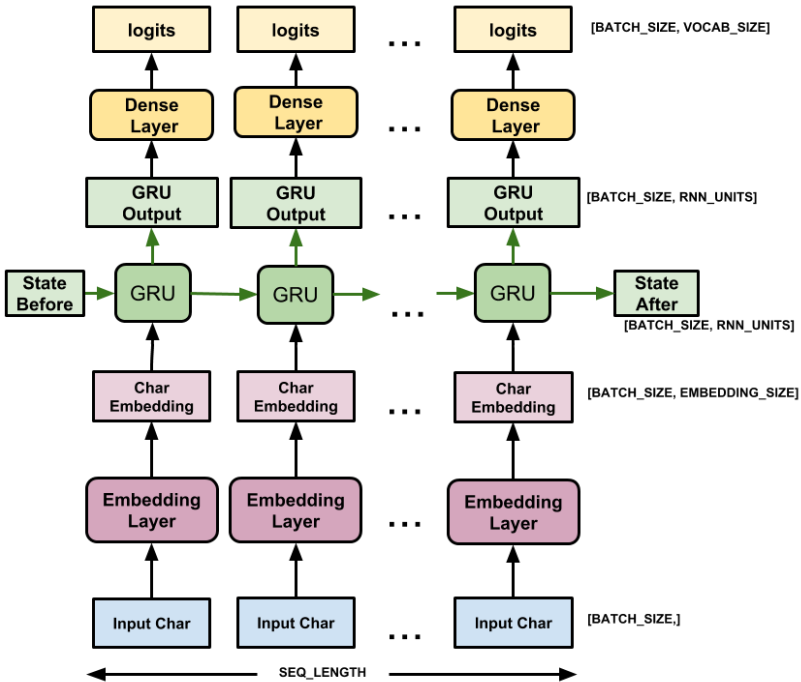


Рисунок 3.3 – Схематичне подання моделі для генерації тексту [29]

Виконаємо **перевірку ще ненавченої моделі** та виведемо узагальнену інформацію про неї:

```
for input_example_batch, target_example_batch in dataset.take(1):
    example_batch_predictions = model(input_example_batch)
    print(example_batch_predictions.shape, "#
(batch_size, sequence_length, vocab_size)")

model.summary()
```

```

sampled_indices =
tf.random.categorical(example_batch_predictions[0],
num_samples=1)
sampled_indices = tf.squeeze(sampled_indices, axis=-
1).numpy()
print("=====",sampled_indices)
print("Input:\n",
text_from_ids(input_example_batch[0]).numpy())
print()
print("Next Char Predictions:\n",
text_from_ids(sampled_indices).numpy())

```

**Встановлення параметрів для навчання моделі та її компіляція (ініціалізація):**

```

loss =
tf.losses.SparseCategoricalCrossentropy(from_logits=True)

example_batch_mean_loss = loss(target_example_batch, exam-
ple_batch_predictions)
print("Prediction shape: ", example_batch_predictions.shape,
" # (batch_size, sequence_length, vocab_size)")
print("Mean loss:          ", example_batch_mean_loss)

print(tf.exp(example_batch_mean_loss).numpy())

model.compile(optimizer='adam', loss=loss)

```

Відзначимо, що враховуючи складність синтезованої моделі для генерації тексту, процес її побудови може зайняти доволі багато часу (порядка декількох годин). Тому доцільно **встановити чекпоінти** (configure checkpoints), тобто конкретні точки у процесі навчання моделі, де стан моделі зберігається і до якого можна повернутися пізніше (щоб уникнути необхідності починати з нуля навчання моделі, якщо цей процес перерветься).

```

checkpoint_dir = './training_checkpoints'
# Name of the checkpoint files
checkpoint_prefix = os.path.join(checkpoint_dir,
"ckpt_{epoch}")

checkpoint_callback =
tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,

```



```
save_weights_only=True)
```

Після цього виконується навчання моделі (процес, що може зайняти декілька годин):

```
EPOCHS = 20
```

```
history = model.fit(dataset, epochs=EPOCHS,  
callbacks=[checkpoint_callback])
```

На четвертому етапі синтезована модель використовується для генерації тексту. Приклад процесу генерації тексту за допомогою синтезованої моделі наведено на рис. 3.4. Найпростіший спосіб згенерувати текст за допомогою синтезованої моделі – запустити її в циклі та відстежувати внутрішній стан моделі у міру її виконання.

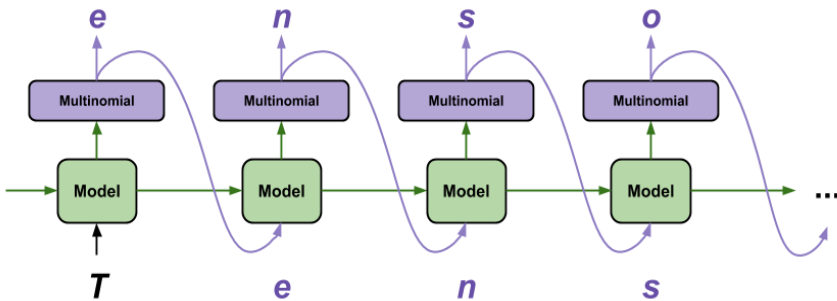


Рисунок 3.4 – Генерація тексту за допомогою синтезованої моделі [29]

Щоразу, коли викликається модель, їй передається деякий текст та внутрішній стан самої моделі. Модель повертає прогноз для наступного символу та її новий стан. Відповідно, для створення наступного символу необхідно передати поточний символ та поточний стан моделі, і продовжувати діяти так до тих пір, поки не буде згенеровано текст потрібної довжини.

Отже, для можливості генерації тексту на основі синтезованої моделі створимо **клас та функцію для генерації одного символу**:

```
class OneStep(tf.keras.Model):  
    def __init__(self, model, chars_from_ids, ids_from_chars,  
temperature=1.0):  
        super().__init__()
```

```

self.temperature = temperature
self.model = model
self.chars_from_ids = chars_from_ids
self.ids_from_chars = ids_from_chars

# Create a mask to prevent "[UNK]" from being generated.
skip_ids = self.ids_from_chars(['[UNK]'])[:, None]
sparse_mask = tf.SparseTensor(
    # Put a -inf at each bad index.
    values=[-float('inf')]*len(skip_ids),
    indices=skip_ids,
    # Match the shape to the vocabulary
    dense_shape=[len(ids_from_chars.get_vocabulary())])
self.prediction_mask = tf.sparse.to_dense(sparse_mask)

@tf.function
def generate_one_step(self, inputs, states=None):
    # Convert strings to token IDs.
    input_chars = tf.strings.unicode_split(inputs, 'UTF-8')
    input_ids = self.ids_from_chars(input_chars).to_tensor()

    # Run the model.
    # predicted_logits.shape is [batch, char,
next_char_logits]
    predicted_logits, states = self.model(inputs=input_ids,
states=states,
return_state=True)

    # Only use the last prediction.
    predicted_logits = predicted_logits[:, -1, :]
    predicted_logits = predicted_logits/self.temperature
    # Apply the prediction mask: prevent "[UNK]" from being
generated.
    predicted_logits = predicted_logits +
self.prediction_mask

    # Sample the output logits to generate token IDs.
    predicted_ids = tf.random.categorical(predicted_logits,
num_samples=1)
    predicted_ids = tf.squeeze(predicted_ids, axis=-1)

    # Convert from token ids to characters
    predicted_chars = self.chars_from_ids(predicted_ids)

    # Return the characters and model state.
    return predicted_chars, states

```

Створимо екземпляр `one_step_model` класу `OneStep`

```
one_step_model = OneStep(model, chars_from_ids,
ids_from_chars)
```

Запустимо однокрокову генерацію символів `one_step_model` у циклі, щоб створити текст (будемо створювати текст розміром 1000 символів).

```
start = time.time()
states = None
next_char = tf.constant(['ROMEO:'])
result = [next_char]

for n in range(1000):
    next_char, states =
one_step_model.generate_one_step(next_char, states=states)
    result.append(next_char)

result = tf.strings.join(result)
end = time.time()
print(result[0].numpy().decode('utf-8'), '\n\n' + '_'*80)
print('\nRun time:', end - start)
```

Як результат буде згенеровано певний текст на основі текстів, поданих у навчальній вибірці. Відзначимо, що при кожному запуску моделі текст може бути різним. Наприклад, результат може бути таким:

```
ROMEO:
'Tis most love, and you are well were born,
Lest in it to make his foe.
```

```
ROMEO:
To serve, him that whether he lengthens ut.
A hably fellow; I had quite my state
Unless dark to your request;
With father was prince'd 'proud; for what they are
knew within; that though you hill nor senjert; but a
divinely of the like, on who is death.
This interchanged can revenge his actor in the right,
And then the treasure of My cut her duty, and
thy father bears the clouds change ty: but they do our disgrace
Teaching their putting soul that kills she duty below his love
And so do I but threaten me out
His household turns virtues sea much before:
My follory ourselves heaven, and when I finds it out,
And mad that know them know that barber'd waves:
Ay, and all foul venom to the malige;
If not, you seek to bring them aumerle.
```

GREGORY:  
 The duke your lordship there:  
 My lord, you shall answer it to me:  
 Fear you we come, I warrant her that can I has  
 the very windows loving dry, or worthrel chassing  
 Then I'll

Дивлячись на згенерований текст, можна побачити, що модель знає, коли використовувати великі літери, робити абзаци та імітує шекспірівський словниковий запас. За малої кількості епох навчання він ще не навчився складати зв'язні речення.

Поліпшити роботу моделі можна, наприклад, шляхом збільшення кількості епох навчання EPOCHS.

**Однокрокову модель можна зберегти та відновити, що дозволяє використовувати її повторно.**

```
tf.saved_model.save(one_step_model, 'one_step')
one_step_reloaded = tf.saved_model.load('one_step')
```

**Використання завантаженої моделі можна виконати таким чином:**

```
states = None
next_char = tf.constant(['ROMEO:'])
result = [next_char]

for n in range(100):
    next_char, states =
one_step_reloaded.generate_one_step(next_char, states=states)
    result.append(next_char)

print(tf.strings.join(result)[0].numpy().decode("utf-8"))
```

Окрім наведеного прикладу генерації текстів за допомогою RNN моделей, рекомендовано ознайомитися з іншими прикладами генерації текстів за допомогою LSTM мереж та GPT моделі, поданими, відповідно, у джерелах [30] та [31].

### 3.4 Завдання на лабораторну роботу

3.4.1 Ознайомитися з основними теоретичними відомостями та рекомендованою літературою за темою роботи.

3.4.2 Обрати вхідні дані, подані вигляді у текстової інформації, для подальшої їх обробки та аналізу.

Як вхідні дані можна обрати, наприклад, такі:

– дані з відомих репозиторіїв та інших відкритих джерел:

<https://archive.ics.uci.edu/ml/datasets.php?format=&task=&att=&are a=&numAtt=10to100&numIns=&type=text&sort=nameUp&view=table>

<https://paperswithcode.com/task/text-classification>

<https://www.exxactcorp.com/blog/Deep-Learning/What-is-Text-Classification>

<https://paperswithcode.com/task/text-generation>

– дані з предметної області, що є добре відомою для студента;

– спеціально згенеровані вхідні дані;

– будь-які інші дані, які подані у вигляді текстової інформації.

3.4.3 Розробити програмне забезпечення для вирішення однієї з задач обробки природної мови (класифікація текстів, генерація текстів або інші).

3.4.3.1 Для створення програмного забезпечення обрати мову програмування та інші засоби розробки (програмні бібліотеки та ін.). Студент не обмежується у виборі мови програмування та інших засоби розробки програмного забезпечення.

3.4.3.2 Проаналізувати обрані вхідні дані та предметну область, визначити вимоги до програмного забезпечення (формати подання вхідних даних, необхідність попередньої обробки даних, необхідність поділу множини даних на навчальну та тестову вибірки даних, використовувані математичні моделі для обробки природної мови, необхідність подання результатів у вигляді графіків та ін.).

3.4.3.3 Здійснити проєктування архітектури програмного забезпечення для обробки текстової інформації.

3.4.3.4 Виконати конструювання програмного забезпечення.

3.4.3.5 Виконати тестування розробленого програмного забезпечення. Подати результати тестування розробленого програмного забезпечення для обробки природної мови у зручному для подальшого аналізу вигляді.

3.4.4. Зробити висновки за роботою.

3.4.5. Оформити звіт з роботи.

3.4.6. Відповісти на контрольні питання.

### 3.5 Зміст звіту

- назва і мета роботи;
- фрагмент вхідних даних та їх короткий опис;
- текст розробленого програмного забезпечення;
- результати аналізу вхідних даних за допомогою розробленого програмного забезпечення;
- висновки, що відображують результати виконання лабораторної роботи.

### 3.6 Контрольні запитання

- 3.6.1 Поясніть поняття «Обробка природної мови».
- 3.6.2 Які завдання відносяться до обробки природної мови?
- 3.6.3 В яких практичних застосуваннях можуть використовувати методи обробки природної мови?
- 3.6.4 Що таке класифікація текстів?
- 3.6.5 Які етапи виконуються при розробці програмного забезпечення для класифікації текстів?
- 3.6.6 Що таке токенізація?
- 3.6.7 Проаналізуйте етап синтезу моделі при побудові текстових класифікаторів.
- 3.6.8 Які типи моделей доцільно використовувати для побудови текстових класифікаторів?
- 3.6.9 У чому полягає етап тестування та перевірки працездатності моделі при побудові текстового класифікатора?
- 3.6.10 Які використовуються критерії оцінювання ефективності текстових класифікаторів?
- 3.6.11 У чому полягає суть генерації текстової інформації?
- 3.6.12 Які типи моделей доцільно використовувати для генерації тексту.
- 3.6.13 Проаналізуйте модель GPT (Generative Pre-trained Transformer) для генерації текстової інформації.
- 3.6.14 Для розв'язання яких практичних завдань застосовуються текстові генератори?
- 3.6.15 Поясніть архітектуру рекурентної нейронної мережі для класифікації текстової інформації.

## 4 ЛАБОРАТОРНА РОБОТА № 4 РОЗПІЗНАВАННЯ ЗВУКУ

### 4.1 Мета роботи

Навчитися реалізовувати програмні проекти у сфері інженерії програмного забезпечення, пов'язані з необхідністю обробки аудіоданих.

### 4.2 Короткі теоретичні відомості

Розпізнавання звуку людської мови (Speech recognition) або перетворення мовлення в текст (speech-to-text) – це здатність машини або програми ідентифікувати слова, сказані вголос, і перетворювати їх на читабельний текст [32], [33].

Для **розпізнавання мовлення** використовується широкий спектр досліджень у галузі інформатики, лінгвістики та комп'ютерної техніки. Багато сучасних пристроїв і програм, орієнтованих на текст, мають функції розпізнавання мовлення [32].

Розпізнавання мовлення та розпізнавання голосу – це дві різні технології [32]:

- розпізнавання мовлення (Speech recognition) використовується для ідентифікації слів усної мови;

- розпізнавання голосу (Voice recognition) – це біометрична технологія ідентифікації голосу людини. Фактично це розпізнавання того, хто говорить (speaker recognition). У цих системах програму необхідно навчити розпізнавати голос конкретних користувачів.

**Системи розпізнавання мовлення** використовують комп'ютерні алгоритми для обробки та інтерпретації вимовлених слів і перетворення їх у текст. Програма перетворює звук, записаний мікрофоном, на письмову мову, зрозумілу комп'ютерам і людям, виконуючи такі **етапи** [32]:

- аналіз аудіозапису;
- розбиття аудіозапису на частини;
- оцифрування аудіозапису у формат, який читається комп'ютером;

– використання алгоритму розпізнавання, щоб зіставити звук з найбільш підходящим представленням тексту.

Програмне забезпечення для розпізнавання мовлення має адаптуватися до надзвичайно мінливої та залежно від контексту природи людської мови. Алгоритми програмного забезпечення, які обробляють і організовують аудіо в текст, **навчаються на різних моделях мовлення**, стилях мовлення, мовах, діалектах, акцентах і фразях. Програмне забезпечення також відокремлює розмовний звук від фонового шуму, який часто супроводжує сигнал. Щоб відповідати цим вимогам, системи розпізнавання мовлення використовують **два типи моделей**:

– акустичні моделі – представляють зв'язок між мовними одиницями мови та звуковими сигналами;

– мовні моделі. Тут звуки зіставляються з послідовністю слів, щоб розрізнити слова, які схожі за звучанням.

Системи розпізнавання мовлення знайшли широке **застосування** у різних галузях [32]:

– мобільні пристрої. Смартфони використовують голосові команди для маршрутизації викликів, обробки мови в текст, голосового набору та голосового пошуку;

– освіта. Програмне забезпечення для розпізнавання мовлення використовується під час навчання мови. Програмне забезпечення чує мову користувача та пропонує допомогу з вимовою;

– обслуговування клієнтів. Автоматизовані голосові помічники слухають запити клієнтів і надають корисні ресурси;

– застосунки для охорони здоров'я;

– застосунки для формування фонограм різних засідань;

– програми для керування голосом.

**Особливості систем розпізнавання мовлення** [32], [33]:

– мовне зважування (Language weighting). Ця функція вказує алгоритму приділяти особливу увагу певним словам, наприклад тим, які часто вимовляються або є унікальними для розмови чи теми. Наприклад, програмне забезпечення можна навчити отримувати посилання на конкретні продукти;

– акустична підготовка (Acoustic training) – можливість системи до адаптації до навколишнього середовища (наприклад, навколишнього шуму в кол-центрі) та стилю мовця (наприклад, висоти голосу, гучності та темпу). Програмне забезпечення з акустичним навчанням мо-



же розрізнити стиль, темп і гучність розмови серед галасу багатьох людей, які говорять в офісі;

– маркування учасників розмови (Speaker labeling). Ця можливість дозволяє програмі позначати окремих учасників і визначати їхній конкретний внесок у розмову;

– фільтрація ненормативної лексики (Profanity filtering).

Як математичний базис для побудови розпізнавальних моделей у **системах розпізнавання мовлення використовуються** такі моделі [32], [33]:

– прихована модель Маркова (Hidden Markov model, HMM) використовуються в автономних системах, де стан можна спостерігати частково або коли вся інформація, необхідна для прийняття рішення, недоступна відразу для датчика (у випадку розпізнавання мови, мікрофона). Прикладом цього є акустичне моделювання, де програма повинна зіставляти мовні одиниці з аудіосигналами за допомогою статистичної ймовірності;

– моделі обробки природної мови. Хоча такі методи та моделі не обов'язково безпосередньо використовуються для розпізнавання мовлення, вони зосереджуються на взаємодії між людьми та машинами через мову та текст. Ці методи та моделі полегшують і прискорюють процес розпізнавання мови;

– N-грами. Цей підхід до мовних моделей створює розподіл ймовірностей для послідовності. Прикладом може бути алгоритм, який розглядає кілька останніх сказаних слів, оцінює історію зразка мовлення та використовує це для визначення ймовірності наступного слова чи фрази, які будуть сказані;

– нейронні мережі – обробляють навчальні дані, імітуючи взаємозв'язок людського мозку через шари нейронів, де кожен нейрон складається з входів, ваг, зміщення (або порогу) і виходу. Хоча нейронні мережі, як правило, точніші та можуть обробляти більше даних, це відбувається за рахунок більш повільного часу навчання у порівнянні з традиційними мовними моделями;

– діаризація мовця (Speaker Diarization). Алгоритми діаризації мовця ідентифікують і сегментують мову за особою мовця. Це допомагає програмам краще розрізняти людей у розмові та часто застосовується в кол-центрах для розрізнення клієнтів і торгових агентів.

### 4.3 Приклад розробки програмного забезпечення для обробки аудіо-даних

Як приклад, будемо використовувати програмні коди, подані у джерелі [34]. Програмне забезпечення буде розроблюватися з використанням бібліотеки tensorflow та мови Python. Розглянемо процес створення програмного забезпечення на основі згорткової нейронної мережі для розпізнавання простих звукових команд "down", "go", "left", "no", "right", "stop", "up", "yes" [34].

Імпортуємо модулі, необхідні для створення програмного забезпечення:

```
import os
import pathlib

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf

from tensorflow.keras import layers
from tensorflow.keras import models
from IPython import display
```

У випадку, якщо потрібні модулі не встановлені, їх необхідно встановити, ввівши відповідну команду у командному рядку. Наприклад, для встановлення модулю seaborn необхідно написати таку команду: `pip install seaborn`.

На першому етапі **завантажимо** та розпакуємо **набір даних**:

```
DATASET_PATH = 'data/mini_speech_commands'

data_dir = pathlib.Path(DATASET_PATH)
if not data_dir.exists():
    tf.keras.utils.get_file(
        'mini_speech_commands.zip',

origin="http://storage.googleapis.com/download.tensorflow.org
/data/mini_speech_commands.zip",
        extract=True,
        cache_dir='.', cache_subdir='data')
```

```
#Набори даних з аудіо кліпів зберігаються у 8ми папках, що
відповідають речовим командам: no, yes, down, go, left, up,
right, and stop
commands = np.array(tf.io.gfile.listdir(str(data_dir)))
commands = commands[(commands != 'README.md') & (commands !=
'.DS_Store')]
print('Commands:', commands)
```

На другому етапі створимо **навчальну та тестову вибірки даних:**

```
train_ds, val_ds =
tf.keras.utils.audio_dataset_from_directory(
    directory=data_dir,
    batch_size=64,
    validation_split=0.2,
    seed=0,
    output_sequence_length=16000,
    subset='both')

label_names = np.array(train_ds.class_names)
print()
print("label names:", label_names)
```

Оскільки завантажений набір даних містить лише одноканальне аудіо, то необхідно використити функцію `tf.squeeze`, щоб видалити додаткову вісь:

```
def squeeze(audio, labels):
    audio = tf.squeeze(audio, axis=-1)
    return audio, labels

train_ds = train_ds.map(squeeze, tf.data.AUTOTUNE)
val_ds = val_ds.map(squeeze, tf.data.AUTOTUNE)
```

Після цього виконаємо розбиття тестової вибірки та тестову та перевірімо:

```
test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)

for example_audio, example_labels in train_ds.take(1):
    print(example_audio.shape)
    print(example_labels.shape)
```

Побудуємо графіки декількох звукових сигналів (рис. 4.1).

```
rows = 3
```

```

cols = 3
n = rows * cols
fig, axes = plt.subplots(rows, cols, figsize=(16, 9))

for i in range(n):
    if i >= n:
        break
    r = i // cols
    c = i % cols
    ax = axes[r][c]
    ax.plot(example_audio[i].numpy())
    ax.set_yticks(np.arange(-1.2, 1.2, 0.2))
    label = label_names[example_labels[i]]
    ax.set_title(label)
    ax.set_ylim([-1.1, 1.1])

plt.show()

```

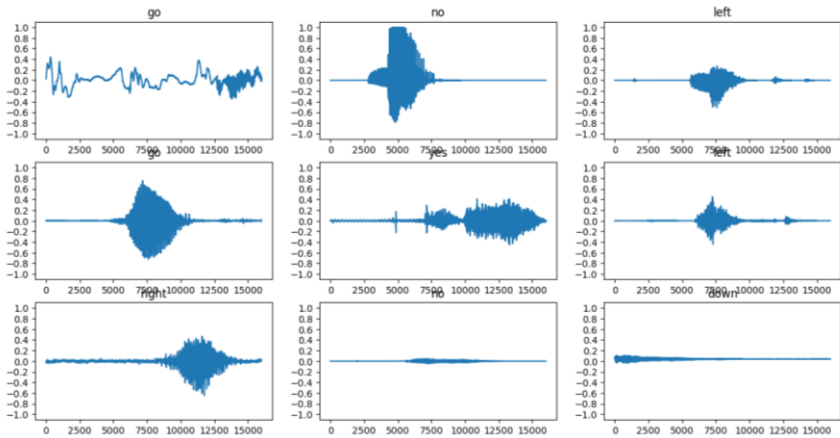


Рисунок 4.1 – Фрагмент вхідних даних (звукові сигнали)

На третьому етапі виконаємо **попередню обробку даних шляхом перетворення звукових сигналів у спектограми**. Сигнали в наборі даних представлені у часовій області. Виконаємо перетворення форми сигналів із сигналів у часовій області сигнали в частотно-часову область шляхом обчислення короткочасного перетворення Фур'є (short-time Fourier transform, STFT) для перетворення сигналів у вигляд спектограм, які показують зміни частоти в часі і можуть бути представлені у вигляді 2D-зображень. Після цього зображення спект-

рограми можна передавати у модель на основі нейронної мережі для її навчання.

Перетворення Фур'є (`tf.signal.fft`) перетворює сигнал на його складові частоти, але втрачає всю інформацію про час. Для порівняння, `STFT` (`tf.signal.stft`) розбиває сигнал на вікна часу і виконує перетворення Фур'є у кожному вікні, зберігаючи деяку інформацію про час і повертаючи двовірний тензор.

Функція для перетворення сигналів у спектрограми має такий вигляд:

```
def get_spectrogram(waveform):
    # отримаємо спектрограму за допомогою STFT
    spectrogram = tf.signal.stft(
        waveform, frame_length=255, frame_step=128)
    # отримаємо магнітуду спектрограми
    spectrogram = tf.abs(spectrogram)
    # Додаємо нову розмірність, щоб спектрограму можна було ви-
    користовувати як вхідні дані, схожі на зображення, із шарами
    згортки (які очікують форми («batch_size», «висота»,
    «ширина», «канали»).
    spectrogram = spectrogram[..., tf.newaxis]
    return spectrogram
```

Відзначимо, що сигнали мають бути однакової довжини, щоб під час їх перетворення на спектрограми результати мали однакові розміри. Це можна зробити простим доповненням нуля (за допомогою функції `tf.zeros`) тих аудіокліпів, які коротші за довжину найбільшого кліпу в заданому наборі даних. Оскільки в досліджуваному у цьому прикладі наборі `mini_speech_commands.zip` всі кліпи мають одну довжину (одну секунду), цю операцію виконувати недоцільно.

Під час виклику функції `tf.signal.stft` необхідно обрати параметри `frame_length` і `frame_step` так, щоб створене "зображення" спектрограми було майже квадратним.

Перетворення `STFT` створює масив комплексних чисел, що представляють величину (магнітуду) та фазу. У цьому прикладі будемо використовувати лише величину, яку можна отримати, застосувавши `tf.abs` до виводу `tf.signal.stft`.

Розробимо другу додаткову функцію для створення наборів даних у вигляді спектрограм із наборів аудіоданих:

```
def make_spec_ds(ds):
    return ds.map(
```

```
map_func=lambda audio,label: (get_spectrogram(audio),
label),
num_parallel_calls=tf.data.AUTOTUNE)
```

**Створимо навчальний, перевірочний та тестовий набори даних у вигляді спектограм**

```
train_spectrogram_ds = make_spec_ds(train_ds)
val_spectrogram_ds = make_spec_ds(val_ds)
test_spectrogram_ds = make_spec_ds(test_ds)
```

**Створимо допоміжну функцію, що виводить на екран спектограму у вигляді рисунку:**

```
def plot_spectrogram(spectrogram, ax):
    if len(spectrogram.shape) > 2:
        assert len(spectrogram.shape) == 3
        spectrogram = np.squeeze(spectrogram, axis=-1)
    log_spec = np.log(spectrogram.T + np.finfo(float).eps)
    height = log_spec.shape[0]
    width = log_spec.shape[1]
    X = np.linspace(0, np.size(spectrogram), num=width,
dtype=int)
    Y = range(height)
    ax.pcolormesh(X, Y, log_spec)
```

**Побудуємо графіки деяких спектограм (рис. 4.2), використовуючи попередньо створену функцію plot\_spectrogram:**

```
for example_spectrograms, example_spect_labels in
train_spectrogram_ds.take(1):
    break
rows = 3
cols = 3
n = rows*cols
fig, axes = plt.subplots(rows, cols, figsize=(16, 9))

for i in range(n):
    r = i // cols
    c = i % cols
    ax = axes[r][c]
    plot_spectrogram(example_spectrograms[i].numpy(), ax)

ax.set_title(label_names[example_spect_labels[i].numpy()])

plt.show()
```

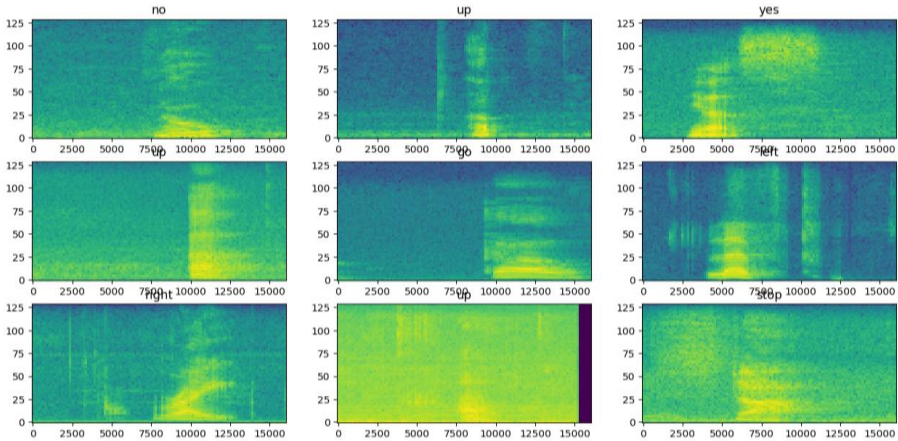


Рисунок 4.2 – Фрагмент вхідних даних у вигляді спектограм

На четвертому етапі виконується **створення та навчання моделі розпізнавання звукових сигналів** на основі нейронної мережі.

Застосуємо операції `Dataset.cache` і `Dataset.prefetch` до наявних наборів даних для зменшення затримки зчитування даних під час навчання моделі:

```
train_spectrogram_ds = train_spectrogram_ds.  
cache().shuffle(10000).prefetch(tf.data.AUTOTUNE)  
val_spectrogram_ds =  
val_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)  
test_spectrogram_ds =  
test_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
```

Для синтезу розпізнавальної моделі будемо використовувати згортову нейронну мережу (CNN), оскільки було попередньо виконано перетворення аудіофайлів на зображення спектрограми. Розпізнавальна моделі буде містити такі додаткові шари попередньої обробки:

- `tf.keras.layers.Resizing` – для зменшення вхідних даних, щоб модель могла навчатися швидше.

- `tf.keras.layers.Normalization` – для нормалізації кожного пікселя на зображенні на основі його середнього значення та стандартного відхилення. Для рівня `Normalization` метод `adapt` спочатку необхідно викликати для навчальних даних, щоб обчислити сукупну статистику (середнє значення та стандартне відхилення).

```

input_shape = example_spectrograms.shape[1:]
print('Input shape:', input_shape)
num_labels = len(label_names)

norm_layer = layers.Normalization()
norm_layer.adapt(data=train_spectrogram_ds.map(map_func
= lambda spec, label: spec))

# Створення екземпляру моделі класу models.Sequential,
що являє собою розпізнавальну модель для обробки
звукових сигналів
model = models.Sequential([
    layers.Input(shape=input_shape),
    layers.Resizing(32, 32), # Downsample the input
    norm_layer, # Normalize
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_labels),
])

model.summary()

```

**Налаштування моделі (оптимізатор Adam, крос-ентропійна функція втрат):**

```

model.compile(
    optimizer=tf.keras.optimizers.Adam(),

    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy'],
)

```

**Навчання моделі (використовуємо 10 епох навчання):**

```

EPOCHS = 10
history = model.fit(
    train_spectrogram_ds,

```



```
validation_data=val_spectrogram_ds,
epochs=EPOCHS,

callbacks=tf.keras.callbacks.EarlyStopping(verbose=1,
patience=2),
)
```

Побудуємо криві втрат під час навчання та перевірки, щоб перевірити, як модель покращувалася під час навчання (рис. 4.3).

```
metrics = history.history
plt.figure(figsize=(16, 6))
plt.subplot(1,2,1)
plt.plot(history.epoch, metrics['loss'], met-
rics['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.ylim([0, max(plt.ylim())])
plt.xlabel('Epoch')
plt.ylabel('Loss [CrossEntropy]')
plt.subplot(1,2,2)
plt.plot(history.epoch,
100*np.array(metrics['accuracy']),
100*np.array(metrics['val_accuracy']))
plt.legend(['accuracy', 'val_accuracy'])
plt.ylim([0, 100])
plt.xlabel('Epoch')
plt.ylabel('Accuracy [%]')
plt.show()
```

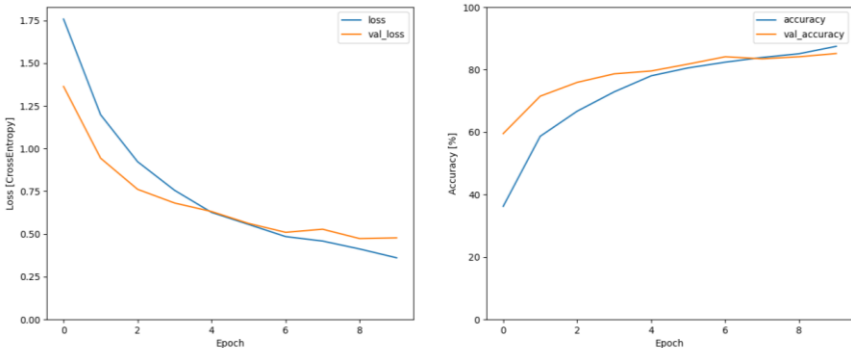


Рисунок 4.3 – Графіки залежності точності та втрат від номеру ітерації

Після побудови моделі можна перейти до п'ятого етапу, пов'язаного з **використанням синтезованої розпізнавальної моделі**.

Побудуємо матрицю невідповідностей (confusion matrix, рис. 4.4) на даних тестової вибірки:

```
model.evaluate(test_spectrogram_ds, return_dict=True)

# confusion matrix
y_pred = model.predict(test_spectrogram_ds)
y_pred = tf.argmax(y_pred, axis=1)
y_true = tf.concat(list(test_spectrogram_ds.map(lambda
s,lab: lab))), axis=0)

confusion_mtx = tf.math.confusion_matrix(y_true,
y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx,
            xticklabels=label_names,
            yticklabels=label_names,
            annot=True, fmt='g')
plt.xlabel('Prediction')
plt.ylabel('Label')
plt.show()
```

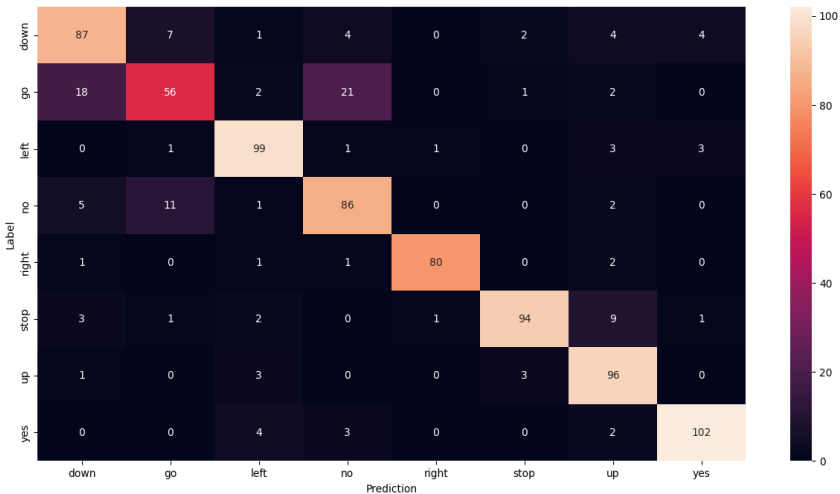


Рисунок 4.4 – Матриця невідповідностей на даних тестової вибірки

Як видно, переважну кількість аудіо-сигналів класифіковано вірно (значення головної діагоналі матриці невідповідностей значно більші за значення елементів матриці, що знаходяться поза головною діагоналлю).

Наведемо приклад прийняття рішення стосовно речової команди, поданої в аудіо-файлі за допомогою синтезованої моделі (визначення значення класу, до якого відноситься новий заданий аудіо-файл 01bb6a2a\_nohash\_0.wav):

```
x = data_dir/'no/01bb6a2a_nohash_0.wav'
x = tf.io.read_file(str(x))
x, sample_rate = tf.audio.decode_wav(x, desired_channels=1, desired_samples=16000,)
x = tf.squeeze(x, axis=-1)
waveform = x
x = get_spectrogram(x)
x = x[tf.newaxis,...]

prediction = model(x)
x_labels = ['no', 'yes', 'down', 'go', 'left', 'up', 'right', 'stop']
plt.bar(x_labels, tf.nn.softmax(prediction[0]))
plt.title('Go')
plt.show()
```

Графік, на якому відображено ймовірності прийняття рішень до кожного з наявних класів ("no", "yes", "down", "go", "left", "up", "right", "stop") наведено на рис. 4.5.

Як видно з рис.4.5, речова команда, подана в аудіофайлі 01bb6a2a\_nohash\_0.wav, розпізнається як команда «go» (за максимальним значенням ймовірності).

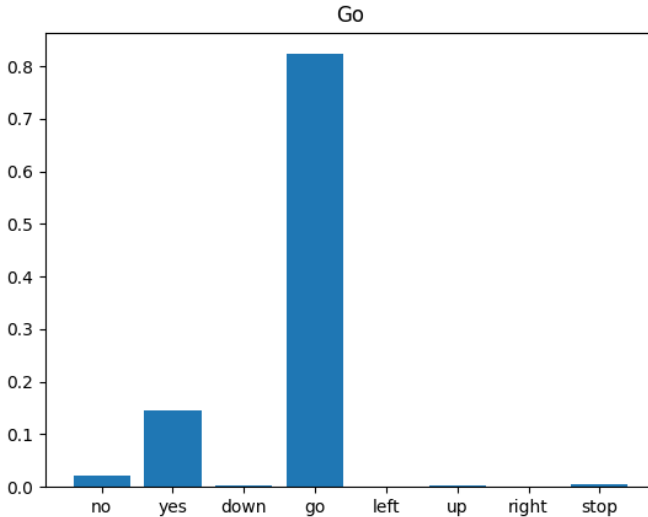


Рисунок 4.5 – Розподіл ймовірностей віднесення речової команди, поданої в аудіо-файлі, до кожного з наявних класів

#### 4.4 Завдання на лабораторну роботу

4.4.1 Ознайомитися з основними теоретичними відомостями та рекомендованою літературою за темою роботи.

4.4.2 Обрати вхідні дані, подані вигляді у аудіо-даних, для подальшої їх обробки та аналізу.

Як вхідні дані можна обрати, наприклад, такі:

– дані з відомих репозиторіїв та інших відкритих джерел:

<https://archive.ics.uci.edu/ml/datasets.php>

<https://paperswithcode.com/task/speech-recognition>

– дані з предметної області, що є добре відомою для студента;

– спеціально згенеровані вхідні дані;

– будь-які інші дані, які подані у вигляді аудіо-даних.

4.4.3 Розробити програмне забезпечення для вирішення однієї з задач обробки аудіо-даних (розпізнавання звуків та ін.).

4.4.3.1 Для створення програмного забезпечення обрати мову програмування та інші засоби розробки (програмні бібліотеки та ін.). Студент не обмежується у виборі мови програмування та інших засоби розробки програмного забезпечення.

4.4.3.2 Проаналізувати обрані вхідні дані та предметну область, визначити вимоги до програмного забезпечення (формати подання вхідних даних, необхідність попередньої обробки даних, необхідність поділу множини даних на навчальну та тестову вибірки даних, використовувані математичні моделі для обробки аудіо-даних, необхідність подання результатів у вигляді графіків та ін.).

4.4.3.3 Здійснити проєктування архітектури програмного забезпечення для обробки аудіо-даних.

4.4.3.4 Виконати конструювання програмного забезпечення.

4.4.3.5 Виконати тестування розробленого програмного забезпечення. Подати результати тестування розробленого програмного забезпечення для обробки аудіо-даних у зручному для подальшого аналізу вигляді.

4.4.4. Зробити висновки за роботою.

4.4.5. Оформити звіт з роботи.

4.4.6. Відповісти на контрольні питання.

## 4.5 Зміст звіту

- назва і мета роботи;
- фрагмент вхідних даних та їх короткий опис;
- текст розробленого програмного забезпечення;
- результати аналізу вхідних даних за допомогою розробленого програмного забезпечення;
- висновки, що відображують результати виконання лабораторної роботи.

## 4.6 Контрольні запитання

4.6.1 Проаналізуйте поняття розпізнавання мовлення.

4.6.2 У чому полягає різниця між розпізнаванням мовлення та розпізнаванням голосу?

4.6.3 Які етапи виконують при автоматичному розпізнаванні мовлення?

4.6.4 Які типи моделей використовуються у системах розпізнавання мовлення?

4.6.5 Проаналізуйте галузі застосування систем розпізнавання мовлення.

4.6.6 Яким чином системи розпізнавання мовлення можуть використовуватися у мобільних пристроях?

4.6.7 Якими особливостями характеризуються системи розпізнавання мовлення?

4.6.8 Що таке мовне зважування?

4.6.9 У чому полягає сутність акустичної підготовки?

4.6.10 Для чого використовується маркування учасників розмови у системах розпізнавання мовлення?

4.6.11 Які види моделей використовуються як математичний базис для побудови розпізнавальних моделей у системах розпізнавання мовлення?

4.6.12 З якою метою перетворюють звукові сигнали на спектограми при побудові розпізнавальних моделей у системах розпізнавання мовлення?

4.6.13 Поясніть призначення короткочасного перетворення Фур'є (short-time Fourier transform, STFT).

4.6.14 Яким чином змінюється значення точності розпізнавальної моделі в залежності від кількості ітерацій (epoch) навчання?

4.6.15 Як по матриці невідповідностей на даних тестової вибірки можна зрозуміти, на скільки якісною є синтезована розпізнавальна модель?

## ЛІТЕРАТУРА

1. What Is Time-Series Forecasting? [Electronic resource]. – Access mode: <https://www.timescale.com/blog/what-is-time-series-forecasting/>.
2. How To Isolate Trend, Seasonality And Noise From A Time Series [Electronic resource]. – Access mode: <https://timeseriesreasoning.com/contents/time-series-decomposition/>.
3. Exponential Smoothing for Time Series Forecasting [Electronic resource]. – Access mode: <https://orangematter.solarwinds.com/2017/06/22/exponential-smoothing-for-time-series-forecasting/>.
4. How to Develop Multilayer Perceptron Models for Time Series Forecasting [Electronic resource]. – Access mode: <https://machinelearningmastery.com/how-to-develop-multilayer-perceptron-models-for-time-series-forecasting/>.
5. How to Develop Convolutional Neural Network Models for Time Series Forecasting [Electronic resource]. – Access mode: <https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/>.
6. Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras [Electronic resource]. – Access mode: <https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/>.
7. Time series forecasting [Electronic resource]. – Access mode: [https://www.tensorflow.org/tutorials/structured\\_data/time\\_series](https://www.tensorflow.org/tutorials/structured_data/time_series).
8. Weather Time Series Dataset [Electronic resource]. – Access mode: <https://www.bgc-jena.mpg.de/wetter/>.
9. Object Detection with 10 lines of code [Electronic resource]. – Access mode: <https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606>.
10. What Is Image Recognition [Electronic resource]. – Access mode: <https://medium.com/dataman-in-ai/module-6-image-recognition-for-insurance-claim-handling-part-i-a338d16c9de0>.
11. Keras Applications [Electronic resource]. – Access mode: <https://keras.io/api/applications/>.

12. ImageNet [Electronic resource]. – Access mode: <https://image-net.org/index.php>.
13. Convolutional Neural Network [Electronic resource]. – Access mode: <https://www.tensorflow.org/tutorials/images/cnn>.
14. Image classification [Electronic resource]. – Access mode: <https://www.tensorflow.org/tutorials/images/classification>.
15. TensorFlow Hub Object Detection Colab [Electronic resource]. – Access mode: [https://www.tensorflow.org/hub/tutorials/tf2\\_object\\_detection](https://www.tensorflow.org/hub/tutorials/tf2_object_detection).
16. Detect an object with OpenCV-Python [Electronic resource]. – Access mode: <https://www.geeksforgeeks.org/detect-an-object-with-opencv-python/>.
17. Object Detection with Python, Deep Learning, and OpenCV [Electronic resource]. – Access mode: <https://dontrepeatyoursself.org/post/object-detection-with-python-deep-learning-and-opencv/>
18. Object Detection : Face Detection using Haar Cascade Classifiers [Electronic resource]. – Access mode: [https://www.bogotobogo.com/python/OpenCV\\_Python/python\\_opencv3\\_Image\\_Object\\_Detection\\_Face\\_Detection\\_Haar\\_Cascade\\_Classifiers.php](https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Object_Detection_Face_Detection_Haar_Cascade_Classifiers.php).
19. RetinaNet model [Electronic resource]. – Access mode: [https://github.com/OlafenwaMoses/ImageAI/releases/download/essentials-v5/resnet50\\_coco\\_best\\_v2.1.0.h5/](https://github.com/OlafenwaMoses/ImageAI/releases/download/essentials-v5/resnet50_coco_best_v2.1.0.h5/).
20. TensorFlow Datasets: a collection of ready-to-use datasets [Electronic resource]. – Access mode: <https://www.tensorflow.org/datasets>.
21. What Is Natural Language Processing (NLP) & How Does It Work? [Electronic resource]. – Access mode: <https://levity.ai/blog/how-natural-language-processing-works>.
22. Text Classification: What it is And Why it Matters [Electronic resource]. – Access mode: <https://monkeylearn.com/text-classification/>
23. What is Text Classification [Electronic resource]. – Access mode: <https://www.exxactcorp.com/blog/Deep-Learning/What-is-Text-Classification>.
24. Text Classification [Electronic resource]. – Access mode: <https://developers.google.com/machine-learning/guides/text-classification?hl=en>.



25. AI Text Generation: Top 12 Use Cases & 2 Case Studies [Electronic resource]. – Access mode: <https://research.aimultiple.com/ai-text-generation/>.
26. Iqbal T. The survey: Text generation models in deep learning [Text] / T. Iqbal, S. Qureshi // Journal of King Saud University – Computer and Information Sciences. – 2022. – Vol. 34, Issue 6, Part A. – P. 2515-2528.
27. Text classification with an RNN [Electronic resource]. – Access mode: [https://www.tensorflow.org/text/tutorials/text\\_classification\\_rnn](https://www.tensorflow.org/text/tutorials/text_classification_rnn).
28. Basic text classification [Electronic resource]. – Access mode: [https://www.tensorflow.org/tutorials/keras/text\\_classification](https://www.tensorflow.org/tutorials/keras/text_classification).
29. Text generation with an RNN [Electronic resource]. – Access mode: [https://www.tensorflow.org/text/tutorials/text\\_generation](https://www.tensorflow.org/text/tutorials/text_generation).
30. Beginners Guide to Text Generation using LSTMs [Electronic resource]. – Access mode: <https://www.kaggle.com/code/shivamb/beginners-guide-to-text-generation-using-lstms>.
31. GPT text generation with KerasNLP [Electronic resource]. – Access mode: [https://keras.io/examples/generative/text\\_generation\\_gpt/](https://keras.io/examples/generative/text_generation_gpt/).
32. Speech recognition [Electronic resource]. – Access mode: <https://www.techtarget.com/searchcustomerexperience/definition/speech-recognition/>.
33. What is speech recognition? [Electronic resource]. – Access mode: <https://www.ibm.com/topics/speech-recognition>.
34. Simple audio recognition: Recognizing keywords [El. resource]. – Access mode: [https://www.tensorflow.org/tutorials/audio/simple\\_audio](https://www.tensorflow.org/tutorials/audio/simple_audio).