Міністерство освіти і науки України
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
з дисципліни
«Інженерія прикладних інтелектуальних застосунків» 2023

ліни “Інженерія прикладних інтелектуальних застосунків” / А.О. Олійник, С. Д. Леощенко, Є.М. Федорченко. – Запоріжжя: НУ «Запорі зька 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 № 1
від “17” серпня 2023 р.

ЗМІСТ

ВСТУП

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

Відповідно до графіка студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою. У цих методичних вказівках використовуються матеріа ли, 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 ос новні, базові теоретичні відомості, необхідні для виконання лаборато рних 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 лі тературі. 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 результа ти 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 по винний виявити знання про зміст роботи та методи виконання кожно го етапу роботи, а також вміти продемонструвати результати роботи на конкретних прикладах. Студент повинний вміти правильно аналі зувати отримані результати. Для самоперевірки при підготовці до ви конання і захисту роботи студент повинен відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

1 ЛАБОРАТОРНА РОБОТА № 1 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ОБРОБКИ ЧАСОВИХ РЯДІВ

1.1 Мета роботи

Навчитися реалізовувати програмні проєкти 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 послідов ностей (time series)** даних, які являють собою ряд точок даних, проін дексованих 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 океань

ких 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 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 да

них. 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 да них.

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 мі стять постійні відхилення. Наприклад, якби інтернет-магазин проана лізував 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 пору шеннями) завжди можуть статися, і ми повинні враховувати це під час створення моделі прогнозування. Вони представляють шум в історич них даних, і вони також непередбачувані.

У наш час використовуються різні **групи методів прогнозу вання часових рядів:**

– декомпозиція часових рядів;
– регресійні моделі часових рядів;
– експоненціальне згладжування;

– моделі ARIMA;

– нейронні мережі.

Часто використовуються різні методи, аналізуються їх результа ти, 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 за мість моделювання за допомогою часових залежностей і автокореля ції. Цей підхід можна використовувати або як окремий метод для про гнозування 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 по тім додати ці прогнози разом, щоб отримати загальний прогноз.

Декомпозиція часових рядів відноситься до техніки, яка розкла дає 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.

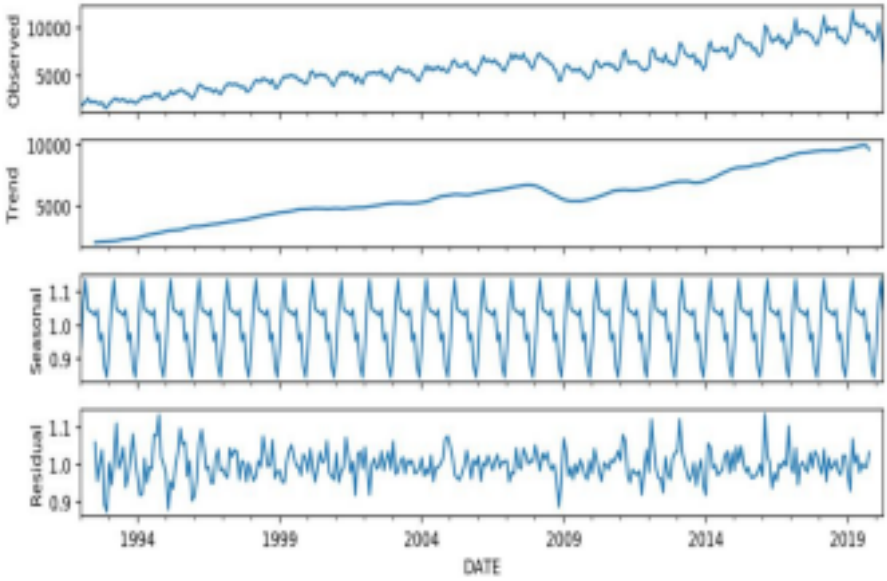


Рисунок 1.1 – Приклад розкладання наявних даних (observed) на тренд (trend), сезонність (seasonal) та залишок (residual) [2]

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 вико ристанням додавання та множення), призводять до початкового часо вого ряду. Кожен із компонентів має певну характеристику або тип поведінки, і зазвичай вони включають:

– T_t: компонент тренду в момент часу t, який описує довгостро кову 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 повторю вані, але неперіодичні коливання. Тривалість цих коливань залежить від характеру часового ряду;

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

– $_t I$: нерегулярний компонент (або «шум») у момент часу t – представляє випадкові та нерегулярні впливи. Його також можна ввa жати 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 функ цією суми своїх компонентів, тому модель можна представити насту пним рівнянням:

$$y = T + C + S + I,$$
$$t\ t\ t\ t\ t$$

де $_t y$ – дані часових рядів, T_t– компонент тренду, C_t– циклічний компонент, $_t S$ – сезонний компонент, $_t I$ – залишок.

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

$$y = T \cdot C \cdot S \cdot I.$$
$$t\ t\ t\ t\ t$$

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 ви хідною змінною. Вхідні змінні іноді називають регресорами, незалеж ними 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 β_0 та β_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 ε_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 зважує історичні дані однаково, щоб зробити прогнози щодо майбутнього, експоненціальне згладжування використовує експоненціальні функції для обчислення зменшення ваг з часом.

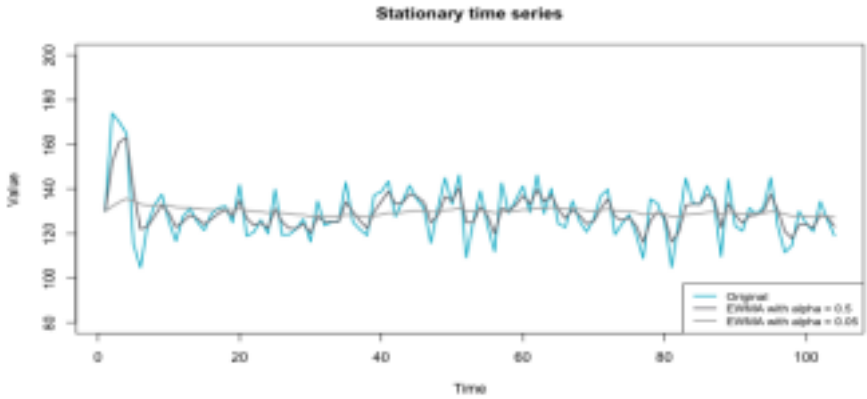


Рисунок 1.2 – Приклад експоненціального згладжування [3]

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

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

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

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

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

– робота з повними даними. Відсутні або пошкоджені дані за звичай не підтримуються;

– фокус на лінійних залежностях: припущення лінійної залеж ності виключає більш складні типи залежностей;

– робота з одновимірними даними: багато реальних проблем мають кілька вхідних змінних, що ускладнює застосування таких ме тодів;

– 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 вхідними 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 обмежен ням. Вони здатні апроксимувати будь-яку нелінійну функцію без по переднього знання про властивості рядів даних.

Нейронні мережі, такі як багатошарові персептрони, пропону ють численні переваги, зокрема:

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

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

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

– багатокрокові прогнози (Multi-step forecasts): кількість вихід них значень також є змінною.

Для прогнозування числових рядів доцільно використовувати такі типи нейронних мереж:

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

Багатошарові персептрони (Multilayer Perceptrons, MLP) можна використовувати для моделювання проблем прогнозування одно факторних та багатофакторних 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 змоделювати 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 деякі (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 однофактор них та багатофакторних даних, а також для однокрокового та для ба гатокрокового прогнозування наведено у джерелі [4].

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) – це тип нейронних мереж, розроблених для ефективної оброб ки 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 необробле ними даними, такими як необроблені значення пікселів, замість спе цифічних для домену або ручних функцій, отриманих із необроблених даних. Потім модель дізнається, як автоматично витягувати функції з необроблених даних, які безпосередньо корисні для вирішення про блеми. При використанні CNN характеристики зображень виділяють ся незалежно від того, як вони відбуваються в даних, забезпечуючи так звану інваріантність перетворення.

Здатність CNN навчатися та автоматично витягувати характери стики 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 багатошарових персептронів для про гнозування 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 най більш релевантним для проблеми прогнозування.

Створення згорткової нейронної мережі з використанням бібліо теки tensorflow та мови Python можна виконати таким чином: model = Sequential()

```
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', in
put_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 одно факторних та багатофакторних 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 спосте реженнями під час вивчення функції відображення входів і виходів, чого не пропонують MLP або CNN. Це тип нейронної мережі, яка до дає власну підтримку вхідних даних, що складаються з послідовнос тей 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 скла дних проблемах обробки природної мови, таких як нейронний ма шинний переклад, де модель повинна вивчати складні взаємозв'язки між словами як у межах певної мови, так і між мовами під час перек ладу 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, ре курентні нейронні мережі також можуть автоматично вивчати часову залежність з даних. Тому найбільш відповідний контекст вхідних спо стережень для очікуваного результату вивчається та може змінювати ся 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 про

гнозування. Модель вивчає відображення від входів до виходів і діз нається, який контекст із вхідної послідовності корисний для відобра ження, 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 ви гляді 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, тем пература повітря, атмосферний тиск та вологість. Вони збиралися ко жні 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 ви користати 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 необролених 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 граду сах. 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 віт ру 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 ви гляд для можливості використання цих параметрів як вхідних аргуме нтів 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 фо рмату на інший:

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

Ще раз відзначимо, що для попередньої обробки даних застосо вуються різні методи та підходи в залежності від особливостей конк ретного набору вхідних даних.

Після попередньої обробки даних виконаємо їх перетворення на три набори: **навчальні, перевірочні та тестові дані**. Для цього буде мо використовувати поділ (70%, 20%, 10%) для навчальних, перевіро чних 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 enumer
ate(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 зна чення вхідних параметрів. Нормалізація є поширеним способом масш табування, який полягає у відніманні середнього значення і поділу на стандартне відхилення кожної ознаки.

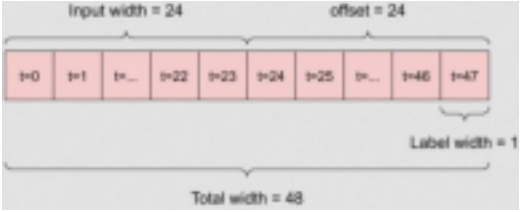
Середнє значення та стандартне відхилення слід обчислювати тільки з використанням навчальних даних, щоб моделі не мали досту пу до значень у перевірочних та тестових наборах.

```
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 функ ції для створення **вікон даних** на основі наявних часових послідовно стей. 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 використо вуються як вхідні дані, які як вихідні або те й інше. Розробимо про грамні функції для створення вікон даних (рис. 1.3) для можливості подальшого синтезу прогностичних моделей для прогнозування з од ним виходом або кількома виходами, а також з одним або декількома часовими кроками.



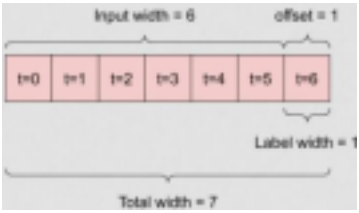


Рисунок 1.3 – Приклад вікон даних

21

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, врахо вуючи шість годин історичних даних. Фактично кожне таке вікно буде являти собою один з багатьох екземплярів даних для синтезу прогнос тичної 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 навчаль них, 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
                                    enumer
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 ме тод 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 ви конаємо **розробку функцій для прогнозування часових рядів на основі різних моделей**.

Найпростіша модель, яку можна побудувати для такого типу да них, – 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, прогно

зуючи «без змін».

Для сторення 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(),
                           met
rics=[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 sin gle_step_window. Створимо більшу кількість вікон за допомогою кла су WindowGenerator та згенеруємо вікна на 24 години послідовних

вхідних даних та міток за раз (рис.1.4).

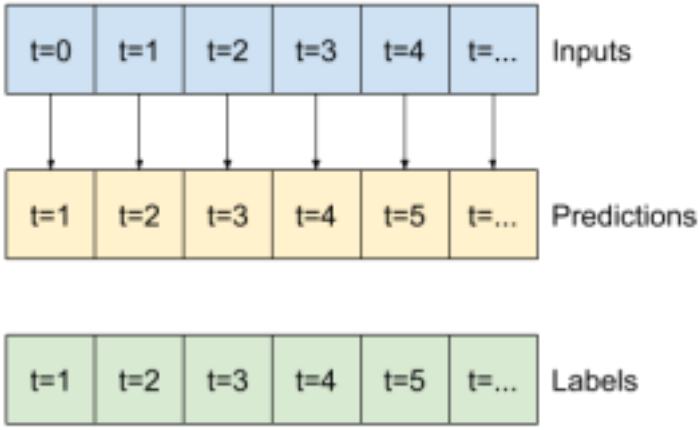


Рисунок 1.4 – Приклад вікон даних wide_window

Нова змінна wide_window не змінює спосіб роботи моделі. Мо дель, 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:', base
line(wide_window.example[0]).shape)
```

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

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

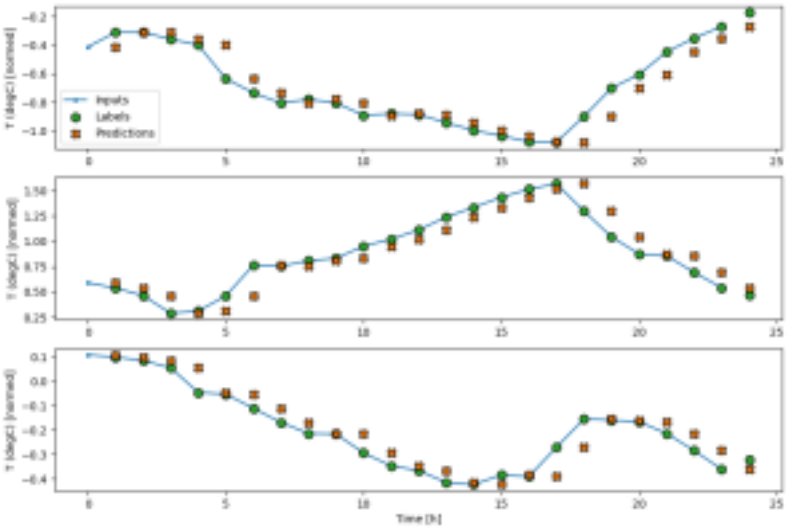



Рисунок 1.5 – Результати прогнозування за допомогою моделі Baseline

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

– синя лінія Inputs показує вхідну температуру на кожному ча совому інтервалі;

– зелені точки Labels показують цільове значення прогнозу (ре альне значення вихідного параметру). Діапазон міток зміщений на 1 крок щодо входів;

– помаранчеві хрести Predictions – це прогнози моделі для кож ного 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 ін ших моделей.

Виконаємо навчання лінійної моделі, а також оцінимо її ефекти вність:

```
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 мо дель робить набір незалежних прогнозів на послідовних часових кро ках. 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)
```

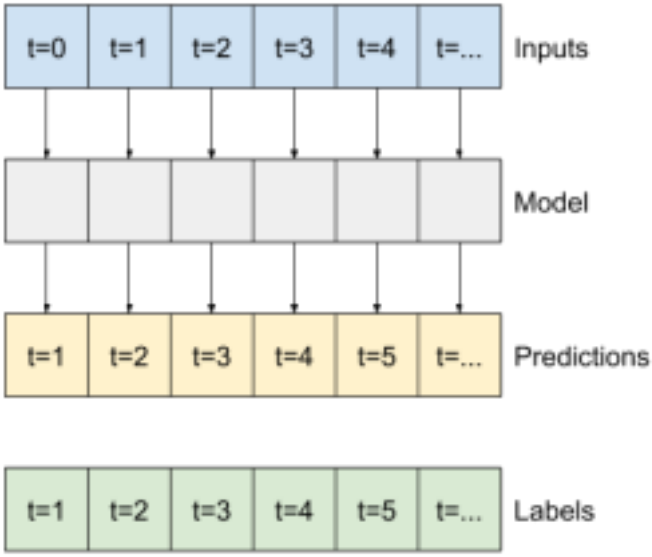


Рисунок 1.6 – Функціонування лінійної моделі

На рис.1.7 наведено результати прогнозування для широкого ві кна даних wide_window за допомогою лінійної моделі. Як видно, в

багатьох випадках прогноз явно кращий, ніж просто повернення вхід ної температури (як у моделі baseline), але в деяких випадках він гір ший.

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

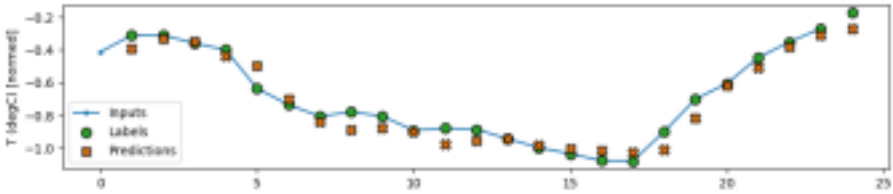


Рисунок 1.7 – Результати прогнозування для широкого вікна даних wide_window за допомогою лінійної моделі

Модель типу ***багатошарового персептрону*** може бути створе на 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 відпові дних вікон даних.

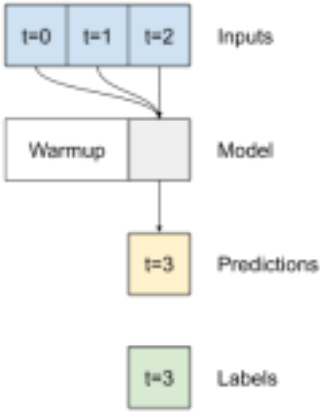


Рисунок 1.8 – Прогнозування на декілька кроків вперед

Створимо екземпляр класу WindowGenerator, який створювати ме 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, перепи сана за допомогою згортки.

Зверніть увагу на зміни:

– tf.keras.layers.Flatten та перший tf.keras.layers.Dense заміню ються 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 результа тів 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 поля гає в тому, що conv_model можна запускати на входах будь-якої дов жини. Згортковий шар застосовується до ковзного вікна вхідних даних (рис. 1.9).

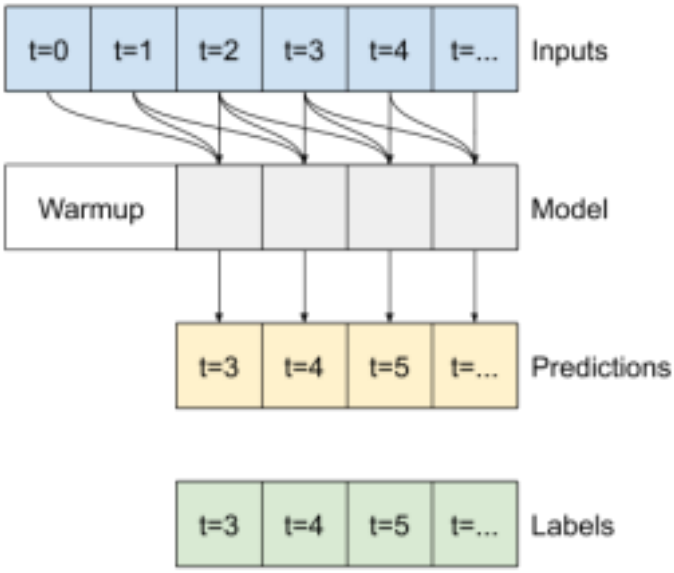


Рисунок 1.9 – Прогнозування на декілька кроків вперед за допомогою згорткової нейронної мережі

Створимо об’єкт класу WindowGenerator для генерації широких вікон з кількома додатковими часовими кроками введення, щоб дов жина 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)
```

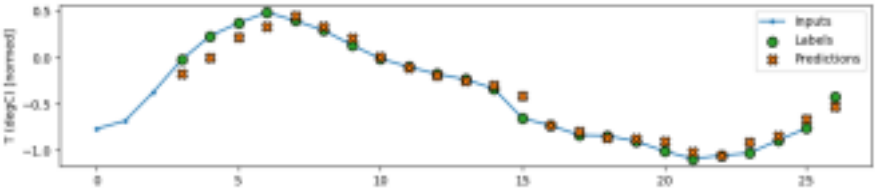


Рисунок 1.10 – Результати прогнозування для широкого вікна даних wide_window за допомогою згорткової мережі

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).

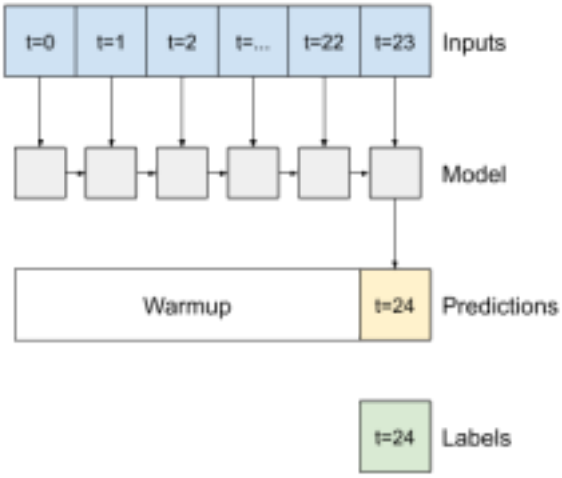


Рисунок 1.11 – Прогнозування за допомогою LSTM при return_sequences=False

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.

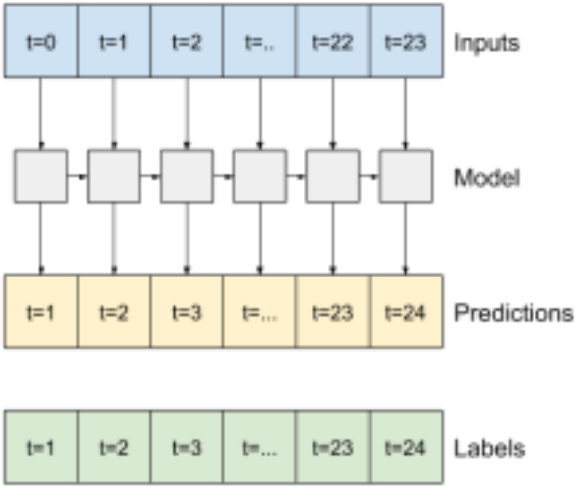


Рисунок 1.12 – Прогнозування за допомогою LSTM при return_sequences= True

За допомогою парамерту 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 (точ ність 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()
```

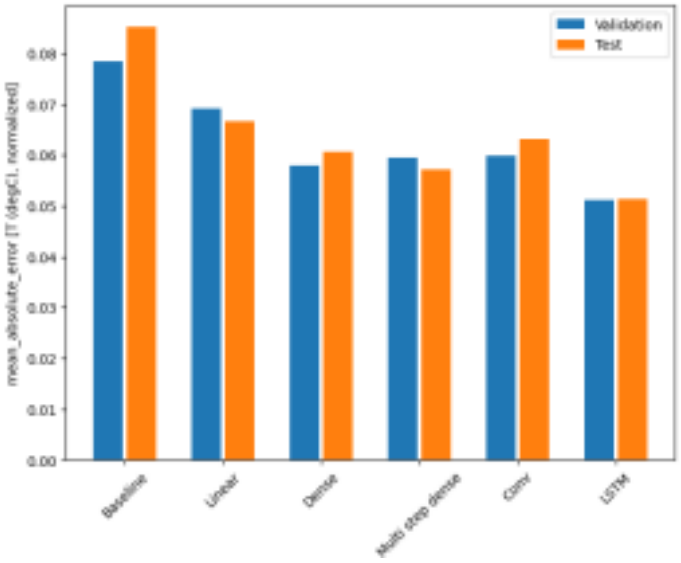


Рисунок 1.13 – Точність роботи різних методів
на тестових наборах даних

Для текстового відображення точності різних моделей використаємо 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 послідовнос тей, 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=&ar e a=&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 послідовнос тей.

1.4.3 Розробити програмне забезпечення для прогнозування ча сових 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 засо би 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, вико ристовувані математичні моделі для прогнозування часових рядів, не обхідність подання результатів у вигляді графіків та ін.). 1.4.3.3 Здійснити проєктування архітектури програмного забез печення для прогнозування часових рядів.

1.4.3.4 Виконати конструювання програмного забезпечення. 1.4.3.5 Виконати тестування розробленого програмного забезпе чення. Здійснити прогнозування на основі обраної послідовності вхід них даних за допомогою різних методів (хоча б двох). Результати про гнозування 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 ря ди?

1.6.4 Що таке прогнозування часових рядів?

1.6.5 З якою метою виконують попередню обробку даних до си нтезу 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 персептрони 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 ме режі для прогнозування часових рядів?

1.6.17 З якою метою виконується нормалізація даних при синте зі моделей для прогнозування часових рядів?

1.6.18 Що таке вікна даних? Як відбувається перетворення вхід ного 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 моде лей 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 зобра жень.

2.2 Короткі теоретичні відомості

Однією з важливих галузей штучного інтелекту є комп’ютерне бачення (Computer Vision, CV). **Комп’ютерне бачення** (комп’ютер ний 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 склада ється 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 шабло нів зображення з метою розпізнавання зображення як окремого об’єкту [10]. Поширеним прикладом розпізнавання зображень є опти чне 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 хара ктеристик (ознак). Внаслідок цього отримується цифрове зображення, яке складається з пікселів (рис.2.1). Кожен піксель представляється числом або набором чисел (діапазон цих чисел називається глибиною кольору або бітовою глибиною). Глибина кольору вказує на максима льну 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 використову ють 24-бітний колір або вище. Зображення з роздільною здатністю 1024 × 768 являє собою сітку з 1024 стовпцями та 768 рядками, яка, отже, містить 1024 × 768 = 0,78 мегапікселя [10].

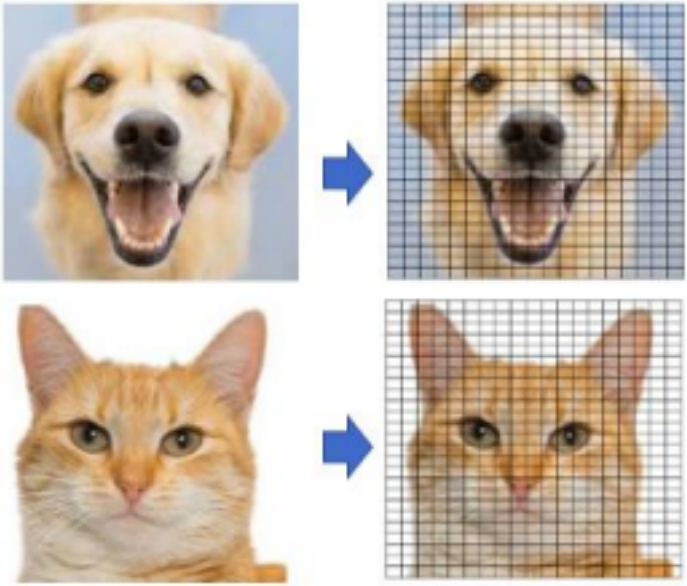


Рисунок 2.1 – Витягання піксельних функцій із зображення [10]

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

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

«кіт», «риба», «птах» та ін. (рис. 2.2). Чим більше зображень буде ви користано для кожної категорії, тим краще можна навчити модель ви значати відповідну категорію (клас) зображення. На цьому етапі відо мою є категорія, до якої належить кожне зображення [10]. Ці дані (зо браження та категорії, до яких вони відносяться) будуть використову ватися на наступному етапі для навчання моделі.

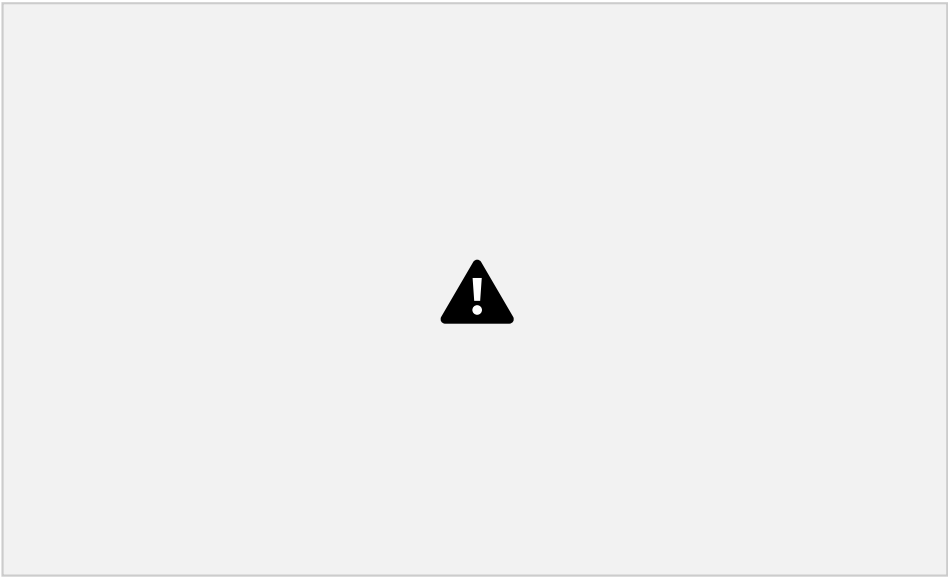


Рисунок 2.2 – Формування навчальної вибірки [10]

Етап 3. Моделювання (синтез розпізнавальної моделі).

На цьому етапі відбувається синтез моделі (підбір значень її па раметрів) шляхом її навчання до класифікації наявних зображень (зо бражень, поданих у навчальній вибірці). На рис.2.3 показано, як мо дель навчається за допомогою попередньо позначених зображень (зо бражень, для яких відомими є категорії, класи). Зображення у витяг нутих 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].

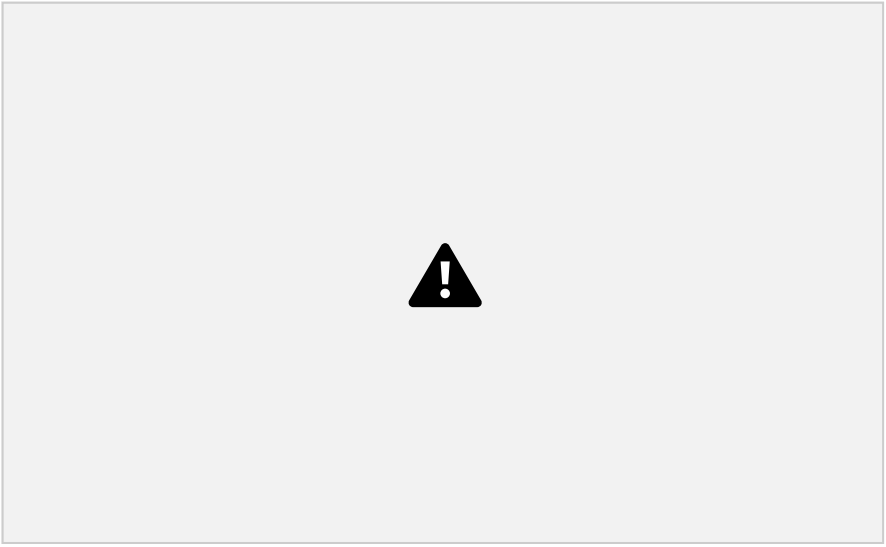


Рисунок 2.3 – Навчання розпізнавальної моделі [10]

Етап 4. Використання синтезованої моделі для розпізнавання нових зображень.

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

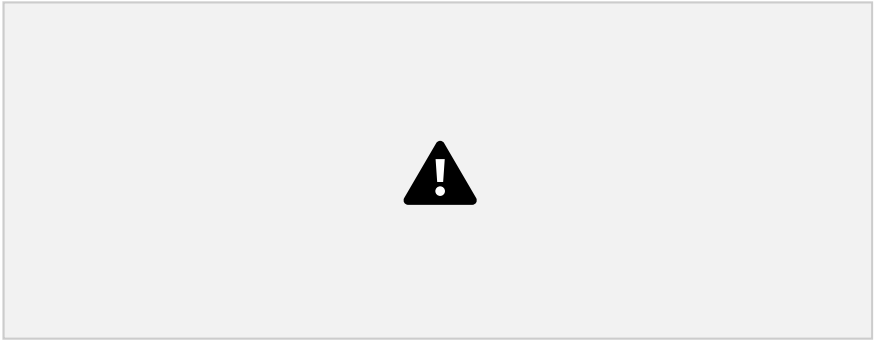


Рисунок 2.4 – Використання синтезованої моделі для розпізнавання нових зображень [10]

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

Рисунок 2.5 – Застосування згорткових нейронних мереж розпізнаван ня зображень [10]

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

Шар згортки (convolution layer). Першим кроком, який викону ють мережі CNN, є створення багатьох маленьких елементів, які нази ваються ознаками (features), як-от квадрати розміром 2x2. Кожна ознака характеризує певну форму вихідного зображення. При скану ванні зображення за допомогою кожної ознаки розраховується відпо відна оцінка частини зображення. Якщо збіг є ідеальним, у цієї части ни зображення (наприклад, квадрату розміром 2x2), їй присвоюється у відповідність високий бал. Якщо збіг є низьким (або взагалі відсут ній), 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 відфі

льтроване зображення з високими та низькими оцінками під час ска-

нування вихідного зображення. Наприклад, червоне поле знайшло чо тири області на оригінальному зображенні, які ідеально збігаються з об’єктом, тому для цих чотирьох областей оцінки високі. Рожеві пря мокутники – 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].

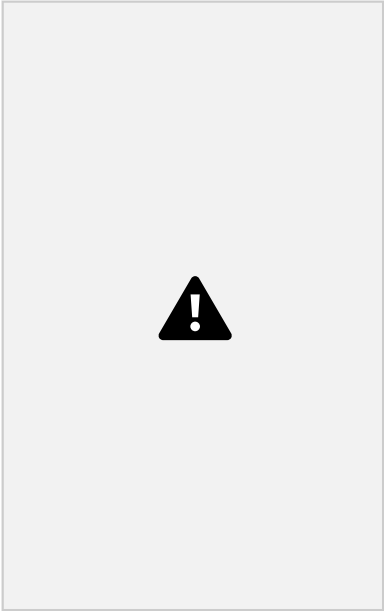


Рисунок 2.6 – Перетворення зображень за допомогою шару згортки [10]

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

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

на 2x2 рамці 1x1 у новому зображенні. Як показано на рисунку, мак симальне значення в першому вікні 2x2 є високим балом (позначеним червоним кольором), тому високий бал призначається полю 1x1 ново го зображення. Поле 2x2 переміщується до другого вікна, де є високий бал (червоний) і низький бал (рожевий), тому високий бал признача ється наступному полю 1x1. Наступному полю призначається світло червоний колір. Після об’єднання створюється новий стек менших відфільтрованих зображень [10].

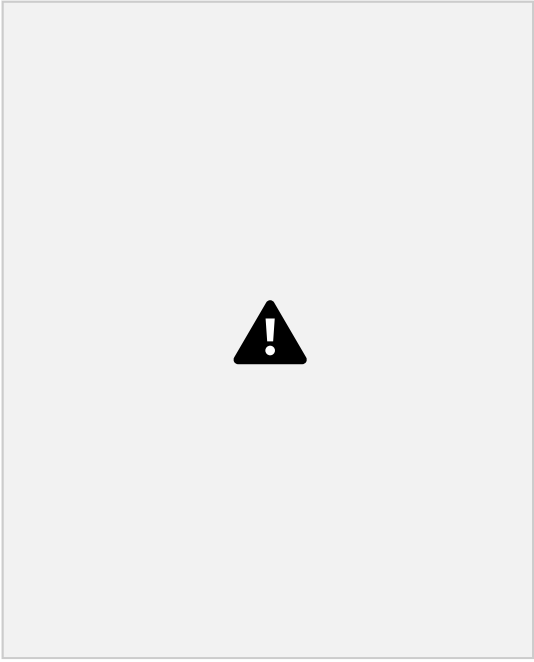


Рисунок 2.7 – Перетворення зображень за допомогою шару Max Pooling [10]

Повноз’вязний шар (фінальний шар). На цьому шарі відбуваєть ся

розділення менш відфільтрованих зображень та складання їх в один список, як показано на рис.2.8. Кожне значення в єдиному списку пе редбачає 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].

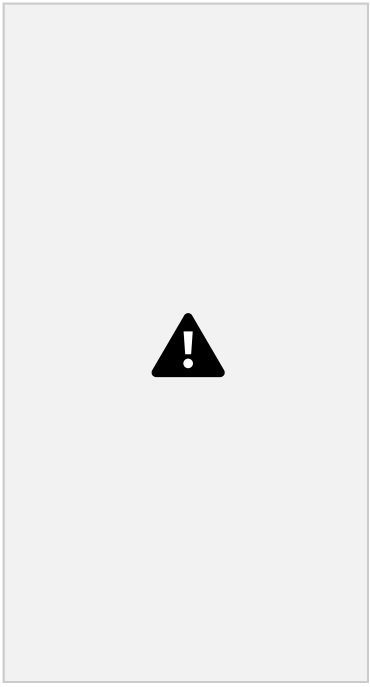


Рисунок 2.8 – Робота останнього, повноз’вязного шару при розпізна ванні зображень [10]

Відзначимо, що багатошарові нейронні мережі складають вихі дне зображення у список і перетворюють його на вхідний шар. Інфор мація між сусідніми пікселями при цьому може не зберігатися. На ві дміну 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 гли бокі 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 прогно зування, виділення ознак і точного налаштування.

Також у відкритому доступі є досить велика базу даних зобра жень 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 зображен ні/сцені та ідентифікувати кожен об'єкт. Виявлення об'єктів широко використовується для виявлення облич 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 ба гатьох 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 алгори тми 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 ал горитмів, заснованих на нейронних мережах та глибокому навчанні, яке також базується на машинному навчанні, потребує великої кілько сті математичних розрахунків [9].

Тому розроблено багато програмних бібліотек, що дозволяють розробникам програмного забезпечення легко інтегрувати сучасні те хнології комп'ютерного зору у розроблювані програми, використову

ючи 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 за вантажити 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 ко ректним, побудуємо перші 25 зображень із навчального набору та ві добразимо 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).

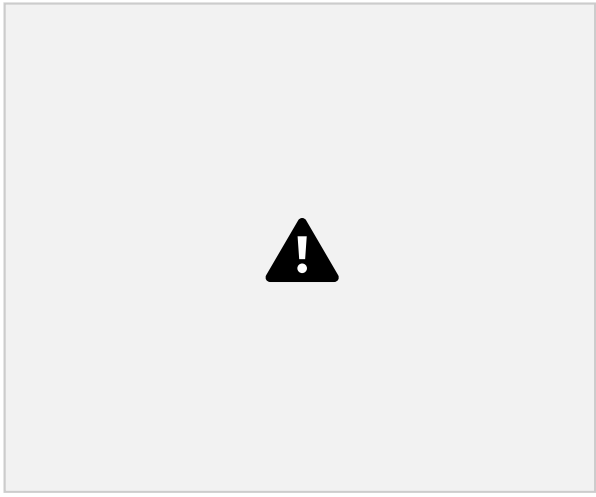


Рисунок 2.9 – Фрагмент вхідних даних

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 (висо та_зображення, ширина_зображення, канали_кольору). У цьому прик ладі 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 викори стовується 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 (MaxPooling (None, 6, 6, 64) 0  2D)

 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 мере жу. Кількість вихідних каналів для кожного рівня Conv2D контролю ється 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 використову ються у багатошарових нейромоделях) для виконання класифікації. Dense-шари приймають вектори як вхідні дані (які є одновимірними), тоді як поточний вихід є 3D-тензором. Тому спочатку необхідно звес ти (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 необ хідно 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 архі

тектуру 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 (MaxPooling (None, 6, 6, 64) 0  2D)

 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 відо браження 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 класифіка ції зображень, що не входили у навчальну вибірку.

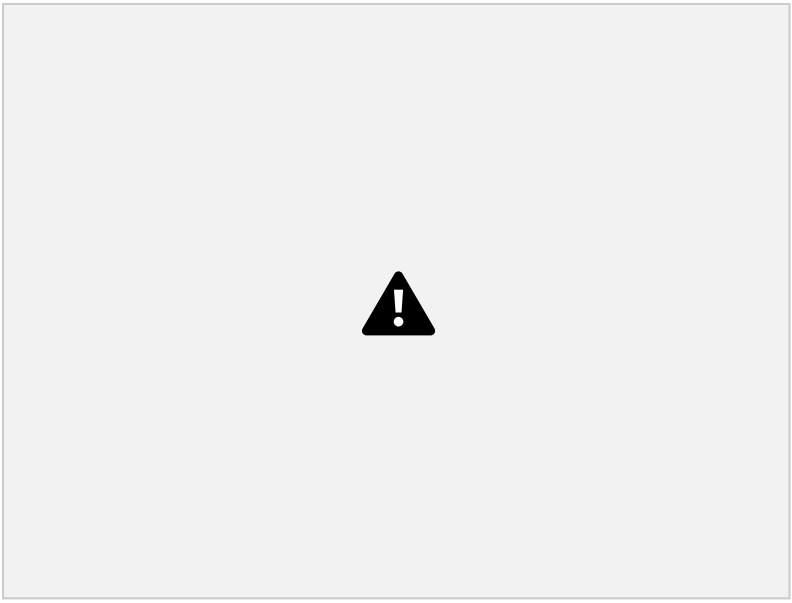


Рисунок 2.10 – Графік залежності точності розпізнавання зображень за допомогою синтезованої моделі від номеру ітерації

Також рекомендовано ознайомитися з іншим прикладом вико ристання 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 реалізова ні та є загально доступними, для пошуку об'єктів (object detection) на графічних зображеннях. Задача синтезу таких моделей і доволі склад ною 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 гра фічних 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 про грами 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 інформа цію для розпізнавання відповідних об'єктів (обличчя, очі та ін.) на зо браженнях. У даному випадку заздалегідь підготовлені файли haarcascade_frontalface_default.xml та haarcascade_eye.xml, які знахо дяться 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 тони 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 pho to.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 графіч них зображеннях може бути розроблено за допомогою бібліотеки ImageAI [9], написаної на мові Python з використанням засобів бібліо теки 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/

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 по дальшої 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 засо би 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, вико ристовувані математичні моделі для обробки зображень, необхідність подання результатів у вигляді графіків та ін.).

2.4.3.3 Здійснити проєктування архітектури програмного забез печення для обробки зображень.

2.4.3.4 Виконати конструювання програмного забезпечення.

2.4.3.5 Виконати тестування розробленого програмного забезпе чення. Подати результати тестування розробленого програмного за безпечення для обробки зображень у зручному для подальшого аналі зу 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 багатошарові персептро

ни?

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 ви явлення 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 тексто вої 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 зда тність читати, розуміти та виводити (генерувати) людську мову. До завдань обробки природної мови відносять визначення настроїв у тек сті, машинний переклад, перевірку орфографії та інші завдання [21]. У процесі обробки природної мови часто використовуються методи лін гвістики, існує чотири етапи обробки мови: морфологія (спосіб тво рення слів і їх зв'язок з іншими словами), синтаксис (як ці слова поєд нуються в реченні), семантика (як значення слів розкривається через граматику та лексичне значення), прагматика (значення слів у кон тексті).

У більшості випадків у процесі обробки природної мови її спо чатку 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, a, the, to, etc.). Та кож попередньо над текстом можуть виконувати лематизацію (Lemmatization), стеммінг (Stemming), токенізацію (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 при значає набір попередньо визначених категорій деякому тексту. Текс тові класифікатори можна використовувати для організації, структу рування та класифікації майже будь-якого типу тексту – від докумен тів, 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 орга нізовувати за настроями; і так далі [22].

Класифікація текстів є одним із основних завдань обробки при родної мови з широким застосуванням, таким як аналіз настроїв, мар кування 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 навчан ня з учителем (supervised machine learning algorithms), тому для на вчання моделі необхідні позначені даними (labeled data). Таким чином дані являють собою множину текстів, кожен з яких позначається пев ною категорією (міткою, класом, тегом). При цьому кожен текст (на приклад, якийсь коментар або пост у соцмережі) відноситься тільки до одного класу (наприклад, позитивний коментар або негативний коме нтар). До одного класу можуть відноситися багато текстів [23]. Етап 2. Фільтрація та обробка даних. Оскільки моделі машинно го 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 мо жуть обробляти дані лише на рівні токенів і не можуть розуміти й об робляти цілі речення [23].

У процесі фільтрації алгоритми машинного та глибокого на вчання можуть розуміти лише числові значення, тому на етапі попе редньої 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 мо жуть кодувати значення даного слова. Використовуються такі методи вбудовування слів [23]:

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

– GloVe (Global Vector) – модель машинного навчання без учи теля 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 ал горитм 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 по трібна для того, щоб модель легко засвоювала ці дані. Для цього на етапі попередньої обробки доцільно видалити непотрібні ознаки, від фільтрувати 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 да них. Часто навчальний набір даних складає 80% (або іншу частку) за даного 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 таки [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). Ал

горитми глибокого навчання вимагають набагато більше навчальних даних, ніж традиційні алгоритми машинного навчання, проте у них немає порогу для навчання з навчальних даних, а класифікатори ста ють кращими, чим більше даних ви їм надаєте.

Також можуть застосовуватися й інші моделі, наприклад, логіс тична регресія, моделі, засновані на правилах, гібрідна модель та ін. Етап 5. Тестування та перевірка працездатності моделі. На цьо му етапі перевіряється якість моделі, використовуючи набір тестових даних, визначений на етапі 3. Тестовий набір даних проганяється че рез модель, внаслідок чого для кожного тексту з тестового набору отримуються значення відповідних класів або міток. Ці класи порів нюється з фактичними значеннями, на основі чого розраховується то чність 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 (Accuracy) – 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 не вірно (клас, до якого було віднесено текст за допомогою моделі, не збігається з фактичним класом);

– правильність, влучність (Precision) – відсоток текстів, які кла сифікатор 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 прави льності та повноти.

Іноді, у випадку отримання незадовільних результатів тестуван ня моделі (низька точність на тестових даних) виконують налашту вання алгоритму машинного навчання. Для цього корегують різні гі перпараметри 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 гене ративний штучного інтелекту, що дозволяє створювати новий текст, зображення, відео тощо на основі наявних вхідних даних. Генератив ний 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 приро дної мови. Генерування тексту, як правило, виконується на основі мо делей глибокого навчання. Як приклад, дослідники тренують генера тивні змагальні мережі (Generative adversarial networks, GAN), які є генеративними моделями, що складаються з генератора та дискримі натора й використовуються для створення синтетичних результатів для генерації тексту [25].

Таким чином, для створення програмного забезпечення для ге нерація текстової інформації необхідно синтезувати відповідну мо дель, 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 ін телекту, які можуть генерувати текст та є досить складними. Однією з таких моделей є GPT (Generative Pre-trained Transformer), або генера тивний попередньо навчений трансформатор. Ця мовна модель, ство

рена OpenAI, має різні моделі, включаючи GPT-3. GPT-3 – це досить велика та складна модель із понад 175 мільярдами параметрів, яка на вчалася на різних джерелах даних, включаючи книги, статті та схови ща коду, щоб створювати реалістичні та схожі на людські тексти. За допомогою GPT-3 можна створювати резюме, відповідати на запитан ня та робити переклади [25].

Іншим підходом до генерування тексту є моделі на основі шаб лону (template-based model). На відміну від GPT-3, ці моделі не пра цюють 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, соціа льних мережах, електронних листів, описів продуктів, слоганів тощо. Текстові генератори застосовується для [25]:

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

– узагальнення тексту – створення резюме довгих текстів. На приклад, створення інформаційних бюлетенів, узагальнення внутріш ніх документів компанії, допомога викладачам у підготовці навчаль ного матеріалу шляхом надання їм узагальненого змісту джерел, до помога у огляді літератури у дослідницькому контексті та ін. Також текстові генератори можуть використовуватися у SEO-оптимізації щоб зробити статтю більш оптимізованою для пошукових систем, те кстові 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 клю чових 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 реа

льного часу, а також готувати персоналізовані відповіді служби підт римки клієнтів.

3.3 Приклад розробки програмного забезпечення для оброб ки 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 ос нові 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 забез печення:

```
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/) для бінарної класифікації текс тів. 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 рецен зія (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 зава нтаженого 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 дові льний 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 від
повідні 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 синте зованої 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 токе нів. Після доповнення невідомими токенами ('[UNK]') вони сортують ся 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 індек си. Тензори (набори) індексів для кожного тексту у наборі даних до повнюються 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 понов леному 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**. Врахо вуючи, 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 те ксту подана на рис.3.1 [27].

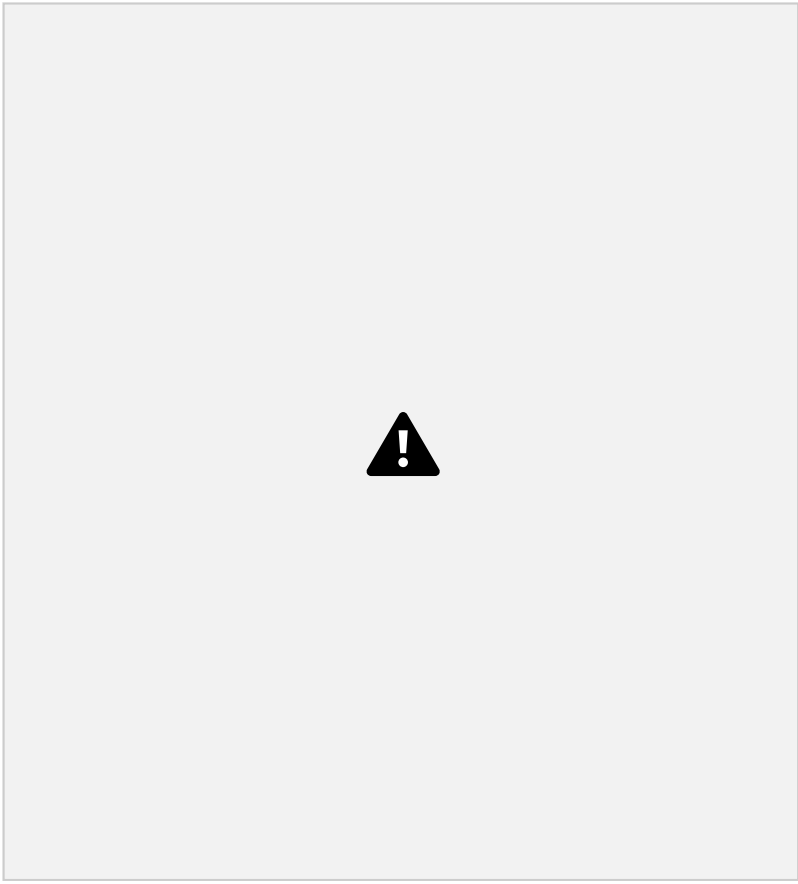


Рисунок 3.1 – Архітектура рекурентної нейронної мережі для класифікації тексту

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 по слідовності індексів слів на послідовності векторів. Ці вектори підда ються 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'єднучи 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, логарифміч на 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 іте рації (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()
```

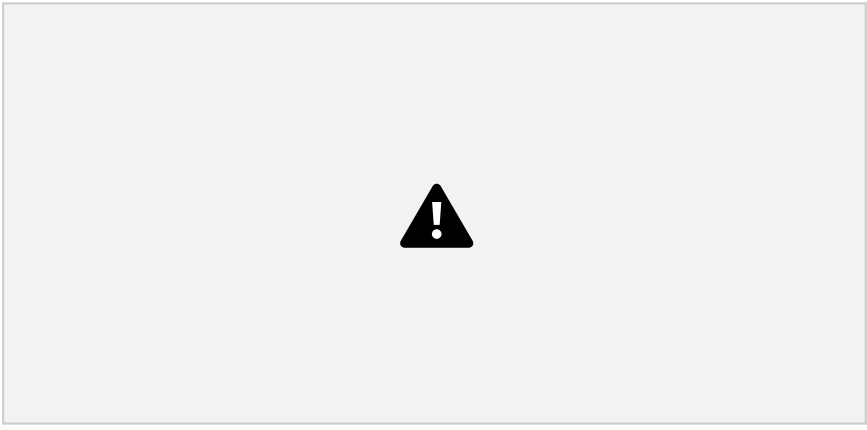


Рисунок 3.2 – Графіки залежності точності та втрат від номеру ітерації

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 класи фікації текстової інформації, поданим у джерелі [28].

3.3.2 Розробка програмного забезпечення для генерації текс тів

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

Отже, завдання синтезу моделі для генерування тексту буде по лягати у тому, щоб враховуючи символ чи послідовність деяких сим волів, визначити, який наступний символ є найбільш вірогідним. Вхі дними даними моделі буде послідовність символів, на основі якої мо дель буде прогнозувати вихідні дані – наступний символ кожному ча совому 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 навчально го тексту**. Перед навчанням моделі необхідно перетворити рядки на числове подання. Шар tf.keras.layers.StringLookup може перетворюва ти кожен символ на числовий ідентифікатор. Для цього спочатку пот рібно розділити текст на токени.

```
example_texts = ['abcdefg', 'xyz']
chars = tf.strings.unicode_split(example_texts, in
put_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 по слідовність міститиме seq_length символів із тексту. Для кожної вхід ної 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 бу де "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 сим
воли 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 розмі рами embedding_dim ;

– tf.keras.layers.GRU – тип RNN з кількістю нейронів, що дорів нює 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 ймовір ність) 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 логари фмічну ймовірність наступного символу (рис. 3.3).

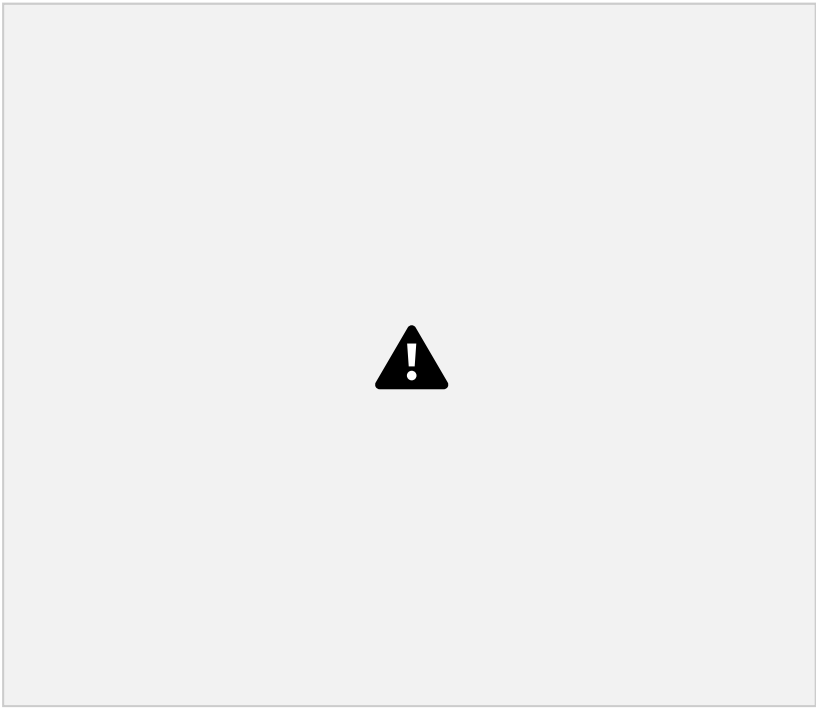


Рисунок 3.3 – Схематичне подання моделі для генерації тексту [29]

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

```
for input_example_batch, target_example_batch in da
taset.take(1):
 example_batch_predictions = mod
el(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 компіля ція** (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 мо делі, 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,
```