Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи
ОС магістра

на тему: «Дослідження засобів штучного інтелекту для розпізнавання усного мовлення»
за освітньою програмою «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»

Виконала: студентка групи ПЗ2421: /КатеринаПАЗИКА/

Керівник: /Олена КУРОП'ЯТНИК/

Нормоконтролер: /Світлана ВОЛКОВА /

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

Дніпро – 2026 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Master's Thesis

on the topic: « Research into artificial intelligence tools for speech recognition»
according to educational curriculum «Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ2421: /Kateryna PAZYKA/

Scientific Supervisor: /Olena KUROPIATNYK/

Normative controller: /Svitlana VOLKOVA/

Dnipro – 2026

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: магістр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)

Дата _____

ЗАВДАННЯ

на кваліфікаційну роботу Магістр

студенці Катерині ПАЗИКІ

1. Тема роботи: «Дослідження засобів штучного інтелекту для розпізнавання штучного мовлення»
   затверджені наказом № 1401 ст від 02.10.2025
2. Строк подання студентом роботи: 05.01.2026 – 11.01.2026
3. Вихідні дані до роботи: набори даних для навчання та тестування моделей розпізнавання мовлення.
4. Зміст пояснювальної записки (перелік питань до розробки):
   4.1 Огляд проблеми розпізнавання усного мовлення;
   4.2 Вибір нейронних моделей та методів для автоматичного розпізнавання мовлення;
   4.3 Проєктування та реалізація системи розпізнавання мовлення;
   4.4 Експериментальне дослідження та аналіз результатів розпізнавання мовних команд.
   5 Перелік графічного матеріалу:
   5.1 Презентація для доповіді;
   5.2 Демонстраційне відео роботи програми.

КАЛЕНДАРНИЙ ПЛАН

№ пор.	Зміст роботи (розділу)	Термін виконання розділів роботи	Примітка
1	Вступ	03.10.2025 – 25.10.2025	
2	Аналіз сучасного стану вирішення обраної задачі та програмно-апаратного забезпечення	26.10.2025 – 09.11.2025	
3	Збір вимог до програми, вибір засобів штучного інтелекту для дослідження	10.11.2025 – 16.11.2025	30%
4	Зовнішнє та внутрішнє проектування	17.11.2025 – 07.12.2025	
5	Розробка програмного забезпечення	08.12.2025 – 14.12.2025	60%
6	Тестування та налагодження програмного забезпечення	15.12.2025 – 20.12.2025	
6	Проведення експериментального дослідження та аналіз результатів	21.12.2025 – 25.12.2025	
7	Оформлення пояснювальної записки	26.12.2025 – 04.01.2026	
8	Розробка демонстраційних матеріалів	05.01.2026 – 11.01.2026	100%

Дата видачі завдання «2» жовтня 2025 р.

Керівник дипломної роботи КУРОП'ЯТНИК Олена Сергіївна

(підпис) (ПІБ)

Завдання прийняла до виконання ПАЗИКА Катерина Станіславівна

(підпис) (ПІБ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра:

92с., 18 рис., 7 табл., 19 джерел, 4 додатки.

Об’єкт дослідження – процеси автоматичного розпізнавання усного мовлення на основі методів штучного інтелекту.

Предмет дослідження – нейромережеві архітектури та впливу наборів навчальних даних, що визначають показники точності та стабільності розпізнавання.

Мета роботи – дослідити сучасні засоби штучного інтелекту для розпізнавання усного мовлення, проаналізувати їх технічні характеристики та визначити оптимальні рішення для практичного використання.

Методи дослідження – експериментальне навчання та тестування нейромережевих моделей розпізнавання мовлення, порівняльний аналіз отриманих результатів за показниками точності та узагальнювальною здатністю.

Отримані результати – розроблено програмний застосунок, для навчання та тестування моделей розпізнавання мовлення, що дозволяє оцінювати точність роботи різних нейромережевих архітектур та їх стійкість до варіацій аудіоданих. Проведено експериментальне дослідження та визначено показники точності розпізнавання для обраних моделей.

Ключові слова: НЕЙРОННА МЕРЕЖА, РОЗПІЗНАВАННЯ МОВЛЕННЯ, АУДІОСИГНАЛ, ДАТАСЕТ.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Нейронна мережа – це математична модель, яка імітує структуру біологічний нейронних мереж з метою вирішення таких задач як класифікація, генерація, прогнозування, тощо.

Штучний інтелект – це набір методів та технологій, що використовуються для розробки об’єктів (програмних або апаратних), здатних виконувати функції, які зазвичай асоціюються з людським інтелектом, такі як навчання, аргументація, самокорекція.

Розпізнавання усного мовлення – це процес автоматичного перетворення акустичного потоку в цифрову послідовність слів або символів за допомогою спеціальних алгоритмів та математичних моделей.

Цифровий аудіосигнал – це дискретне представлення аналогового звукового сигналу, отримане шляхом його квантування за амплітудою та дискретизації за часом. Він представляє послідовність числових значень, кожне з яких відображає інтенсивність звукового тиску в конкретний момент часу.

Датасет – це структурована сукупність однорідної інформації, відібраної та підготовленої для статистичного аналізу або використання в алгоритмах машинного навчання з метою виявлення закономірностей та побудови прогнозних моделей.

Аугментація даних – метод збільшення обсягу та різноманітності навчальної вибірки шляхом створення модифікованих копій існуючих даних, наприклад додавання шуму, зміна тональності.

Логарифміча мел-сектрограма – частотно-часове представлення аудіосигналу, що використовується як вхідний ознаковий простір для нейронних мереж.

Згорткова нейронна мережа – архітектура глибокого навчання, заснована на операції математичної згортки, яка призначена для автоматичного виділення ієрархічних просторових ознак із вхідних даних.

Рекурентна нейронна мережа – тип архітектури нейронних мереж, що містить зворотні зв’язки, які дозволяють зберігати та використовувати інформацію про попередні стани обробки.

ВСТУП

Актуальність роботи. Швидкий розвиток інтелектуальних систем висуває жорсткі вимоги до надійності алгоритмів обробки звукових сигналів у реальному часі та зумовлює актуальність досліджень у цій сфері. Системи автоматичного розпізнавання мовних команд є важливою складовою сучасних програмних засобів, зокрема систем голосового керування, інтерактивних інтерфейсів та вбудованих систем. Ефективність таких систем значною мірою визначається якістю засобів штучного інтелекту, що використовуються.

У сьогоденні для розпізнавання мовлення широко застосовуються нейромережеві архітектури, які демонструють високі показники точності в межах окремих навчальних вибірок. Проте, на практиці залишається невирішеною проблема узагальнювальної здатності таких моделей, а саме – їх стабільність при зміні складу або структури навчального датасету. Зазвичай оцінка якості моделі обмежується тестуванням на даних, близьких до тренувальних, але це не дозволяє робити висновки щодо надійності моделей у реальних умовах застосування.

Особливої актуальності ця проблема набуває в програмній інженерії, де вибір нейромережевої архітектури та навчальних даних безпосередньо впливає на:

- масштабованість програмних рішень. Моделі, які демонструють гарні результати на малих наборах даних, можуть значно знижувати ефективність на великих вибірках;

- ефективність розробки та розгортання. Різні архітектури мають різну складність обчислень і вимоги до ресурсів;

- адаптованість, тобто здатність моделі зберігати прийнятну якість при варіюванні умов(шум, акценти, тощо).

Огляд наукової літератури показує, що в дослідженнях, присвячених точності моделей розпізнавання мовлення, обмежена увага приділяється

питанню порівняльного аналізу якості моделей при зміні версій навчальних даних та умов експерименту.

Таким чином, актуальною є задача дослідження ефективності та узагальнювальної здатності нейромережевих моделей розпізнавання мовних команд при навчанні на різних датасетах, та аналізу результатів їх тестування. Розв’язання цієї задачі є доцільним для підвищення надійності програмних систем розпізнавання мовлення та розвитку підходів до їх проєктування в сучасній програмній інженерії.

Об’єктом дослідження є процеси розпізнавання мовних команд. Предметом дослідження є нейромережеві архітектури та вплив наборів навчальних даних, що впливають на показники точності та стабільності розпізнавання.

Метою роботи є отримання обґрунтованих рекомендацій щодо вибору засобів штучного інтелекту для розпізнавання усного мовлення, враховуючи архітектури нейронних мереж та характеристики навчальних датасетів, для підвищення точності та стабільності результатів розпізнавання у програмних системах.

Поставлена мета зумовила необхідність вирішення таких завдань:

- проаналізувати сучасні засоби штучного інтелекту для розпізнавання усного мовлення;
- обґрунтувати вибір нейромережевих моделей та навчальних даатсетів для проведення експериментального дослідження;
- реалізувати експериментальне програмне середовище для навчання та оцінки моделей розпізнавання мовних команд;
- провести порівняльний аналіз показників точності та визначити рівень стабільності розпізнавання для обраних моделей на різних навчальних вибірках;

- сформувати рекомендації щодо вибору засобів штучного інтелекту для використання в програмних системах розпізнавання усного мовлення.

Методи дослідження, використані у роботі, базуються на застосуванні експериментального підходу до аналізу засобів штучного інтелекту для розпізнавання усного мовлення. Дослідження проводилось шляхом навчання та тестування нейромережевих моделей з різними архітектурами на стандартизованих наборах даних.

Якість розпізнавання оцінювалась на основі значень точності класифікації, отриманих на навчальній та валідаційній вибірках. Це дозволило проаналізувати вплив архітектури нейронної мережі та характеристик навчальних даних на результати розпізнавання.

Порівняльний аналіз експериментальних результатів використовувався для визначення відмінностей у поведінці моделей та формування висновків щодо доцільності застосування окремих засобів штучного інтелекту в системах розпізнавання мовних команд.

Наукова новизна роботи полягає в уточненні та конкретизації результатів щодо ефективності нейромережевих засобів розпізнавання усного мовлення шляхом їх експериментального порівняння. Отримані результати розширюють існуючі знання про вплив архітектури нейронної мережі та характеристик навчальних даних на показники точності й стабільності розпізнавання мовних команд. Важливе практичне значення мають узагальнені експериментальні висновки, які можуть використовуватись для обґрунтування вибору засобів штучного інтелекту під час проектування систем розпізнавання мовлення.

Практичне значення результатів дослідження полягає в можливості їх використання під час вибору засобів штучного інтелекту для задач розпізнавання мовлення. Отримані експериментальні результати та

узагальнені висновки можуть використовуватись розробниками програмного забезпечення як довідкові дані при проектуванні та налаштуванні систем розпізнавання мовних команд з урахуванням вимог до точності та стабільності роботи. Крім того, результати дослідження можуть бути застосовані в навчальному процесі під час вивчення дисциплін, пов’язаних із машинним навчанням та обробкою мовлення, як приклад порівняльного аналізу нейромережевих підходів.

Основні положення магістерської роботи були представлені на XIX Міжнародній науково-практичній конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості та освіті» м. Дніпро, 18.12.2025 – 19.12.2025 р. [1], на науковому семінарі кафедри КІТ 09.01.2025 року.

1 ОГЛЯД ПРОБЛЕМИ РОЗПІЗНАВАННЯ УСНОГО МОВЛЕННЯ

Розпізнавання мовлення (Automatic Speech Recognition, ASR) є однією з ключових задач систем штучного інтелекту в сучасності. Це процес, спрямований на автоматичне перетворення акустичного мовного сигналу у формалізоване символьне представлення. Задачі ASR знаходять своє застосування в багатьох сферах: голосові асистенти, системи керування програмним та апаратним забезпеченням, автомобільні інформаційні системи, робототехнічні комплекси, тощо. Окремим напрямом є розпізнавання коротких мовних команд, що застосовується у системах з обмеженими обчислювальними ресурсами та високими вимогами до швидкодії, зокрема в системах реального часу та інтернеті речей (IoT).

Не дивлячись на активний розвиток методів штучного інтелекту, задача розпізнавання усного мовлення залишається нетривіальною. Це пов’язано зі складною природою мовного сигналу, який є нестаціонарним, варіативним та дуже залежним від індивідуальних особливостей мовця, інтонації, акценту, темпу мовлення та акустичних умов. Фонові шуми, відмінності записувальних пристроїв, обмеженість навчальних даних для окремих мов створюють додаткові труднощі.

Для вбудованих та мобільних систем, де необхідно забезпечити компроміс між точністю розпізнавання, обчислювальною складністю моделей та енергоспоживанням, задача ASR є особливо складною. У таких умовах вибір архітектури нейронної мережі та способу подання вхідного сигналу є вирішальним для досягнення практично допустимих характеристик системи.

Розпізнавання усного мовлення – міждисциплінарна задача яка поєднує методи обробки сигналів, машинного навчання та програмної інженерії. Дослідження сучасних засобів штучного інтелекту для розв’язання цієї задачі потребує глибокого аналізу як математичних моделей, що лежать в основі

алгоритмів розпізнавання, так і властивостей навчальних даних, які мають безпосередній вплив на якість та стабільність отриманих результатів.

1.1 Усне мовлення як об’єкт обробки в системах штучного інтелекту

Усне мовлення є одним із основних засобів комунікації людини та водночас складним багатокомпонентним явищем, що поєднує фізичні, біологічні та лінгвістичні процеси. Із точки зору автоматизованої обробки, мовлення розглядається як фізичний акустичний сигнал, що містить інформацію про мовний зміст, зовнішні умови середовища та індивідуальні особливості мовця.

Мовлення фізично виникає внаслідок коливання повітря, що зумовлено роботою голосового апарату людини. Це нестаціонарний процес, його статистичні характеристики з часом змінюються залежно від інтонації, темпу мовлення, емоційного стану мовця та вимовлюваних фонем. Саме ця особливість і є основним ускладненням при спробах побудови універсальних алгоритмів аналізу та розпізнавання.

У комп’ютерних системах для представлення мовлення необхідно виконувати перехід від аналогового сигналу до цифрової форми. Цей процес включає дискретизацію за часом та квантування за амплітудою [2]. У результаті неперервний сигнал замінюється дискретною послідовністю відліків, що визначається частотою дискретизації:

$$x[n] = x(nT_s) \tag{1.1}$$

де $x[n]$ – значення сигналу в n-й момент дискретного часу, n – номер відліку, T_s – період дискретизації, що визначається частотою дискретизації

Вибір частоти дискретизації є компромісом між обчислювальними ресурсами та точністю представлення мовного сигналу, тому що занадто велика кількість відліків ускладнює подальшу обробку.

Мовлення розглядається штучним інтелектом як носій структурованої інформації, яка повинна автоматично інтерпретуватися. На відміну від задач класичної обробки сигналів, де на меті стоїть стиснення або фільтрація, у задачі розпізнавання мовлення необхідно відновити абстрактний мовний зміст – слова, команди або фонетичні одиниці – на основі оброблюваних акустичних даних. Виділяють три рівні варіативності .які створюють основні труднощі для систем штучного інтелекту:

- акустична: зумовлена відмінностями у записувальних пристроях, акустиці приміщення та наявністю фонового шуму;
- інтрадикторна: зміни характеристик мовлення одного й того самого мовця залежно від його емоційного стану, темпу та фізичного здоров’я;
- інтердикторна: відмінності в інтонації, акценті та тембрі голосу між різними дикторами.

У цьому контексті традиційні алгоритми, засновані на жорстких правилах, показують низьку стійкість. Саме тому сучасні системи переходять до навчання за даними, де нейромережеві моделі повинні виділяти інваріантні ознаки мовлення, що залежать від перелічених факторів. Хоча така залежність від даних підвищує вимоги до якості представлення сигналу та подальших етапів його обробки.

1.2 Представлення мовного сигналу та попередня обробка

Мовний сигнал для подальшого аналізу повинен бути представлений у цифровій формі та підданий попередній обробці. Мета цього етапу –

перетворення сирого часового сигналу в інформативне подання, яке є придатним для подальшого математичного аналізу і навчання моделей.

Після дискретизації неперервного мовного сигналу $x(t)$ (див. формулу 1.1) з періодом дискретизації T_s формується дискретна послідовність відліків $x[n]$ (див. формулу 1.2). Отриманий сигнал є одновимірним часовим рядом, значення якого відображають зміну звукового тиску в часі. Для того щоб коректно відтворити спектральний склад людського мовлення зазвичай використовується частота дискретизації 8 – 16 кГц. Але безпосереднє використання часових відліків $x[n]$ (див. формулу 1.2) як вхідних даних для алгоритмів розпізнавання не є ефективним через високу варіативність сигналу, його чутливість до зсувів у часі, відсутність явної інформації про частотний склад мовлення.

Мовний сигнал нестаціонарний у довгостроковій перспективі, але його можна вважати квазістаціонарним на коротких часових інтервалах (10-30 мс). Це дозволяє застосувати підхід короткочасного аналізу, при якому сигнал поділяється на перекривні фрагменти – фрейми.

Нехай $x[n]$ – дискретний мовний сигнал, тоді k-й фрейм формується так:

$$X_k[n] = x[n + kH], \quad n = 0, 1, \ldots, N - 1 \qquad (1.2)$$

де k – індекс фрейму, H – крок зсуву між сусідніми фреймами, N – довжина фрейму у відліках.

Фреймування дозволяє локалізувати аналіз у часі та підготувати сигнал до переходу в частотну область.

Пряме обмеження сигналу кінцевим інтервалом є причиною спектральних спотворень. Щоб зменшити це спотворення кожен фрейм множиться на віконну функцію:

$$x_k^{(w)}[n] = x_k[n] \times w[n] \quad (1.3)$$

де $w[n]$ – віконна функція довжини N

В обробці сигналів віконна функція – це математична функція, яка має нульове значення за межами деякого вибраного інтервалу, зазвичай симетричного навколо середини інтервалу. У задачах розпізнавання мовлення найпопулярнішими є вікна Ханна або Геммінга, які зменшують вплив країв фрейму та покращують спектральну локалізацію [3].

Для кожного фрейму обчислюється дискретне перетворення Фур'є (формула 1.4), у результаті якого сигнал переходить з часової області в частотну [4].

$$X[k] = \sum_{n=0}^{N-1} \times\ e^{-j\frac{2\pi nk}{N}} \quad (1.4)$$

де $X[k]$ - комплексна амплітуда для частоти k, N – загальна кількість відліків у вхідній послідовності, n – індекс відліку, j – уявна одиниця

Послідовне застосування цього перетворення до всіх фреймів формує короткочасне перетворення Фур'є, що дозволяє одночасно аналізувати частотний склад сигналу та його зміну в часі. Короткочасне перетворення Фур'є – фундаментальний інструмент у задачах аналізу мовлення, воно відображає динамічну природу мовного сигналу, де фонетичні одиниці характеризуються спектральними формами і їх часовою зміною.

Для подальшої обробки мовлення використовується спектрограма, що визначається як квадрат модуля короткочасного перетворення Фур'є:

$$S(k,m) = |X_k[m]|^2 \quad (1.5)$$

де k – індекс фрейму, m – індекс частотного біну, $X_k[m]$ – комплексне подання значення короткочасного перетворення Фур'є.

Спектрограма – двовимірне подання мовного сигналу, що відображає розподіл енергії сигналу в часово-частотній області. Кожна точка спектрограми характеризує інтенсивність певної частоти в конкретний момент часу. Спектрограма є інформативним поданням, оскільки:

- форматна структура голосних відображається у вигляді стабільних смуг енергії;
- приголосні звуки характеризуються різкими спектральними змінами;
- часові переходи між звуками зберігають свою локалізацію.

Спектрограма може бути інтерпретована як матриця ознак, що є природним вхідним поданням для згорткових нейронних мереж. Такий спосіб подання мовного сигналу забезпечує баланс між інформативністю, обчислювальною складністю та стабільністю навчання, що зумовлює їх широке застосування в сучасних системах штучного інтелекту.

1.3 Вилучення ознак

Метод розпізнавання мовлення на основі ознак припускає, що кожен об'єкт , що підлягає класифікації, можна охарактеризувати вектором ознак та безліччю значень ознак. Відбір ознак заснований на силі їх впливу на якість розпізнавання . Систему ознак обирають інтуїтивно. Ознаки мовлення повинні:

- зберігати інформацію про спектральний склад сигналу;
- бути стійкими до адитивного шуму та змін амплітуди;
- враховувати психоакустичні особливості сприйняття звуку людиною;
- мати фіксовану розмірність для подальшої обробки моделями машинного навчання.

Ці вимоги визначили вибір спектральних та кепстральних ознак як основних у системах ASR.

Людський слух сприймає частоти нелінійно: роздільна здатність є вищою в низькочастотній області та знижується зі зростанням частоти. Для моделювання такої властивості використовують mel-шкалу, яка є нелінійним відображенням частотної осі (рисунок 1.1)

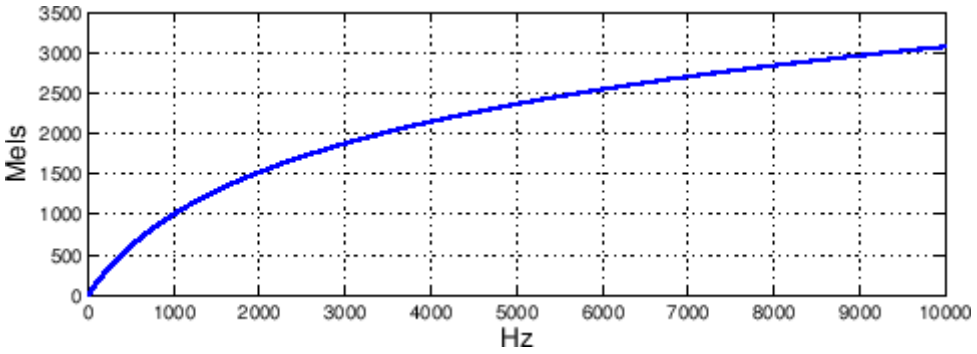

Рисунок 1.1 – Графік шкали мелів

Перехід від лінійної частоти *f* (у герцах) до mel-частоти здійснюється за формулою нижче

$$m(f) = 2595 \times \log_{10}(1 + \frac{f}{700}) \quad (1.6)$$

де f - частота сигналу в герцах, $m(f)$ – відповідна частота в mel-шкалі.

На основі спектра сигналу формується mel-фільтровий банк. Він складається з перекривних трикутних фільтрів, рівномірно розташованих у mel-шкалі. Для кожного фільтра обчислюється енергія:

$$E_i = \sum_k |X[k]|^2 H_i[k] \quad (1.7)$$

де $X[k]$ – спектральні коефіцієнти сигналу, $H_i[k]$ – і-й mel-фільтр.

Подальше застосування логарифмічного перетворення:

$$\bar{E}_i = \log(E_i) \tag{1.8}$$

зменшує динамічний діапазон значень та наближає подання до особливостей сприйняття гучності людиною.

Мел-кепстральні коефіцієнти (MFCC) – класичний набір ознак, які часто застосовуються в системах розпізнавання мовлення [5]. Їх обчислюють шляхом застосування дискретного косинусного перетворення (DCT) до log-mel спектру:

$$c_n = \sum_{i=1}^{N} \bar{\mathrm{E}}_i \cos[\frac{\pi N}{n}(i - 0.5)] \tag{1.9}$$

де c_n – n-й MFCC коефіцієнт, N – кількість mel-фільтрів.

MFCC дозволяють декорелювати спектральні ознаки та зменшити розмірність вхідних даних при цьому зберігши основну інформацію про формат і структуру мовлення. Через це вони довго були стандартом у класичних ASR-системах. Водночас застосування косинусного перетворення призводить до втрати просторової структури спектра, що обмежує ефективність MFCC у глибоких нейромережах.

Отже, MFCC здатні забезпечувати компактне на добре інтерпретоване подання, але втрачають просторову структуру спектру. Log-mel спектри та спектрограми зберігають усю часово-частотну інформацію, що дає змогу нейромережам автоматично визначати релевантні патерни. Ця властивість log-mel спектрів і робить їх основним вибором для згорткових та рекурентно-згорткових архітектур, які використовуються в системах розпізнавання мовних команд.

1.4 Типологія та класифікація систем автоматичного розпізнавання мовлення

Різноманіття прикладних задач, умов використання та обмежень обчислювальних ресурсів зумовлює існування широкого спектра ASR-систем. Вони значно відрізняються між собою за принципами побудови, вимогами до вхідних даних та іншими характеристиками. Для систематизації підходів до розробки та аналізу ASR-систем у науковій літературі їх класифікують за певними ознаками, які відображають і властивості мовлення, і особливості алгоритмів реалізації процесу розпізнавання.

За характером мовного сигналу, що розпізнається, системи розпізнавання мовлення діляться на системи для дискретного на неперервного мовлення. Дискретне мовлення – те, у якому між окремими словами або мовними командами присутні явно виражені паузи. Цей тип мовлення характерний для систем керування за допомогою голосових команд, де користувач вимовляє окремі слова чи фрази з паузами між нами. Наявність таких пауз значно спрощує задачу сегментації мовного сигналу та зменшує складність алгоритмів розпізнавання. Неперервне мовлення, навпаки, є природньою формою людського мовлення, тому чітких меж між словами немає. Для неперервного мовлення задача розпізнавання є складною через необхідність одночасного визначення меж слів, їх послідовності та відповідності мовному словнику. Системи розпізнавання неперервного мовлення мають потребу в більш складних моделях та великому обсягу навчальних даних [6].

Залежно від користувачів, для яких призначена система, ASR-системи поділяють на дикторозалежні та дикторонезалежні. Дикторозалежні системи розпізнавання мовлення налаштовуються на одного користувача або обмежену групу користувачів. Такі системи зазвичай демонструють вищі показники точності розпізнавання за рахунок адаптації до індивідуальних

особливостей диктора, таких як тембр голосу, швидкість мовлення, інтонація, акцент, тощо. Проте, дикторозалежні системи обмежуються в застосуванні лише тими сценаріями, у яких можливе попереднє навчання або калібрування системи для кожного користувача. Дикторонезалежні системи орієновані на роботу з довільними користувачами без попереднього навчання. Для цього вони мають бути стійкими до великої міждикторної варіативності, це досягається шляхом використання великих та різноманітних навчальних вибірок і вибором моделей з високою узагальнюючою здатністю. Дикторонезалежні системи є більш універсальними і найчастіше застосовуються у сучасних прикладних рішеннях.

Ще однією важливою характеристикою ASR-систем є обсяг словника, тобто кількість мовних одиниць, які система здатна коректно розпізнавати. Системи з малим словником зазвичай працюють з обмеженим набором слів або команд. Вони часто використовуються у задачах голосового керування, де кількість можливих варіантів відповідей є визначеною. Малий словник скорочує час обробки та підвищує надійність розпізнавання. Системи з великим словником призначені для розпізнавання широкого спектра слів, включно з рідкісними та контекстно залежними мовними одиницями. Вони потребують складних мовних моделей і механізмів узагальнення, спроможних працювати з великим обсягом даних. Чим більший словник, тим складніша задача та вищі вимоги до обчислювальних ресурсів [6]

Розглянуті класифікаційні ознаки безпосередньо впливають на вибір математичних моделей. Алгоритмів обробки сигналу та методів навчання, що використовуються у ASR-системах. Тип мовлення, множина дикторів та обсяг словника визначають складність задачі, вимоги до якості навчальних даних. Обчислювальних ресурсів та можливість застосування тих чи інших підходів до моделювання.

1.5 Основні підходи до автоматичного розпізнавання мовлення

У процесі розвитку систем автоматичного розпізнавання мовлення сформувались кілька основних підходів, що відрізняються способом подання мовного сигналу, характером математичних моделей та рівнем використання апріорних знань про мовлення. Кожен підхід відображає певний етап еволюції уявлень про природу мовлення та можливості його формалізації.

Акустично-фонетичний підхід грунтується на лінгвістичній теорії мовлення і припущенні, що мовний сигнал може описуватись як послідовність фонетичних одиниць (фонем), кожна з яких має характерні акустичні ознаки. Основна ідея цього підходу – явне виділення та моделювання акустичних параметрів, які відповідають артикуляційним особливостям мовлення. У межах цього підходу мовний сигнал аналізується щоб виявити фонетичні ознаки, такі як форматні частоти, спектральна енергія в певних діапазонах та часові характеристики. Далі отримані ознаки зіставляються із заздалегідь визначеними фонетичними правилами. Перевагами цього методу є його інтерпретованість та тісний зв'язок із лінгвістичною теорією мовлення. Проте, на практиці він показав посередні результати, виявившись малостійким до варіативності мовлення, шумів та індивідуальних особливостей дикторів. Крім того, формалізація повного набору фонетичних правил є надзвичайно складною. Зважаючи на це, у класичній інтерпретації акустично-фонетичний підхід майже не використовується в сучасних ASR-системах, хоча його окремі ідеї були інтегровані в пізніші моделі.

Підхід розпізнавання образів розглядає мовлення як послідовність акустичних шаблонів, що можуть бути безпосередньо порівняні між собою без явного лінгвістичного опису. Згідно цього підходу мовний сигнал подається у вигляді набору числових ознак, а задача розпізнавання зводиться до пошуку найбільш схожого шаблону серед еталонів. Одним із класичних методів у межах цього підходу є алгоритм динамічного вирівнювання за часом (Dynamic

Time Warping, DTW), який дозволяє порівнювати дві часові послідовності різної довжини, знаходячи оптимальне вирівнювання між ними. DTW широко використовується у ранніх системах розпізнавання мовних команд, зокрема в умовах малого словника та обмеженої кількості дикторів. Перевагою цього підходу є його концептуальна простота та відсутність складних мовних моделей. Але він має суттєві обмеження, пов'язані з низькою масштабованістю, чутливістю до шумів та необхідністю зберігання великої кількості еталонів. Зі збільшенням словника та варіативності мовлення ефективність таких систем сильно знижується.

Наступний етап розвитку ASR-систем – використання ймовірнісних моделей, зокрема прихованих марківських моделей (Hidden Markov Models, HMM) та моделей гаусових сумішей (Gaussian Mixture Models, GMM). У цьому підході мовлення моделюється як стохастичний процес , у якому досліджувані акустичні ознаки залежать від прихованих станів, що відповідають фонемам або їх частинам. HMM дозволяють ефективно описувати часову структуру мовлення, натомість GMM використовуються для апроксимації розподілу акустичних ознак у кожному стані. Ймовірнісні моделі тривалий час були стандартом у промислових ASR-системах і застосовувались, зокрема, у ранніх версіях систем розпізнавання мовлення від IBM, Microsoft та Nuance. Перевагами HMM та GMM підходів є наявність ефективних алгоритмів навчання та декодування, формальна математична обґрунтованість, можливість інтеграції мовних моделей. Однак ці моделі мають обмежену здатність до моделювання складних нелінійних залежностей та значною мірою залежні від якості вручну розроблених ознак [7].

Сучасний етап розвитку автоматичного розпізнавання мовлення характеризується переходом від ручного проєктування ознак і моделей до підходів, заснованих на глибоких нейронних мережах. У таких системах більшість закономірностей мовлення автоматично визначається з даних у

процесі навчання. Згорткові нейронні мережі ефективно застосовуються для обробки двовимірного представлення мовного сигналу – спектрограм і log-mel спектрів, завдяки їх здатності виявляти локальні часово-частотні патерни. Рекурентні нейронні мережі дозволяють моделювати часові залежності у мовному сигналі. Комбінація цих підходів і подальший розвиток нейронних архітектур дали можливість створювати високоточні системи розпізнавання мовлення, які використовуються у сучасних голосових асистентах, системах командного керування. Прикладами таких систем є сучасні рішення від Google, Apple, Amazon. Домінування нейронних мереж у сучасних ASR-системах зумовлене їх здатністю до узагальнення, адаптації до різних умов та ефективного використання великих навчальних вибірок. Водночас це зумовлює зростання вимог до обчислювальних ресурсів, що стимулює розвиток компактних та оптимізованих архітектур для практичного застосування.

1.6 Порівняльний аналіз аналогів та існуючих рішень

Розглянемо популярні рішення у сфері розпізнавання мовлення, які широко застосовуються на практиці: Google Speech-to-Text, Amazon Transcribe та Apple Siri.

Google Speech-to-Text це хмарний сервіс автоматичного розпізнавання мовлення, що функціонує в рамках платформи Google Cloud. Система орієнтована на розробників та інтеграцію у прикладні програмні рішення. У основі Google Speech-to-Text лежить багаторівнева нейронна архітектура, яка поєднує в собі згорткові нейронні мережі для первинної обробки спектральних ознак, рекурентні мережі або трансформерні блоки для моделювання часових залежностей та мовні моделі великого масштабу для уточнення послідовності слів. Аудіосигнал, перетворений у спектральне представлення обробляється акустичною моделлю, результати поєднуються з мовною моделлю, що

підвищує точність розпізнавання в умовах шуму або неточної вимови. Цей сервіс застосовується для автоматичної транскрипції аудіо- та відеоматеріалів, у голосових інтерфейсах застосунків, в системах автоматичного створення субтитрів, а також для аналітики аудіоконтенту. Безумовними перевагами сервісу Google Speech-to-Text є висока точність розпізнавання при стандартних умовах запису, підтримка великої кількості мов та діалектів і можливість адаптації під конкретні домени. Проте, є і недоліки, такі як залежність від хмарної інфраструктури та доступу до мережі, обмеження конфіденційності даних, неможливість повного контролю над внутрішньою архітектурою моделей.

Amazon Transcribe це хмарний сервіс ASR, який є складовою екосистеми Amazon Web Services (AWS). Система орієнтована на масштабовані серверні рішення та корпоративне використання. Amazon Transcribe використовує глибокі нейронні мережі для акустичного аналізу мовлення, поєднані з мовними моделями, оптимізованими для потокової та пакетної обробки аудіо. Значна увага приділяється обробці довготривалих аудіозаписів та інтеграції з іншими сервісами AWS. Архітектура системи дозволяє автоматично розпізнавати паузи та структуру мовлення, виконувати нормалізацію тексту та працювати з потоковими аудіоданими в реальному часі. Застосовується для аналізу діалогів, транскрипції телефонних розмов а також інтегрується у корпоративні інформаційні системи. Серед переваг сервісу можна виділити високу масштабованість, тісну інтеграцію з сервісами AWS та підтримку потокового розпізнавання. Недоліками Amazon Transcribe є складність налаштування, менша гнучкість у порівнянні з відкритими рішеннями та повна залежність від хмарного середовища.

Apple Siri це персональний голосовий помічник, інтегрований у пристрої Apple. Його основна мета – забезпечення голосової взаємодії користувача з операційною системою Apple та сервісами компанії. У Siri використовується

поєднання хмарних та локальних моделей машинного навчання. Завдяки такому підходу розпізнавання ключових слів та простих команд може виконуватись безпосередньо на пристрої, тоді як складні запити обробляються на серверах Apple. Отже, Siri застосовується для голосового керування пристроями, виконання системних команд та взаємодії з сервісами Apple. Перевагами голосового помічника є простота використання для кінцевого користувача, можливість часткової локальної обробки та тісна інтеграція з апаратним та програмним забезпеченням Apple. Обмежена можливість кастомізації, невелика кількість підтримуваних мов та орієнтація виключно на екосистему Apple є недоліками цього рішення.

У таблиці 1.1 наведено порівняння розглянутих систем за ключовими технічними та функціональними критеріями.

Таблиця 1.1 – Порівняння аналогів

Критерій оцінювання	Google Speech-to-Text	Amazon Transcribe	Apple Siri
Тип системи	Хмарний сервіс ASR	Хмарний сервіс ASR	Вбудований голосовий помічник
Основна архітектура	Глибокі нейромережі (CNN та RNN/Transformer)	Глибокі нейромережі з потоковою обробкою	Комбінована (локальні та хмарні моделі)
Підтримка мов	Дуже широка (більше 100 мов і діалектів)	Широка, але залежить від регіону	Обмежена
Режим роботи	Реальний час та пакетна обробка	Реальний час та пакетна обробка	Переважно реальний час

Продовження таблиці 1.1

Критерій оцінювання	Google Speech-to-Text	Amazon Transcribe	Apple Siri
Можливість адаптації	Обмежена через API	Обмежена через AWS	Відсутня
Масштабованість	Висока	Висока	Низька (орієнтація на кінцевого користувача)
Контроль над моделлю	Відсутній	Відсутній	Відсутній
Залежність від хмари	Повна	Повна	Часткова

Існуючі комерційні рішення орієновані на практичне використання і не забезпечують контролю над архітектурою моделей і процесом навчання. Це обмежує їх застосування у дослідженні де важлива відтворюваність експериментів, контроль над наборами даних та можливість модифікації моделей. Тому в роботі використовується власна реалізація системи розпізнавання мовлення, що дає змогу детально дослідити вплив структури даних, аугментації та вибору архітектури на якість розпізнавання.

1.7 Постановка задачі

На основі проведеного аналізу предметної області та існуючих засобів штучного інтелекту для розпізнавання усного мовлення виявлено, що основною проблемою є складність забезпечення високої точності класифікації при обмежених обчислювальних ресурсах, а також варіативність вхідних даних. Основна маса сучасних нейромереж демонструє високу ефективність

на статичних вибірках, проте їх стабільність при зміні структури датасету є недостатньо вивченою. Враховуючи виявлену проблеми існуючих рішень, для досягнення поставленої мети, що полягає у дослідженні засобів штучного інтелекту та визначенні найкращих, необхідно виконати наступні завдання:

- огляд та аналіз існуючих підходів до задачі розпізнавання мовлення:
    1. проаналізувати методи попередньої обробки звукових сигналів та їх перетворення у спектрально-часове представлення, обрати найбільш прийнятний метод;
    2. визначити сильні та слабкі сторони класичних архітектур при роботі з короткими мовними командами.
- підготовка експериментальної бази:
    1. обрати та підготувати набір даних для проведення навчання та тестування моделей;
    2. здійснити поділ даних на підмножини для навчання, валідації та фінального тестування.
- вибір та адаптація нейромережевих архітектур:
    1. на основі проведеного раніше аналізу архітектур, обрати дві найбільш підходящі для проведення дослідження;
    2. визначити та налаштувати параметри процесу навчання.
- програмна реалізація:
    1. обрати засоби реалізації програмної системи для проведення дослідження, такі як мова програмування, середовище розробки, бібліотеки;
    2. розробити програмне забезпечення для проведення експерименту.
- експериментальне дослідження:
    1. провести навчання обраної моделі на обраних наборах даних;
    2. виконати тестування навчених моделей на тестових вибірках з набору даних.

- оцінка результатів та формування рекомендацій:
    1. за отриманими в результаті тестування показниками точності зробити висновки щодо якості розпізнавання, яку демонструє конкретна модель;
    2. провести порівняльний аналіз та сформувати рекомендації щодо застосування досліджуваних нейромережевих архітекур.

Висновок до розділу 1

У першому розділі проведено теоретичний аналіз задачі автоматичного розпізнавання мовлення, розглянуто основні етапи функціонування систем розпізнавання. Зокрема процеси попередньої обробки аудіоданих, виділення інформативних ознак, побудови акустичних моделей та прийняття рішень на основі результатів класифікації. Проаналізовано особливості сучасних нейромережевих підходів до розпізнавання. Визначено, що згорткові нейронні мережі є ефективними для аналізу спектральних представлень мовлення та виділення локально-частотних закономірностей, а рекурентні – забезпечують моделюваня часової структури сигналу. Поєднання цих підходів у гібридних архітектурах дає змогу враховувати і просторові, і часові характеристики мовлення одночасно.

У рамках аналізу сучасних комерційних рішень визначено, що високі показники якості розпізнавання досягаються завдяки використанню глибоких нейронних мереж та попереднього навчання на великих вибірках. Разом з тим, для наукового дослідження ключовим є не відтворення готових рішень, а експериментальне вивчення впливу архітектури моделі, навчальних даних та методів попередньої обробки на кінцеву якість розпізнавання.

Таким чином, результати аналізу ключових аспектів першого розділу формують теоретичне підґрунтя для подальшого експериментального дослідження. Проведений аналіз дає змогу обґрунтовано обрати нейронні

архітектури для реалізації та порівняння. Отримані теоретичні висновки визначають напрям подальшої роботи та слугують основою для розробки і реалізації програмної системи для розпізнавання мовлення.

2 ВИБІР НЕЙРОННИХ МОДЕЛЕЙ ТА МЕТОДІВ ДЛЯ АВТОМАТИЧНОГО РОЗПІЗНАВАННЯ МОВНИХ КОМАНД

Автоматичне розпізнавання мовлення є однією з основних задач сучасної обробки сигналів та штучного інтелекту. Окреме місце в цій сфері займає задача розпізнавання коротких мовних команд, яка широко застосовується у вбудованих системах, голосових асистентах, системах керування безконтактними інтерфейсами, а також у пристроях із обмеженим обчислювальним ресурсом. У процесі розвитку методів глибинного навчання відбувся перехід від класичних статистичних підходів до нейронних архітектур, які дають можливість моделювати складні часово-частотні залежності мовного сигналу. Не існує універсального рішення для задачі розпізнавання мовних команд. Ефективність системи значною мірою залежить від обраної архітектури нейронної мережі, від характеристик навчальних даних, умов збору даних, способів попередньої оброки.

2.1 Обґрунтування напряму дослідження та постановка експериментальної задачі

У межах кваліфікаційної роботи постає задача експериментального дослідження, спрямованого на порівняльний аналіз ефективності різних нейронних архітектур у контрольованих умовах. Виникає науково обґрунтоване питання: яким чином вибір архітектури нейронної мережі та версії датасету впливає на якість розпізнавання мовних команд та узагальнювальну здатність моделі при зміні умов тестування.

Мета практичної частини роботи полягає у дослідженні впливу архітектури нейромереж та версії навчальних вибірок на якість розпізнавання коротких мовних команд. Для досягнення цієї мети висувається наступна гіпотеза: Якість розпізнавання мовних команд визначається не тільки обсягом навчальних даних, а і структурою нейронної мережі; при цьому моделі, що

поєднують просторову і часову обробку ознак, демонструють кращу узагальнювальну здатність у задачах перехресного тестування. Для перевірки гіпотези проводиться серія експериментів з використанням різних нейромережевих архітектур та різних конфігурацій навчальних і тестових вибірок.

Дослідження має експериментальних характер і ґрунтується на чисельному моделюванні та обчислювальному експерименті. Засобом перевірки гіпотези є програмний застосунок, який дає змогу навчати нейронні мережі на фіксованих вибірках даних та оцінювати їх якість за допомогою стандартних метричних показників. За основний критерій ефективності взято точність класифікації, що визначається за формулою 2.1

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} \mathrm{I}(\hat{y}_i = y_i) \qquad 2.1$$

де N – кількість тестових прикладів, y_i – істинна мітка класу, $\hat{y}_i$ – передбачена моделлю мітка, I – індикаторна функція.

Додатково аналізуються помилки класифікації для окремих слів, стійкість моделі до зміни версії датасету та результати прямого і перехресного тестування.

Для проведення порівняльного аналізу було обрано дві нейромережеві архітектури: згортково-рекурентна (CRNN) власної реалізації та готове рішення MobileNetV2, в основі якої лежить згорткова нейронна мережа (CNN). Вибір архітектур здійснювався на основі аналізу сучасних підходів до розпізнавання мовлення та наукової літератури [8], [9]. Згорткові нейронні мережі ефективні для аналізу спектрограм як двовимірних структур, рекурентні мережі дають змогу моделювати часові залежності мовного сигналу, компактні архітектури, такі як MobileNetV2, орієнтовані на практичне застосування в обмежених апаратних умовах. Отже для роботи було

обрано архітектури які демонструють різні підходи до розпізнавання усного мовлення, що дає змогу здійснити повноцінний порівняльний аналіз. Практична цінність роботи полягає у можливості використання результатів дослідження при розробці ефективної системи розпізнавання мовлення а також для подальших досліджень.

2.2 Огляд датасетів Speech Commands

Датасет Speech Commands це відкритий набір мовних даних, розроблений компанією Google у межах ініціативи з розвитку інструментів машинного навчання. Датасет поширюється через відкриту програмну бібліотеку для машинного навчання TensorFlow. Основна мета створення цього набору даних – забезпечення стандартизованої експериментальної бази для досліджень у галузі розпізнавання мовних команд (keyword spotting). Датасет забезпечує можливість проведення змістовних порівнянь результатів різних моделей. Speech Commands призначений для вирішення задачі багатокласової класифакації аудіосигналів, де кожному мовному фрагменту відповідає одна команда з обмеженого словника. Такий вид задач характерний для систем голосового керування пристроїв, де важлива компактність моделі, швидкість обробки та стійкість до шумів.

Організація даних у Speech Commands ієрархічна та інтуїтивно зрозуміла. Окрема мовна команда (клас) відповідає окремій директорії, що містить WAV-файли з відповідними аудіозаписами. Також у датасеті є директорія background_noise, яка містить записи фонового шуму. Дані з цієї директорії не є окремим класом, але використовуються для аугментації навчальних даних та підвищення стійкості моделі до шуму. Speech Commands містить також два текстові файли validation_list.txt та testing_list.txt, у яких записано назви аудіофайлів різних класів. Файл validation_list.txt використовується для перевірки результатів під час навчання для налаштування гіперпараметрів

класифікатора. Файл testing_list.txt використовується для оцінки результатів вже навченої моделі. Решта аудіофайлів, які не входять до validation_list.txt та testing_list.txt, відповідно використовуються для тренування моделі.

Існує дві версії датасету Speech Commands v0.01 та Speech Commands v0.02. Вони відрізняються за обсягом, перша версія містить приблизно 64727 аудіозаписів, тоді як друга – 105829. Кількість дикторів в першій версії становить 1881, у другій – 2618. Датасети містять різний склад класів, у першій версії їх 30, які спільні для v0.01 та v0.02, у другій 35, тобто додано 5 нових класів, таким чином словник v0.01 є підмножиною словника v0.02. Спільними характеристиками обох датасетів є формат аудіофайлів WAV, тривалість кожного 1 секунда, а частота дискретизації становить 16кГц [10]. У таблиці 2.1 наведено порівняння двох версій Speech Commands за класами та обсягом.

Таблиця 2.1 – Порівняння версій датасету Speech Commands

Назва класу	Кількість аудіозаписів у Speech Commands v0.01	Кількість аудіозаписів у Speech Commands v0.02
backward	-	1664
bed	1713	2014
bird	1731	2064
cat	1733	2031
dog	1746	2128
down	2359	3917
eight	2352	3787
five	2357	4052
follow	-	1579
forward	-	1557
four	2372	3728
go	2372	3880

Продовження таблиці 2.1

Назва класу	Кількість аудіозаписів у Speech Commands v0.01	Кількість аудіозаписів у Speech Commands v0.02
happy	1742	2054
house	1750	2113
learn	-	1575
left	2353	3801
Marvin	1746	2100
nine	2364	3934
no	2375	3941
off	2357	3745
on	2367	3845
one	2370	3890
right	2367	3778
seven	2337	3998
Sheila	1734	2022
six	2369	3860
stop	2380	3872
three	2356	3727
tree	1733	1759
two	2373	3880
up	2375	3723
visual	-	1592
wow	1745	2123
yes	2337	4044
zero	2376	4052

Обидві версії мають 30 спільних класів, що дозволяє проведення експериментів, у яких моделі навчаються на одній версії датасету, тестуються на іншій. Однак наявність додаткових п’яти класів backward, forward, follow, learn, visual призводить до асиметрії між датасетами, що впливає на кількість доступних тестових прикладів. Зважаючи на це, модель, навчена на Speech Commands v0.02 і протестована на цій же версії, використовує повний тестовий набір, тоді як модель, навчена на Speech Commands v0.01 і протестована на Speech Commands v0.02, може бути коректно оцінена тільки на спільній множині класів, що автоматично зменшує кількість тестових прикладів. Наявні відмінності між версіями датасету мають принциповий вплив на експериментальний процес:

- збільшення кількості класів у Speech Commands v0.02 ускладнює задачу класифікації за рахунок розширення простору рішень;
- саме відмінності між датасетами роблять доцільним використання прямого і перехресного тестування для комплексної оцінки ефективності моделей;
- перехресне тестування (навчання на Speech Commands v0.01 – тестування на Speech Commands v0.02) дає змогу оцінити здатність моделі узагальнювати знання, отримані на менш різноманітному наборі даних.

Отже огляд обраного датасету показує, що Speech Commands v0.01 та Speech Commands v0.02, незважаючи на спільну основу, значно відрізняються за складом класів і кількістю аудіоданих. Це створює передумови для глибокого експериментального дослідження узагальнювальних властивостей нейронних моделей та обґрунтовує використання перехресного тестування як інструменту наукового аналізу.

2.3 Обґрунтування вибору нейронні архітектури для розпізнавання мовлення

2.3.1 Згортково-рекурентна нейронна мережа

Згортково-рекурентна нейронна мережа (англ. Convolutional Recurrent Neural Network, CRNN) це гібридна архітектура, що поєднує переваги і згорткових, і рекурентних нейронних мереж для аналізу сигналів, що мають просторово-часову структуру. Вхідний сигнал у задачах розпізнавання мовлення зазвичай подається у вигляді часово-частотного подання, зокрема логарифмічної мел-спектрограми (див. розділ 1.3). така спектрограма є двовимірною матрицею:

$$X \in R^{F \times T} \qquad 2.2$$

де F – кількість частотних смуг, T – кількість часових фреймів.

Архітектура CRNN дає змогу застосовувати рекурентні нейромережі для моделювання часових залежностей між послідовними фреймами та згорткові нейромережі для автоматичного вилічення локальних спектральних ознак. Таким чином вибір такої архітектури є цілком логічним для задачі розпізнавання коротких мовних команд, де важливо враховувати спектральні патерни та їх динаміку в часі.

Згорткова частина архітектури CRNN виконує роль ієрархічного екстрактора ознак. Вона складається з декількох згорткових блоків, кожен з яких зазвичай містить:

- операцію двовимірної згортки:

$$(X \times K)(i,j) = \sum_{m,n} X(i + m, j + n)\, K(m,n) \qquad 2.3$$

де X – вхідна матриця ознак (часово-частотне подання мовного сигналу), K – згорткове ядро, i,j – координати елемента вихідної карти ознак, m,n – індекси елементів ядра згортки.

- нелінійну активаційну функцію (ReLu);
- нормалізацію (Batch Normalization);
- операцію субдискретизації (Max Pooling).

Операція згортки полягає у послідовному накладанні ядра K на вхідну матрицю X з обчисленням зваженої суми елементів у відповідному локальному вікні (формула 2.3). Згортки виконуються одночасно по частотній та часовій осях спектрограми, що дає змогу виявляти локальні кореляції між сусідніми частотами і фреймами. Застосування операцій субдискретизації зменшує розмірність ознак і збільшує стійкість моделі до незначних зсувів у часі та частоті, що характерно для реального мовлення [11]. У результаті роботи згорткових блоків формується багатовимірне представлення

$$Z \epsilon R^{C \times F' \times T'} \qquad 2.4$$

де C – кількість каналів (карт ознак), F', T' - зменшені частотна та часова розмірності.

Після того як вхідний сигнал оброблено згортковими блоками, формується послідовність компактних векторів ознак, впорядкованих у часі. Для моделювання часових залежностей між ними в CRNN використовується рекурентний нейронний блок на основі елементів типу GRU (Gated Recurrent Unit). Особливою ознакою рекурентних нейронних мереж є наявність внутрішнього стану пам’яті. Це дозволяє враховувати інформацію з попередніх часових кроків під час обробки поточного елементі послідовності. Така властивість рекурентних мереж принципово важлива для мовлення, оскільки акустичні характеристики звуку змінюються поступово, а значення

окремого фрейму часто може залежати від сусідніх у часі фрагментів сигналу. GRU містить механізм керування потоком інформації, який реалізується за допомогою спеціальних gate-механізмів. Основними є gate оновлення, що визначає, яка частина попереднього стану пам'яті має бути збережена, та gate скидання, що регулює силу впливу попереднього стану на формування нового. Завдяки такому механізму модель може зменшити вплив застарілої інформації, зберегти важливі ознаки на довших часових інтервалах та уникнути зникання градієнтів [12].

Формально, робота GRU описується системою рівнянь:

$$z_t = \sigma(W_z x_t + U_z h_{t-1}),$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1}),$$
$$\hat{h_t} = \tanh(W_h x_t + U_h (r_t \Theta h_{t-1})),$$
$$h_t = (1 - z_t) \Theta h_{t-1} + z_t \Theta \hat{h_t}. \quad 2.5$$

де x_t – вектор вхідних ознак на часовому кроці t, h_{t-1} – прихований стан мережі на попередньому часовому кроці t, h_t – оновлений прихований стан, W,U – матриці вагових коефіцієнтів, що навчаються, σ – сигмоїдна функція активації, tanh – гіперболічний тангенс, Θ - поелементне множення.

Для розпізнавання мовних команд важливий не лише попередній, але й наступний контекст сигналу. Саме тому використано двонапрямлений рекурентний блок, у якому послідовність обробляється одночасно у прямому та зворотному напрямках часу. Для кожного часового фрейму формується два приховані стани, один – на основі попередній фреймів, другий – на основі наступних фреймів. Отримані стани об'єднуються в єдиний вектор ознак для більш повного врахування контексту мовного сигналу. Це дуже ефективно для коротких мовних команд, де інформація про початок і кінець слова має значення для коректної класифікації.

Після отримання вектору прихованих ознак, його потрібно відобразити у простір класів, що відповідають можливим мовним командам. Цю функцію виконує класифікаційний шар нейронної мережі. У архітектурі CRNN класифікаційний блок реалізується через повнозв'язні шари, що на вхід приймають прихований стан рекурентної мережі. Для двонапрямленого GRU вхідний вектор класифікатора формується завдяки конкатенації прихованих станів прямого та зворотного напрямків, що подвоює його розмірність. Задля зменшення перенавчання у класифакаційному блоці та покращенню узагальнювальної здатності використовується регуляризація шляхом випадкового виключення нейронів. Це дуже актуально для задач з обмеженою кількістю навчальних даних. Класифікаційний шар є заключним у архітектурі CRNN. Він формує фінальне рішення щодо класу вхідного сигналу.

2.3.2 Згорткова нейронна мережа MobileNetV2

MobileNet – це сімейство згорткових нейронних мереж, які були розроблені для використання із обмеженими обчислювальними ресурсами. Основною метою створення таких нейромереж є досягнення компромісу між точністю розпізнавання та обчислювальною складністю моделі. Уперше MobileNet запропонували дослідники з Google, маючи ідею створення моделей, придатних для мобільних пристроїв та вбудованих систем без втрати точності порівняно з класичними глибокими CNN [13].

У процесі розвитку цієї концепції з'явилась архітектура MobileNetV2, у якій усунули недоліки попередньої версії, такі як обмежена здатність до збереження інформації при зменшенні параметрів. Архітектура демонструє високу ефективність представлення ознак при мінімальній обчислювальній складності завдяки трьом ключовим ідеям:

- використання глибинно-роздільних згорток (depthwise separable convolutions) ;

- застосування інверсних резидуальних блоків;
- лінійна проекція виходу згорткових блоків.

Глибинно-роздільна згортка – основний структурний елемент MobileNetV2 та є факторизованою формою стандартної двовимірної згортки. У класичній згортковій операції фільтр одночасно виконує міжканальне та просторове узагальнення, при цьому збільшуючи кількість параметрів. Натомість глибинно-роздільна згортка ці операції розділяє. Depthwise-згортка окремо застосовується до кожного каналу вхідного тензора та відповідає за просторове вилучення ознак, pointwise-згортка з ядром 1 × 1 лінійно комбінує канали. Таким чином значно зменшується кількість операцій множення-додавання і параметрів мережі, а виразна здатність моделі достатньо зберігається.

Головні різниця між попередніми версіями та MobileNetV2 полягає у використанні інверсних резидуальних блоків (inverted residual blocks). У класичних резидуальних з'єднаннях відбувається спочатку зменшення розмірності ознак, а потім її відновлення, MobileNetV2 робить навпаки. Спочатку відбувається розширення простору ознак за допомогою використання точкової згортки, а потім виконується глибинна згортка і завершальний етап – лінійна проекція назад у простір меншої розмірності без застосування нелінійної активації. Це дає змогу мінімізувати втрату інформації, яка може виникати при застосуванні нелінійностей у просторах з малою розмірністю. Лінійна проекція, з математичної точки зору, забезпечує коректне відображення багатовимірних ознак у компактний простір без погіршення їх структури.

MobileNetV2 призначена для обробки зображень, проте її універсальна згорткова структура дає можливість застосувати модель і до спектрограм мовлення. Логарифмічна мел-спектрограма може бути інтерпретована як двовимірне зображення, у якому одна вісь відповідає частоті, а інша – часовим

фреймам. Щоб забезпечити сумісність зі стандартною реалізацією MobileNetV2 спектрограми подаються як багатоканальний тензор, що дозволяє використовувати існуючі архітектурні блоки без значних змін.

Значна перевага MobileNetV2 – можливість використовувати попередньо навчені ваги, отримані на великомасштабних наборах зображень. Підхід перенесення навчання дає можливість розглядати згорткову частину мережі як універсальний екстрактор ознак, який потім адаптується до особливостей мовного сигналу. Навчені ваги пришвидшують збіжність алгоритму навчання та підвищують стабільність результатів.

Компактні архітектури мають низку переваг у задачах розпізнавання коротких мовних команд. Зменшення кількості параметрів знижує ризик перенавчання, що важливо при роботі з наборами даних обмеженого розміру. Зменшення обчислювальної складності дозволяє використання моделі у режимі реального часу. MobileNetV2 показує високу ефективність при роботі з даними фіксованого розміру та у випадках з логарифмічними мел-спектрограмами, завдяки чому є перспективною для використання у системах голосового керування та інтелектуальних інтерфейсах.

2.3.3 Порівняльний аналіз обраних нейронних архітектур

Для проведення дослідження орано дві нейронні архітектури різної концептуальної спрямованості: гібридну згортково-рекурентну нейронну мережу та згорткову мережу MobileNetV2. Їх можна порівнювати за декількома ключовими характеристиками, зокрема складність моделей, кількість параметрів, очікувана якість розпізнавання та придатність для вбудованих систем.

Складність моделей: CRNN поєднує у своїй структурі згорткові шари для вилучення локальних спектральних ознак та рекурентні шари для моделювання часових залежностей. Таким чином ця архітектура є складнішою

з точки зору обчислювальної структури, тому що рекурентні шари обробляють послідовності фреймів та мають внутрішні стани, що зберігаються у часі. MobileNetV2 навпаки, повністю згорткова архітектура, оптимізована для зменшення обчислювальної складності. Ця архітектура не містить рекурентних шарів, аналіз часової структури сигналу здійснюється опосередковано, через згортки по часовій осі спектрограми. Отже, з точки зору архітектурної складності CRNN є більш гнучкою, але також більш ресурсоємною моделлю, тоді як MobileNetV2 може працювати в умовах обмежених ресурсів.

Кількість параметрів важлива для визначення вимог до пам’яті та обчислювальних ресурсів. CRNN має більше параметрів через рекурентні шари, у яких кількість параметрів зростає пропорційно до розміру прихованого стану та кількості напрямків обробки. MobileNetV2, завдяки глибинно-роздільним згорткам, має менше параметрів, що робить її більш перспективною для застосування у середовищах з обмеженими ресурсами, наприклад мобільних системах.

З точки зору очікуваної якості розпізнавання, CRNN краще для задач. Де важливий часовий контекст сигналу. Рекурентний блок забезпечує врахування послідовності спектральних ознак, що особливо актуально для мовлення, де інформація розподілена в часі. MobileNetV2 орієнтована на вилучення просторових патернів зі спектрограми і може показувати високі показники точності у задачах класифікації коротких мовних команд. Проте відсутність явного механізму моделювання часових залежностей може обмежувати її здатність до узагальнення у складніших акустичних умовах.

Однією з цілей дослідження є формування рекомендацій практичного застосування моделей. MobileNetV2 одразу розроблялась для застосування у мобільних та вбудованих системах, що робить її придатною для реалізації на пристроях з обмеженим обчислювальним ресурсом. CRNN потребує значно

більших обсичлювальних ресурсів і є менш придатною для безпосереднього використання у вбідованих системах без додаткової оптимізації.

2.4 Вибір критеріїв для порівняння результатів

Щоб проаналізувати ефективність нейронних архітектур у роботі використовується обмежений, але інформативний набір показників, які обчислюються безпосередньо у процесі тестування моделей та представлені у вигляді класифікаційного звіту. Це дозволить зосередитись на показниках, які мають чітке інтерпретаційне значення у задачі розпізнавання мовлення.

Основний показник якості моделі це точність класифікації, вона відображає частку правильно розпізнаних мовних команд серед усіх тестових прикладів. Точність класифікації дозволяє дати оцінку загальної ефективності моделі та є базовим критерієм порівняння. У рамках дослідження порівняння моделей здійснюється між різними архітектурами (CRNN та MobileNetV2), між різними умовами тестування (пряме та перехресне) і між версіями датасету Speech Commands. При проведенні аналізу результатів увага приділяється не абсолютному значенню точності, а її відносній зміні. Якщо при зміні тестового набору даних точність зменшується, це вважається індикатором зниження здатності моделі до узагальнення.

Загальна оцінка точності не дозволяє зробити висновок про те, наскільки рівномірно модель розпізнає окремі мовні команди, оскільки помилки можуть зосереджуватись в окремих класах. Тому доцільно використовувати класифікаційний звіт з показниками precision, recall та F1-score для кожного класу. Це стандартні метрики у задачах багатокласової класифікації які застосовуються у дослідженнях автоматичного розпізнавання мовлення [14].

У класифікаційному звіті наводиться значення support, що означає кількість тестових прикладів, що належать до певного класу. Це значення

безпосередньо не вказує на якість, але дозволяє інтерпретувати precision, recall та F1-score.

Точність (precision) показує яка частка прикладів, віднесених моделлю до певного класу, дійсно належить цьому класу:

$$\text{Precision} = \frac{TP}{TP+FP} \tag{2.6}$$

де TP – кількість правильних передбачень даного класу, FP – кількість помилкових розпізнавань, коли модель неправильно віднесла інші команди до цього класу.

Високе значення precision свідчить про низьку кількість помилкових розпізнавань.

Повнота (recall) характеризує здатність моделі знаходити всі приклади певного класу в тестовому наборі:

$$\text{Recall} = \frac{TP}{TP+FN} \tag{2.7}$$

де TP – кількість правильних передбачень, FN – кількість пропущених прикладів, коли команда присутня у сигналі, але не розпізнана моделлю.

Низьке значення recall означає, що модель часто не розпізнає команду, навіть якщо вона присутня у вхідному сигналі.

F1-міра є гармонійним середнім між precision та recall і застосовується як узагальнений показник якості розпізнавання класу:

$$\text{F1} = 2 \times \frac{Precision \times Recall}{Precision+Recall} \tag{2.8}$$

Класифікаційний звіт використовується для визначення класів, які розпізнаються стабільно незалежно від датасету, аналізу класів, для яких F1-міра значно падає та для порівняння поведінки моделей при прямому та перехресному тестуванні.

Додатково обчислюється середнє арифметичне значень F1-міри для всіх класів а також зважене усереднення weighted F1, що обчислюється за формулою:

$$F1_{weighted} = \sum_{i=1}^{N} \frac{support_i}{\sum_{j=1}^{N} support_j} \quad 2.9$$

де N – кількість класів, support – кількість тестових прикладів у кожному класі, i – номер класу, $\sum_{j=1}^{N} support_j$ – загальна кількість тестових прикладів у вибірці.

Weighted F1 показує наскільки добре модель працює в середньому, враховуючи реальну кількість прикладів кожного класу.

Оцінка узагальнювальної здатності моделей є непрямою та відбувається шляхом порівняння результатів тестування на тому ж датасеті, на якому модель навчалась та на іншій версії датасету. Під узагальнювальною здатністю мається на увазі здатність моделі зберігати прийнятну якість розпізнавання при зміні статистичних властивостей вхідних даних, у нашому випадку – версій датасету Speech Commands. Перехресне тестування дозволяє змоделювати ситуацію зміни умов застосування системи без додаткового перенавчання. У випадку, якщо спостерігається помірне зниження загальної точності та збереження допустимих значень F1-міри для більшості спільних класів, вважаємо, що узагальнювальна здатність хороша. Якщо зафіксовано різке погіршення точності, це свідчить про надмірну адаптацію моделі до навчальних даних.

2.5 Аугментація даних та підвищення узагальнювальної здатності моделей

Одна з проблем навчання нейронних мереж для розпізнавання мовлення це обмеженість і неоднорідність навчальних даних. Реальні мовні сигнали можуть містити фоновий шум, різну гучність та подібні спотворення. Які можуть бути не представлені повною мірою в навчальному наборі. Для зменшення впливу цих факторів застосовується аугментація даних [15].

Основний вид аугментації в рамках дослідження це додавання фонового шуму до аудіосигналів під час навчання. Таким чином відбувається імітація реальних умов використання системи розпізнавання мовлення. Фоновий шум додається до сигналу з різним співвідношенням сигнал/шум для того щоб модель вчилась виділяти мовні ознаки навіть при наявності акустичного спотворення.

Аугментація застосовується тільки на етапі навчання моделі, тобто кожна ітерація може використовувати різні зашумлені варіанти одного аудіозапису, що фактично збільшує різноманіття навчальних даних без розширення самого датасету. Аугментація не застосовується під час валідації та тестування, оскільки метою цих етапів є оцінка реальної якості моделі на фіксованих відтворюваних даних. Застосування аугментації для тестування призвело б до некоректного порівняння моделей, тому що результати залежали б не тільки від архітектури а і від випадкових змін у вхідних даних.

Висновок до розділу 2

У цьому розділі здійснено обґрунтований вибір напряму дослідження, нейронних архітектур та критеріїв оцінювання результатів, що є основою для подальшого експериментального аналізу засобів штучного інтелекту для розпізнавання усного мовлення. Після детального аналізу предметної області та задачі класифікації коротких мовних команд визначено доцільність

використання логарифмічних мел-спектрограм для часово-частотного представлення мовного сигналу. Це подання дає можливість поєднати часову динаміку мовлення та спектральну інформацію і є сучасним підходом для систем розпізнавання мовлення.

Здійснено вибір датасету Speech Commands для навчання та тестування нейромережевих моделей та описано особливості двох використовуваних версій цього датасету. Визначено, що різна кількість класів та обсяг даних мають безпосередній вплив на умови навчання та тестування моделей, що зумовлює необхідність застосування прямого і перехресного тестування для оцінки якості розпізнавання.

Детально описано вибір нейронних архітектур. Згортково-рекурентна нейронна мережа поєднує можливості згорткових шарів для автоматичного виділення ознак та рекурентних шарів для моделювання часових залежностей мовного сигналу. Згорткова нейронна мережа MobileNetV2 в свою чергу є прикладом компактної та обчислювально ефективної архітектури, що може застосовуватись у середовищах з обмеженими ресурсами.

Обрано критерії порівняння та оцінки нейронних мереж, які будуть використані у подальшому дослідженні. Обґрунтовано застосування метрик загальної точності а також класифікаційного звіту з оцінками precision, recall та F1-score для аналізу ефективності розпізнавання всієї моделі та окремих класів. Аналіз за обраними метриками при різних умовах тестування дозволяє робити висновки щодо стабільності та узагальнювальної здатності досліджуваних архітектур. Отже, за результатами другого розділу сформовано методичну основу для проведення експериментального дослідження.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ РОЗПІЗНАВАННЯ МОВЛЕННЯ

3.1 Вимоги до програмного засобу

Функціональні вимоги визначають які дії повинна виконувати система у межах поставленої задачі дослідження. Програмний засіб повинен забезпечувати:

- зчитування аудіофайлів у форматі .wav з датасету, перевірку їх тривалості та автоматичне доповнення або обрізання файлу при невідповідності стандарту;
- автоматичне формування навчальної, валідаційної та тестової вибірки;
- підтримку аугментації даних під час навчання;
- перетворення амплітудно-часового сигналу у часово-частотну область;
- цикл навчання з автоматичним підрахунком функції втрат та поточного значення точності на кожній епосі та збереження ваг моделі;
- переривання процесу навчання коли параметри моделі перестають покращуватись для запобігання перенавчання;
- пряме та перехресне тестування навчених моделей;
- обчислення та відображення метрик якості;
- графічний інтерфейс користувача, з можливістю вибору моделі та версії датасету для навчання та тренування, режиму виведення результатів тестування;
- обчислення значення точності розпізнавання та показників, за якими оцінюється якість розпізнавання моделі та формування звіту.

Для формалізації функціональних вимог програми використано діаграму прецедентів (рис. 3.1), яка відображає можливості системи та взаємодію користувача з нею.

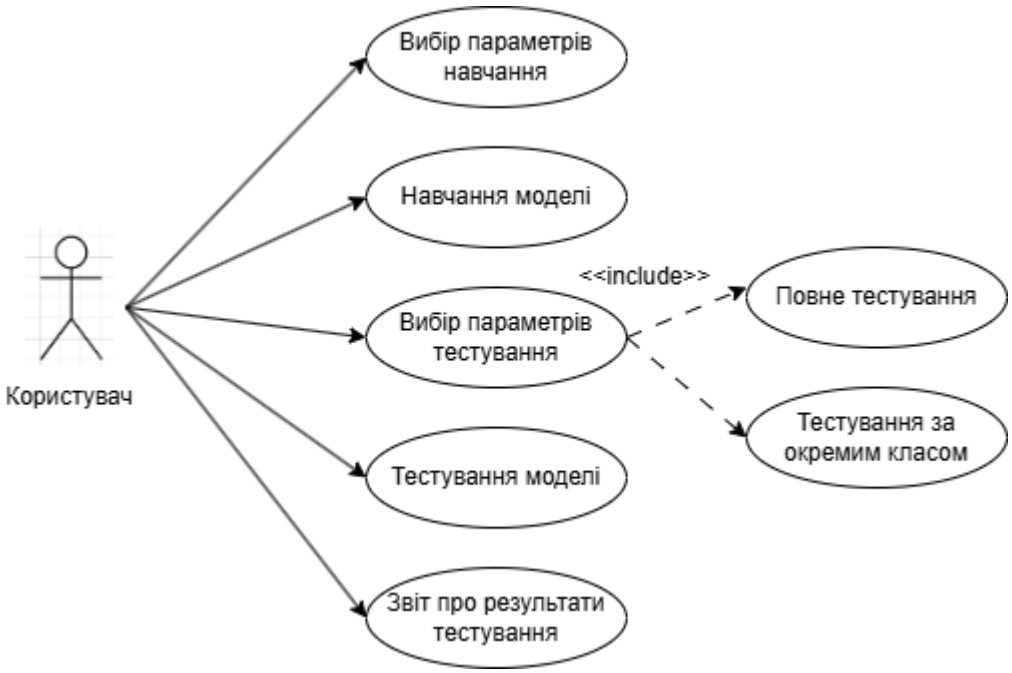

Рисунок 3.1 – Діаграма прецедентів

Робота програми починається з навчання моделей. Користувач обирає параметри навчання, а саме архітектуру моделі та версію датасету, після чого натисканням відповідної кнопки розпочинає навчання обраної моделі. Після завершення навчання проводиться тестування навченої моделі. Для цього користувач має обрати яку саме модель він збирається тестувати на на якій версії датасету. За замовчуванням тестування виконується в повному режимі, але за потреби є можливість змінити режим та виконувати тестування за окремим класом. Після завершення процесу тестування звіт із результатами виводиться на екран.

Нефункціональні вимоги визначають якісні характеристики системи, що не пов’язані з обчислювальною логікою. До нефункціональних вимог належать:

- модульність та розширюваність;

- відтворюваність експериментів. Результати навчання та тестування повинні бути відтворюваними за умови використання однакових даних та параметрів;
- розділення відповідальності між компонентами. Кожен модуль системи має виконувати чітко визначену функцію;
- придатність для наукових досліджень. Система має забезпечувати прозорість експериментів і можливість аналізу результатів.

Вхідні дані – одновимірні цифрові сигнали у форматі .wav тривалістю 1 секунда, з частотою дискретизації 16 кГц.

Вихідні дані – навчені нейронні моделі у вигляді файлів контрольованих точок .pth, значення загальної точності класифікації, класифікаційний звіт з показниками precision, recall, F1-score, текстові виведення процесу навчання та тестування, приклади правильних та помилкових класифікацій.

3.2 Вибір середовища розробки та інструментальних засобів

Розробка програмного забезпечення для виконання дослідження нейронних моделей розпізнавання мовних команд вимагає використання інструментів, які підтримують обробку сигналів, реалізацію та навчання глибоких нейронних мереж та проведення експерименту. Вибір середовища розробки, мови програмування, відповідних програмних бібліотек є важливим кроком при створенні ПЗ та впливає на відтворюваність експериментів, точність реалізації алгоритмів і можливість подальшого розширення системи.

Для реалізації програми було обрано мову програмування Python. По-перше, ця мова програмування є стандартом для задач машинного навчання та обробки мовлення, що підтверджується його використанням у наукових дослідженнях ASR [16]. По-друге, Python має велику кількість спеціалізованих бібліотек для чисельних обчислень, обробки аудіосигналів та побудови нейромереж, це дозволяє приділити більше уваги дослідницькій

частині роботи, і спрощує реалізацію алгоритмів. Python також забезпечує високу читабельність коду, модульність і простоту прототипування, що відіграє важливу роль у контексті експериментальних досліджень, де архітектура моделей та параметри навчання можуть часто змінюватись.

Для розробки та тестування програмного забезпечення обрано інтегроване середовище розробки PyCharm, яке забезпечує:

- зручність при роботі з багатомодульними проектами;
- засоби статичного аналізу коду;
- налагодження;
- інтеграцію з системами керування версіями.

Використання повнофункціонального середовища розробки підвищує надійність реалізації та мінімізує ймовірність помилок у процесі експериментів.

У процесі реалізації системи використовувались фреймворки та бібліотеки з відкритим вихідним кодом:

- PyTorch – фреймворк для проектування та навчання глибоких нейромереж [17]. Динамічна обчислювальна графова модель, яка лежить в його основі, спрощує реалізацію складних архітектур та забезпечує ефективне використання графічного процесору для прискорення процесу навчання;
- Librosa – бібліотека для роботи з аудіосигналами [18]. Вона використовується для завантаження аудіоданих, перетворення сигналів у спектральне подання та обчислення логарифмічних мел-спектрограм. Це стандартна бібліотека у дослідженнях з обробки мовлення та музичних сигналів.
- NumPy – базова бібліотека Python для чисельних обчислень. Застосовується для роботи з багатовимірними масивами та виконання математичних операцій над представленнями ознак;

- Scikit-learn – бібліотека машинного навчання, у межах дослідження використовувалась для обчислення метрик якості класифікації та формування класифікаційного звіту [16].
- Tkinter – стандартна бібліотека Python для створення графічного інтерфейсу користувача.

Система підтримує виконання на центральному та графічному процесорах за наявності відповідного апаратного забезпечення. Таким чином забезпечується масштабованість експериментів і скорочення часу на навчання нейронних моделей.

3.3 Архітектура програмного забезпечення

Архітектура програмного забезпечення системи розпізнавання мовних команд спроектована з урахуванням вимог, описаних у підрозділі 3.1. Обрана архітектура спирається на модульний підхід, де кожен компонент системи реалізує окрему функціональну підсистему та взаємодіє з іншими компонентами через визначені інтерфейси. Це дає змогу ізолювати логіку обробки даних, навчання моделей, тестування та взаємодії з користувачем та спрощує супровід і подальше розширення програмної системи.

ПЗ реалізовано як набір взаємопов’язаних модулів, які логічно згруповані відповідно до етапів експериментального дослідження. Проект включає такі основні підсистеми:

- взаємодія з користувачем за допомогою графічного інтерфейсу;
- підготовка та обробка даних;
- нейронні моделі;
- навчання;
- тестування та оцінка результатів.

Кожна з перерахованих підсистем реалізована як окремий програмний компонент, що забезпечує слабкий зв’язок між частинами системи і можливість їх модифікації незалежно один від одного.

Модульна організація програмного забезпечення розроблена згідно принципів об’єктно-орієнтованого програмування та концепції розділення відповідальності.

Функціональне призначення модулів:

- модуль графічного інтерфейсу (GUI) відповідає за взаємодію програми і користувача. Цей модуль забезпечує можливість введення параметрів експерименту та відображення результатів;
- модуль обробки ознак відповідає за перетворення сирого аудіосигналу в числове представлення, яке підходить для подачі на вхід нейронних мереж;
- модуль роботи з датасетом виконує завантаження аудіофайлів, відповідає за розбиття на тренувальну, валідаційну та тестову вибірки і застосування аугментації даних;
- два модулі нейронних моделей містять реалізації архітектур CRNN та MobileNetV2 відповідно;
- модуль навчання відповідає за оптимізацію параметрів моделі;
- модуль тестування реалізує процедури оцінювання якості розпізнавання та формування класифікаційного звіту.

Завдяки такому розподілу експериментальна логіка чітко відокремлена від інтерфейсу користувача та технічних деталей препроцесингу.

Щоб описати структуру програмного забезпечення та взаємодії його частин використано UML-діаграму компонентів (рис. 3.2). Діаграма відображає основні програмні компоненти та напрямки обміну даними між ними.

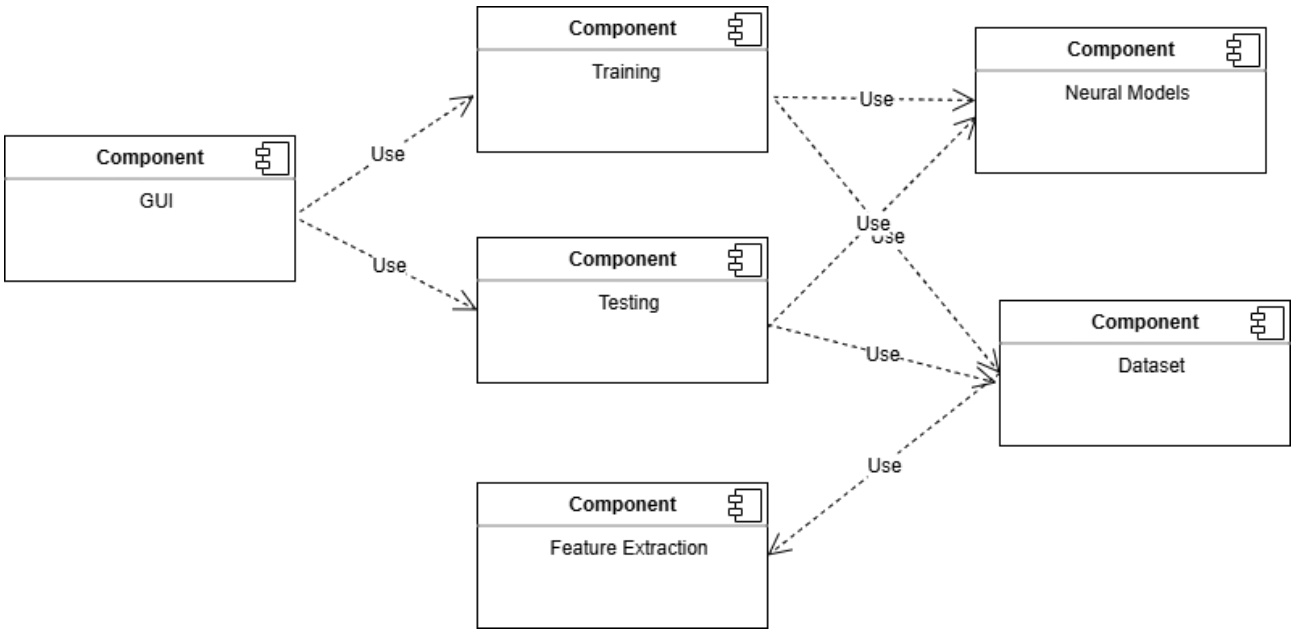

Рисунок 3.2 – Діаграма компонентів

На діаграмі кожен компонент зображений у вигляді прямокутника з назвою модуля. Взаємодія між компонентами показана за допомогою стрілок, що відображають залежність одного компонента від іншого.

Компонент графічного інтерфейсу GUI є вхідною точкою користувача у систему. Цей компонент ініціює процеси навчання та тестування, передаючи обрані користувачем параметри, такі як назва моделі та версія датасету, до компонентів навчання або тестування. Прямого доступу до реалізації моделей або датасету GUI не має, що забезпечує ізоляцію інтерфейсної логіки.

Компонент Dataset для роботи з датасетом призначений для завантаження аудіофайлів, застосування попередньої обробки та формування батчів даних. Якщо виконується процес навчання цей компонент виконує аугментацію даних. Dataset передає підготовані дані до компонентів навчання та тестування.

Компонент обчислення ознак Feature Extraction виконує перетворення аудіосигналу на логарифмічну мел-спектрограму. Компонент Dataset

використовує його як окремий етап обробки незалежно від нейронної архітектури.

Компонент нейронних моделей Models відповідальний за реалізацію архітектур нейронних мереж, що використовуються у системі. У межах цього компоненту реалізовано дві різні архітектури – CRNN та MobileNetV2. Вони розміщені в окремих програмних модулях, але мають спільне функціональне призначення та уніфікований інтерфейс використання.

Компонент Trainer керує процесом оптимізації параметрів нейронної мережі. Він використовує дані, отримані від компонента Dataset та модель з компонента Models та формує навчену модель, яка зберігається для подальшого використання.

Компонент Tester працює з навченою моделлю та тестовою вибіркою і виконує обчислення метрик якості. Отримані результати Tester передає GUI для відображення.

Потік даних у системі відповідає типовому циклу машинного навчання:

- користувач задає параметри експерименту через графічний інтерфейс;
- компонент Dataset завантажує аудіодані та виконує попередню обробку;
- компонент Feature Extraction формує спектральні ознаки;
- дані передаються до компонента Trainer або Tester, в залежності від того яку операцію обрав користувач;
- результати навчання або тестування повертаються до графічного інтерфейсу користувача.

3.4 Класова структура програми

Для наочного представлення структури програмного забезпечення та взаємозв'язків між його елементами використано UML-діаграму класів (рис. 3.3), яка відображає основні класи системи, їх методи та залежності.

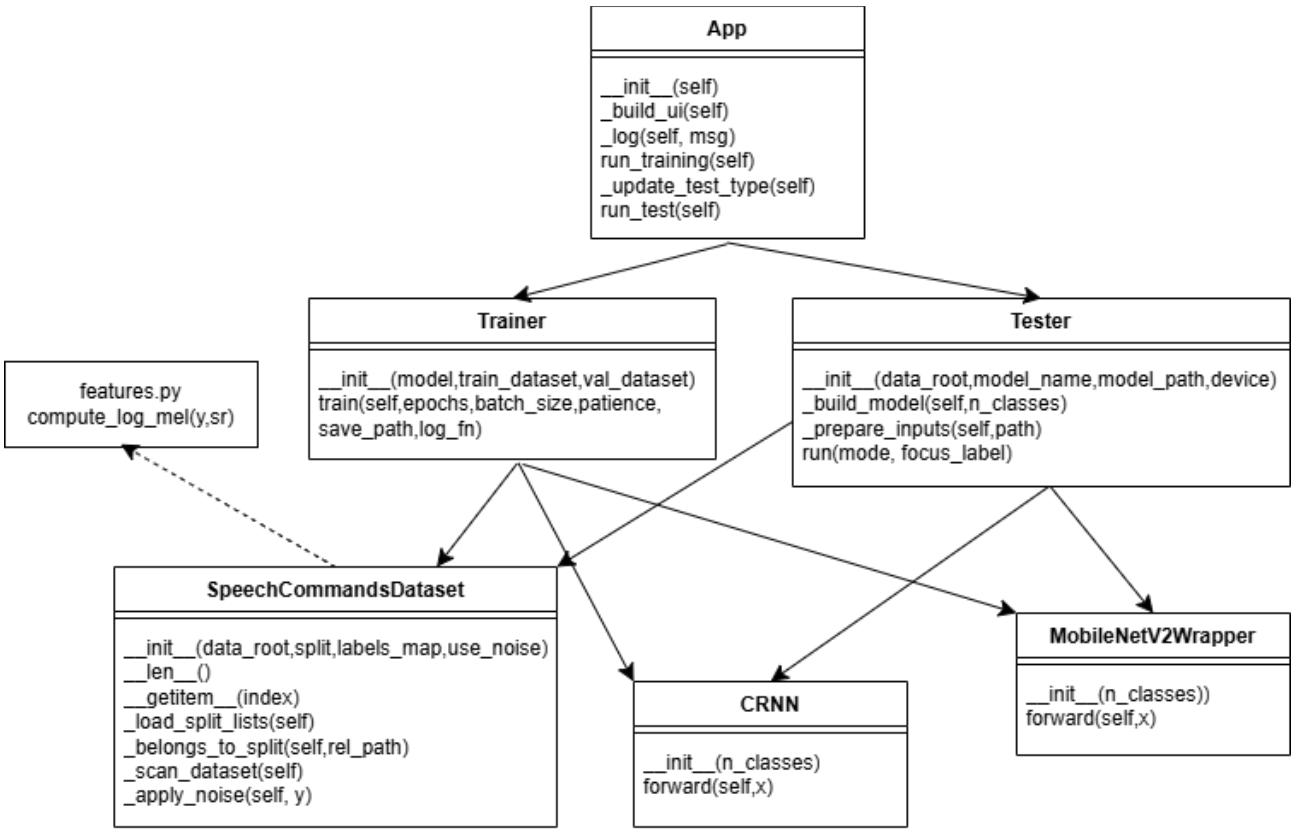

Рисунок 3.3 – Діаграма класів

Клас SpeechCommandsDataset реалізує доступ до аудіоданих датасету SpeechCommands та відповідає за підготовку вхідних даних для процесів навчання та тестування нейронних моделей. Основне призначення класу:

- зчитування аудіофайлів з файлової системи;
- розбиття датасету на тренувальну, валідаційну та тестову вибірки;
- відображення текстових міток класів у числові індекси;
- попередня обробка аудіосигналів та їх приведення до формату, придатного для подальшої роботи.

Основні методи класу:

- __init__(data_root, split, labels_map=None, use_noise=False) ініціалізує датасет, визначає кореневу директорію з даними, тип підвибірки та параметри аугментації;
- __len__() повертає кількість доступних аудіофайлів;
- __getitem__(index) дає доступ до окремого прикладу і його мітки;

Для формування ознак використовується зовнішня функція compute_log_mel. Ця функція виконує обчислення логарифмічної мел-спектрограми. Така залежність показана на діаграмі класів (рис 3.2) у вигляді пунктирної стрілки, що вказує на використання функціонального, а не об’єктного компонента.

Функціональний модуль features.py реалізує процедури попередньої обробки аудіосигналів. На відміну від інших компонентів системи цей модуль не представлений у вигляді класу, тому що не потребує збереження стану. Основна функція модуля compute_log_mel(y, sr, ...) виконує перетворення аудіосигналу в логарифмічну мел-спектрограму фіксованої розмірності.

Клас Trainer відповідає за процес навчання нейронних мереж. Його основна функція це керування циклом навчання, валідації та збереження оптимальних параметрів моделі. Обов’язками класу є:

- ітеративне навчання моделі;
- обчислення значень функції втрат;
- реалізація механізму ранньої зупинки для запобігання перенавчанню;
- збереження навченої моделі разом зі словником класів.

Основні методи класу:

- __init__(model, train_dataset, val_dataset) ініціалізує тренер, приймаючи нейронну модель та датасет;
- train(epochs, batch_size, patience, save_path, log_fn) запускає навчання з заданими параметрами та виконує логування результатів.

Клас не залежить від конкретної архітектури нейромережі та може бути використаний і для CRNN, і для MobileNetV2.

Клас Tester реалізує процедуру оцінювання якості розпізнавання навченої моделі на тестовій вибірці даних. Основна задача класу – забезпечення коректного тестування незалежно від архітектури мережі та версії датасету.

Функціонал класу включає:

- завантаження збереженої навченої моделі;
- встановлення відповідності між індексами та назвами класів;
- виконання передбачень для тестової вибірки;
- формування класифікаційного звіту та обчислення загальної точності розпізнавання.

Основні методи класу:

- __init__(data_root, model_name, model_path, device) ініціалізує тестер та завантажує модель;
- run(mode, focus_label) виконує тестування в обраному користувачем режимі.

Нейронні моделі реалізовані як окремі класи, кожен з яких представляє відповідну архітектуру. Клас CRNN реалізує згортково-рекурентну нейромережу, клас MobileNetV2Wrapper реалізує адаптацію згорткової мережі MobileNetV2 для роботи зі спектрограмими. Спільні методи моделей:

- __init__(n_classes) ініціалізує модель із заданою кількістю класів (заледить від версії датасету);
- forward(x) визначає прямий прохід даних через мережу.

Клас App реалізовує графічний інтерфейс користувача та взаємодію між користувачем і компонентами системи. Функціонал класу:

- вибір архітектури та версії датасету;
- запуск процесів навчання та тестування;
- виведення результатів оцінювання;

- контроль помилок користувача.

3.5 Реалізація графічного інтерфейсу користувача

Графічний інтерфейс користувача було реалізовано за допомогою стандартної бібліотеки Tkinter. Ця бібліотека є базовим інструментом для створення GUI на мові Python. Tkinter було обрано спираючись на такі критерії як кросплатформеність, низькі накладні витрати, широкий функціонал для реалізації інтерфейсу, інтеграція з Python без необхідності використання додаткових фреймворків. Tkinter рекомендований для створення легковагових наукових і навчальних застосунків, де в пріоритеті функціональність і стабільність роботи, а не візуальні ефекти [19].

GUI реалізовано у вигляді одного головного вікна, яке розділене на функціональні блоки відповідно до основних сценаріїв використання системи (рисунок 3.4).

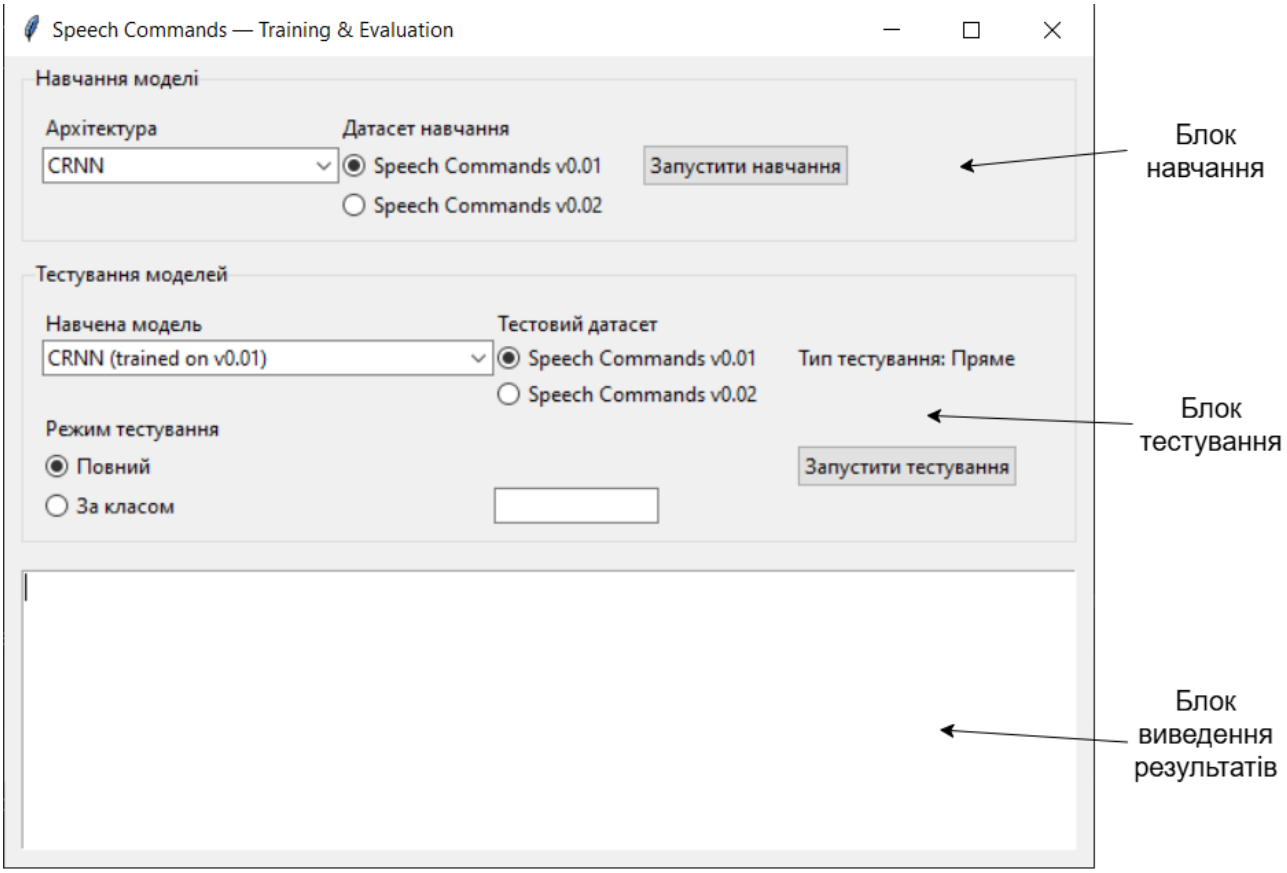

Рисунок 3.4 – функціональні блоки GUI

Блок навчання призначений для керування процесом запуску тренування нейронних мереж. Блок містить такі елементи:

- випадаючий список для вибору архітектури нейромережі, доступні варіанти вибору CRNN та MobileNetV2;
- перемикач вибору версії датасету, доступні версії Speech Commands v0.01 та Speech Commands v0.02;
- кнопка «запустити навчання» ініціює процес навчання обраної моделі на обраній версії датасету.

Під час навчання у блоці виведення результатів (рис. 3.4) відображається інформація про стан процесу (рис. 3.5).

Блок тестування реалізує оцінювання якості попередньо навчених моделей та містить наступні компоненти:

- випадаючий список доступних моделей, серед яких є чотири доступні варіанти, дві моделі треновані на двох різних версіях датасету;
- перемикач вибору версії датасету, доступні версії Speech Commands v0.01 та Speech Commands v0.02;
- вибір режиму тестування, повний або за класом;
- текстове поле для введення назви класу при використанні режиму тестування за класом;
- назва типу тестування яке проводиться, пряме або перехресне, змінюється автоматично;
- кнопка «Запустити навчання» яка запускає процес оцінювання моделі з виведенням результатів.

Текстове поле виведення є центральним елементом інтерфейсу де відображаються процес навчання та результати тестування.

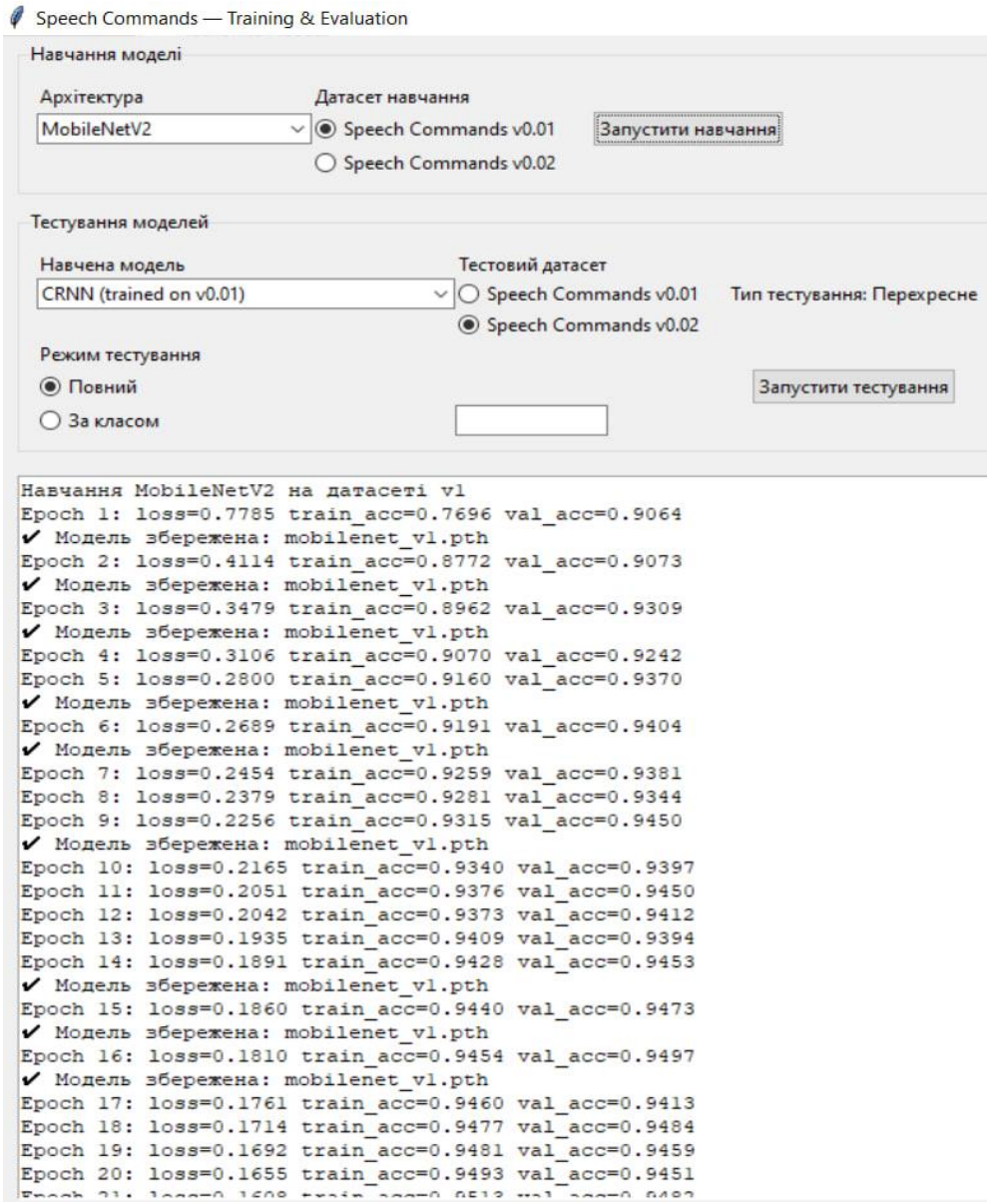

Рисунок 3.5 – вивід інформації про хід тренування

Висновки до розділу 3

У третьому розділі описано проектування та реалізацію програмної системи для проведення експериментального дослідження засобів штучного інтелекту для розпізнавання мовлення. У процесі роботи сформовано модульну архітектуру програмного забезпечення. Таким чином досягнуто чіткого розподілу відповідальності між компонентами. Така організація

спрощує супровід системи та дозволяє незалежно один від одного модифікувати окремі компоненти.

Реалізовано повний цикл роботи з даними, що починається завантаженням та препроцесингом аудіосигналів і закінчується навчанням нейронної мережі. Важливим етапом роботи з даними є розмежування датасету на вибірки для навчання, валідації та тестування а також застосування аугментації даних. Створений інтерфейс користувача забезпечує доступ до основних функцій та дає змогу керувати експериментами буз необхідності безпосереднього втручання у програмний код. Інтерфейс орієнтований на дослідницьке використання.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНЛІЗ РЕЗУЛЬТАТІВ РОЗПІЗНАВАННЯ МОВНИХ КОМАНД

4.1 Умови та організація експериментальних досліджень

Для практичної перевірки ефективності обраних нейронних архітектур у задачі розпізнавання мовних команд проводяться екпериментальні дослідження. Основна мета експериментів – кількісне та якісне порівняння згортково-рекурентної нейромережі CRNN та згорткової нейронної мережі MobileNetV2 за різних умов навчання та тестування. Дослідження мають експериментальний характер та базуються на серії контрольованих обчислювальних експериментів, проведених у єдиному програмному і апаратному середовищі. Таким чином гарантується коректність порівняння та відтворюваність отриманих результатів.

Проведення експериментів виконувалось на персональному комп’ютері з такими характеристиками:

- центральний процесор AMD Ryzen 5 з частотою 2.10 ГГц;
- оперативна пам’ять 8 ГБ;
- графічний процесор NVIDIA GeForce GTX 1050 з підтримкою технології CUDA.

Програмне середовище включає:

- операційна система Microsoft Windows 10;
- інтегроване середовище розробки PyCharm;
- мова програмування Python;
- фреймворк PyTorch для навчання та тестування нейронних мереж;
- бібліотеки NumPy**,** librosa**,** scikit-learn Tkinter.

У межах експерименту досліджуються дві нейромережеві архітектури – згортково-рекурентна CRNN та згорткова MobileNetV2. Вхідними даними для обох моделей є логарифмічні мел-спектрограми, отримані з аудіозаписів

мовних команд. Щоб уніфікувати вхідні дані, усі спектрограми були приведені до одного розміру за кількістю частотних смуг і часових фреймів.

Кероване навчання нейронних мереж відбувалось з використанням тренувальної та валідаційної вибірок датасету Speech Commands. Для навчання моделей застосовувались однакові базові параметри:

- оптимізатор Adam;
- функція втрат – категоріальна крос-ентропія;
- обмеження кількості епох навчання з використанням механізму ранньої зупинки.

Для підвищення стійкості моделей до варіативності реальних умов запису застосовувалась аугментація даних на етапі навчання. Під час валідації та тестування аугментація не використовувалась.

Для оцінки ефективності було використано два основні сценарії тестування – пряме та перехресне. При прямому тестуванні модель навчається і тестується на одній версії датасету Speech Commands. При перехресному тестування навчання здійснюється на одній версії датасету, а тестування на іншій. Такий тип тестування використаний для дослідження здатності моделей до узагальнення та оцінки їх чутливості до змін складу класів. Щоб забезпечити достовірність та відтворюваність результатів експерименти проводились у контрольованому середовищі за фіксованих параметрів моделей з використанням однакового препроцесингу. Отримані результати представлено у вигляді числових показників точності та класифікаційних звітів.

4.2 Проведення експерименту

Експериментальне дослідження починається з етапу навчання розроблених нейромережевих архітектур. Для порівняльного аналізу обрано чотири конфігурації моделей що включають архітектури CRNN та MobileNetV2, які

тренувались на різних версіях датасету Speech Commands. Процес навчання кожної моделі супроводжувався виведенням значень точності та функції втрат (рис. 4.1 – 4.4)

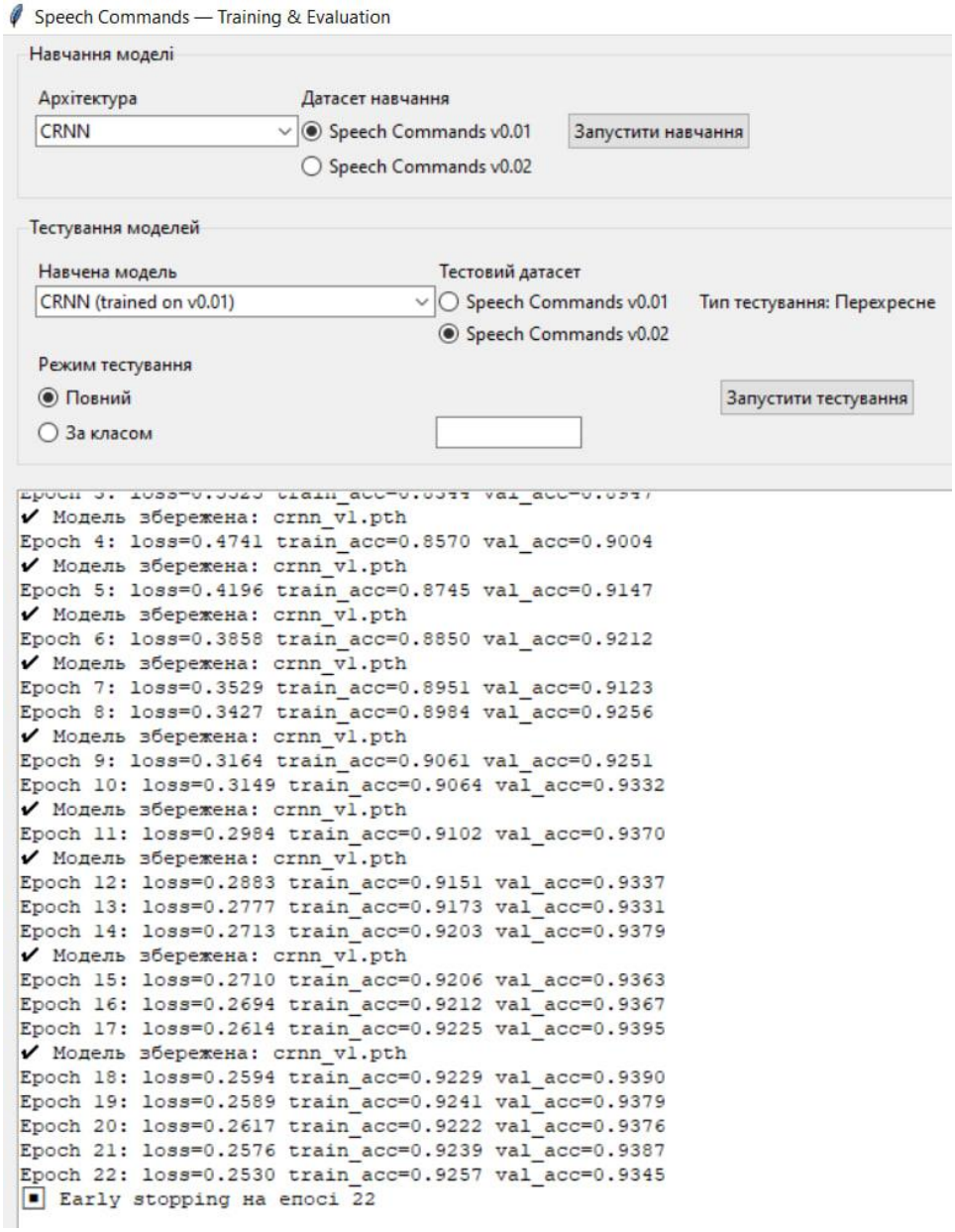

Рисунок 4.1 – результати навчання моделі CRNN на датасеті Speech Commands v0.01

Модель продемонструвала доволі високу точність 93.45% та стабільне збігання на 22-й епосі (рис.4.1). Процес навчання був зупинений функцією early stopping.

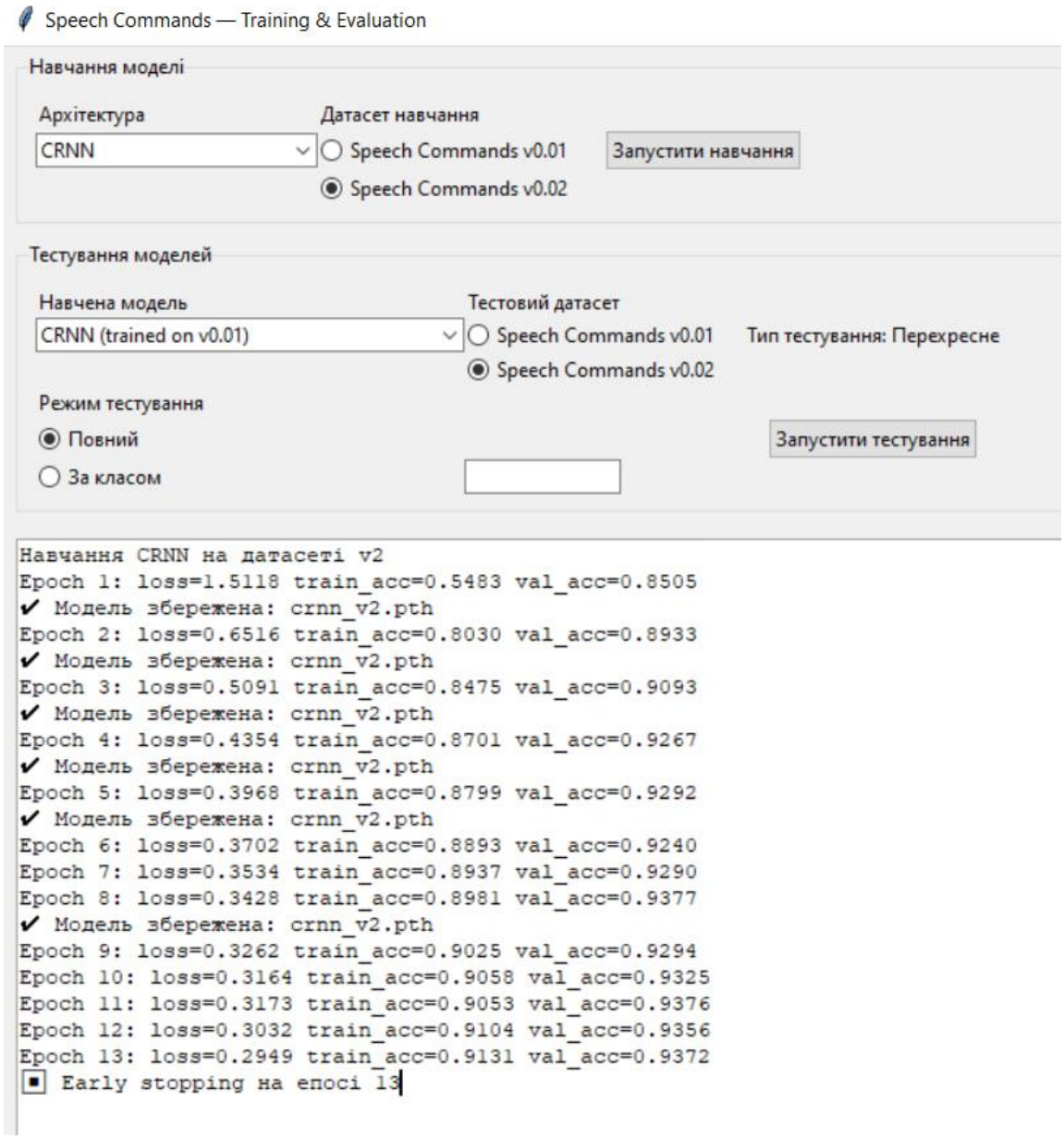

Рисунок 4.2 – результати навчання моделі CRNN Speech Commands v0.02

Тут тренування було зупинено вже на 13-й епосі (рис. 4.2). Показник точності розпізнавання моделі 93.72%, дещо нижчий, порівняно з попередньою моделлю.

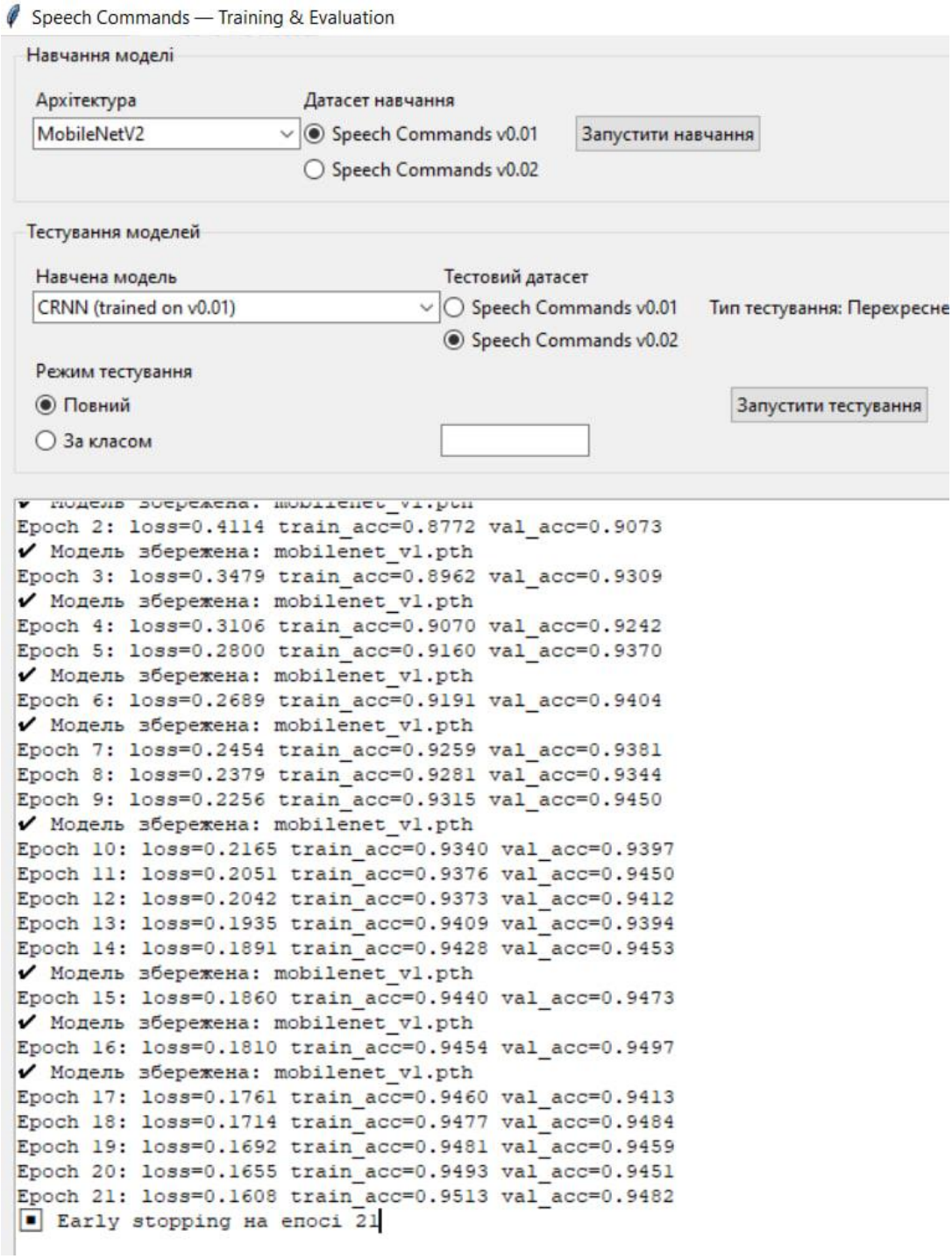

Рисунок 4.3 - результати навчання моделі MobileNetV2 на датасеті Speech Commands v0.01

Модель MobileNetV2 досягла доволі високої точності 94.82% що вище за попередні результати моделі CRNN. Навчання закінчено на 22-й епосі (рис. 4.3). Процес був зупинений функцією early stopping.

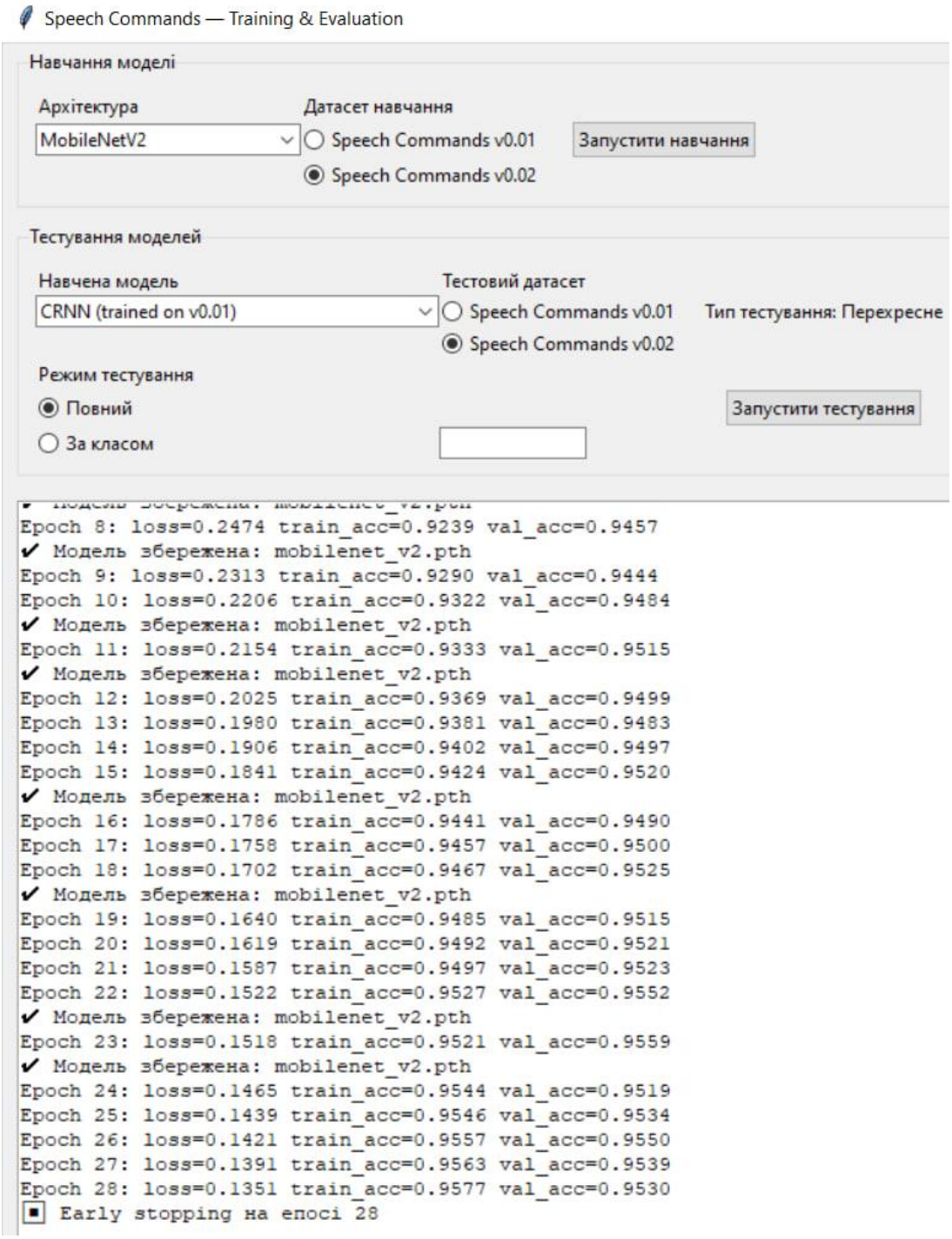

Рисунок 4.4 - результати навчання моделі MobileNetV2 на датасеті Speech Commands v0.02

Остання модель закінчила тренування за 28 епох, при цьому продемонструвавши 95.30% точності (рис. 4.4). Це найвищий показник серед усіх моделей.

4.2.1 Пряме тестування

Після навчання всіх моделей можна переходити до їх прямого та перехресного тестування та порівняння результатів. Починаємо з проведення прямого тестування моделей, тренованих на першій та другій версіях датасету (рис. 4.5 – 4.8).

```
Модель: CRNN (trained on v0.01)
Тестовий датасет: v1
----------------------------------------------------------------
Загальна кількість прикладів: 6835
Кількість помилок: 414
Accuracy: 0.9394
----------------------------------------------------------------

Classification report:

              precision    recall  f1-score   support

         bed       0.90      0.95      0.93       176
        bird       0.98      0.91      0.94       158
         cat       0.95      0.99      0.97       166
         dog       0.97      0.94      0.96       180
        down       0.92      0.90      0.91       253
       eight       0.98      0.94      0.96       257
        five       0.93      0.93      0.93       271
        four       0.95      0.97      0.96       253
          go       0.85      0.87      0.86       251
       happy       1.00      0.98      0.99       180
       house       0.99      0.96      0.97       150
        left       0.94      0.94      0.94       267
      marvin       0.98      0.96      0.97       162
        nine       0.98      0.96      0.97       259
          no       0.90      0.87      0.88       252
         off       0.96      0.92      0.94       262
          on       0.95      0.93      0.94       246
         one       0.96      0.94      0.95       248
       right       0.93      0.92      0.93       259
       seven       0.96      0.98      0.97       239
      sheila       0.99      0.94      0.96       186
         six       0.95      0.97      0.96       244
        stop       0.96      0.98      0.97       249
       three       0.91      0.87      0.89       267
        tree       0.90      0.91      0.90       193
         two       0.87      0.96      0.92       264
          up       0.84      0.98      0.90       272
         wow       0.96      0.96      0.96       165
         yes       0.99      0.95      0.97       256
        zero       0.97      0.93      0.95       250

    accuracy                           0.94      6835
   macro avg       0.94      0.94      0.94      6835
weighted avg       0.94      0.94      0.94      6835
```

Рисунок 4.5 – результати тестування моделі CRNN, тренованої на першій версії датасету, на тестовій вибірці першої версії датасету

```
Модель: MobileNetV2 (trained on v0.01)
Тестовий датасет: v1
------------------------------------------------------------
Загальна кількість прикладів: 6835
Кількість помилок: 313
Accuracy: 0.9542
------------------------------------------------------------

Classification report:

              precision    recall  f1-score   support

         bed       0.96      0.97      0.96       176
        bird       0.98      0.96      0.97       158
         cat       0.97      0.93      0.95       166
         dog       0.99      0.96      0.97       180
        down       0.86      0.96      0.91       253
       eight       0.99      0.96      0.97       257
        five       0.91      0.97      0.94       271
        four       0.99      0.96      0.97       253
          go       0.92      0.94      0.93       251
       happy       0.98      0.99      0.99       180
       house       0.98      0.99      0.98       150
        left       0.98      0.98      0.98       267
      marvin       1.00      0.96      0.98       162
        nine       0.94      0.98      0.96       259
          no       0.98      0.87      0.92       252
         off       0.99      0.93      0.96       262
          on       0.97      0.96      0.96       246
         one       0.97      0.94      0.95       248
       right       0.99      0.93      0.96       259
       seven       0.97      0.98      0.98       239
      sheila       1.00      0.94      0.97       186
         six       0.97      0.98      0.98       244
        stop       0.99      0.98      0.99       249
       three       0.92      0.91      0.91       267
        tree       0.96      0.88      0.92       193
         two       0.94      0.98      0.96       264
          up       0.78      0.98      0.87       272
         wow       0.98      0.97      0.97       165
         yes       1.00      0.95      0.97       256
        zero       0.95      0.97      0.96       250

    accuracy                           0.95      6835
   macro avg       0.96      0.95      0.96      6835
weighted avg       0.96      0.95      0.95      6835
```

Рисунок 4.6 – результати тестування моделі MobileNetV2, тренованої на першій версії датасету, на тестовій вибірці першої версії датасету

```
Модель: CRNN (trained on v0.02)
Тестовий датасет: v2
------------------------------------------------------------
Загальна кількість прикладів: 11005
Кількість помилок: 778
Accuracy: 0.9293
------------------------------------------------------------

Classification report:

              precision    recall  f1-score   support

    backward       0.95      0.94      0.95       165
         bed       0.84      0.86      0.85       207
        bird       0.84      0.93      0.88       185
         cat       0.93      0.89      0.91       194
         dog       0.92      0.87      0.89       220
        down       0.91      0.90      0.91       406
       eight       0.94      0.96      0.95       408
        five       0.91      0.98      0.94       445
      follow       0.90      0.87      0.88       172
     forward       0.88      0.83      0.85       155
        four       0.88      0.92      0.90       400
          go       0.85      0.89      0.87       402
       happy       0.98      0.94      0.96       203
       house       0.98      0.94      0.96       191
       learn       0.87      0.85      0.86       161
        left       0.90      0.96      0.93       412
      marvin       0.95      0.94      0.95       195
        nine       0.95      0.95      0.95       408
          no       0.89      0.95      0.92       405
         off       0.95      0.91      0.93       402
          on       0.94      0.94      0.94       396
         one       0.97      0.89      0.93       399
       right       0.97      0.92      0.94       396
       seven       0.95      0.97      0.96       406
      sheila       0.97      0.95      0.96       212
         six       0.96      0.97      0.96       394
        stop       1.00      0.95      0.97       411
       three       0.93      0.94      0.94       405
        tree       0.95      0.82      0.88       193
         two       0.89      0.97      0.93       424
          up       0.94      0.93      0.94       425
      visual       0.93      0.93      0.93       165
         wow       0.88      0.93      0.91       206
         yes       0.98      0.97      0.98       419
        zero       0.97      0.93      0.95       418

    accuracy                           0.93     11005
   macro avg       0.93      0.92      0.92     11005
weighted avg       0.93      0.93      0.93     11005
```

Рисунок 4.7 - результати тестування моделі CRNN, тренованої на другій версії датасету, на тестовій вибірці другої версії датасету

```
Модель: MobileNetV2 (trained on v0.02)
Тестовий датасет: v2
------------------------------------------------------------
Загальна кількість прикладів: 11005
Кількість помилок: 494
Accuracy: 0.9551
------------------------------------------------------------

Classification report:

              precision    recall  f1-score   support

    backward       0.94      0.99      0.96       165
         bed       0.98      0.88      0.93       207
        bird       0.93      0.96      0.94       185
         cat       0.94      0.96      0.95       194
         dog       0.95      0.95      0.95       220
        down       0.97      0.95      0.96       406
       eight       0.97      0.96      0.97       408
        five       0.98      0.97      0.97       445
      follow       0.92      0.87      0.90       172
     forward       0.92      0.86      0.89       155
        four       0.91      0.96      0.93       400
          go       0.90      0.94      0.92       402
       happy       0.97      0.96      0.97       203
       house       0.96      0.96      0.96       191
       learn       0.93      0.87      0.90       161
        left       0.97      0.98      0.97       412
      marvin       0.98      0.97      0.98       195
        nine       0.99      0.97      0.98       408
          no       0.90      0.98      0.94       405
         off       0.94      0.92      0.93       402
          on       0.97      0.95      0.96       396
         one       0.95      0.95      0.95       399
       right       0.97      0.97      0.97       396
       seven       0.98      0.99      0.98       406
      sheila       0.97      0.98      0.97       212
         six       0.98      0.97      0.98       394
        stop       0.98      0.99      0.98       411
       three       0.94      0.95      0.94       405
        tree       0.94      0.85      0.89       193
         two       0.91      0.98      0.94       424
          up       0.97      0.96      0.96       425
      visual       0.93      0.95      0.94       165
         wow       0.97      0.93      0.95       206
         yes       0.99      0.98      0.99       419
        zero       0.98      0.94      0.96       418

    accuracy                           0.96     11005
   macro avg       0.95      0.95      0.95     11005
weighted avg       0.96      0.96      0.96     11005
```

Рисунок 4.8 - результати тестування моделі MobileNetV2, тренованої на другій версії датасету, на тестовій вибірці другої версії датасету

При проведення прямого тестування моделі навчались та оцінювались на одній і тій самій версії датасету Speech Commands v0.01 що міслить 30 мовних класів. Результати прямого тестування моделей представлені на рисунках 4.5

та 4.6. CRNN продемонструвала точність 93.94%, що свідчить про високу здатність архітектури розпізнавати короткі мовні команди в контрольованому середовищі. Аналіз класифікаційного звіту показав що для більшості класів F1-міра перевищує 93% а для команд happy, house, marvin, seven, stop, yes досягає показників 97-99% що свідчить про стабільне розпізнавання команд із чітко вираженими спектрально-часовими характеристиками. Для деяких класів, зокрема go, up, three, no виявлено нижчий рівень F1-міри (86% - 90%) Це може бути пов’язано з короткою тривалістю цих слів, їх фонетичною схожістю між собою та з іншими командами .

MobileNetV2, навчена та протестована на тій же версії датасету показує вищу загальну точність – 95,42%. Підвищення точності супроводжується зростанням середніх значень метрик у класифікаційному звіті. Для MobileNetV2 характерна більш рівномірна якість розпізнавання між класами. Для більшості команд F1-міра перевищує 95%, а для деяких класів навіть досягає 98% - 99%.

Загалом результати прямого тестування на Speech Commands v0.01 демонструють, що обидві архітектури забезпечують високу якість розпізнавання мовних команд, проте MobileNetV2 показує кращу загальну точність та стабільніші показники для окремих класів у порівнянні з CRNN.

Пряме тестування на датасеті Speech Commands v0.02 є складнішим завданням, що пояснюється більшою кількістю класів, розширеним словником та більшим обсягом тестових даних (11005 прикладів проти 6835 у першій версії). Для CRNN загальна точність становить 92.93%. цей результат дещо нижчий порівняно з показниками першого датасету. Зменшення точності пояснюється більшою кількістю класів та новими командами. Не дивлячись на це CRNN зберігає високі значення F1-міри (93% - 98%) для більшості команд, зокрема stop, seven, yes, nine, six. Більше проблем виникає з класами bed, forward, follow, tree для яких F1-міра знижується до 85% -88%. Це вказує на

обмежені можливості рекурентного компонента мережі при моделюванні складніших спектральних патернів у багатокласовому просторі.

MobileNetV2 на другій версії датасету демонструє найвищий результат 95,51% точності. Класифікаційний звіт підтверджує не лише високу загальну точність і стабільність якості розпізнавання між класами.

Таблиця 4.1 – Таблиця результатів прямого тестування

Модель	CRNN (v0.01)	MobileNetV2 (v0.01)	CRNN (v0.02)	MobileNetV2 (v0.02)
Версія датасету для тестування	v0.01	v0.01	v0.02	v0.02
Кількість класів	30	30	35	35
Кількість тестових прикладів	6835	6835	11005	11005
Кількість правильних розпізнавань	6421	6522	10227	10511
Кількість помилок	414	313	778	494
Точність	93,94%	95,42%	92,93%	95,51%
Середнє precision	94%	96%	93%	95%
Середнє recall	94%	95%	92%	95%
Середнє F1	94%	95%	92%	95%

У таблиці 4.1 наведені середні значення показників precision, recall, F1-міри, точності розпізнавання та кількісні значення правильних і неправильних розпізнавань. Таким чином можна зробити загальний висновок щодо ефективності та якості розпізнавання моделі.

Для детальнішої оцінки роботи моделей проаналізовано результати розпізнавання за окремими класами, результати наведені у таблиці 4.2.

Таблиця 4.2 – Результати тестування за класами

Клас	Модель	Precision	Recall	F1-score	Кількість тестових прикладів
yes	CRNN v0.01	99%	95%	97%	256
yes	MobileNetV2 v0.01	100%	95%	97%	256
no	CRNN v0.01	90%	87%	88%	252
no	MobileNetV2 v0.01	98%	87%	92%	252
go	CRNN v0.01	85%	87%	86%	251
go	MobileNetV2 v0.01	92%	94%	93%	251
up	CRNN v0.01	84%	98%	90%	272
up	MobileNetV2 v0.01	78%	98%	87%	272
three	CRNN v0.01	91%	87%	89%	267
three	MobileNetV2 v0.01	92%	91%	91%	267

Класи no та go є прикладом акустично подібних коротких команд, що ускладнює розпізнавання. За результатами, наведеними у таблиці 4.2 видно, що значення precision та F1-міри знижуються, особливо для моделі CRNN. Це свідчить про хибні спрацювання та плутанину між цими командами. MobileNetV2 демонструє вищі показники для цих класів, отже ця модель є більш ефективною в даному випадку.

Клас yes має стабільно високе значення показників якості при використанні обох архітектур. Модель MobileNetV2 за показником precision показала 100%, що означає відсутність хибних спрацювань.

Клас up поводиться доволі специфічно. Для моделі MobileNetV2 при високому recall демонструє зниження precision.Це свідчить про те, що модель правильно розпізнає більшість прикладів цього класу, але також помилково відносить до нього приклади інших класів. Ця ситуація типова для коротких команд.

Клас three демонструє зниження F1-міри та recall, що може бути спричинено довшою тривалістю вимови слова та більшою варіативністю. Для цього класу модель MobileNetV2 показала кращі результати, порівняно з CRNN, що говорить про її вищу узагальнювальну здатність.

Покласовий аналіз підтвердив результати загального прямого тестування та показав, що архітектура MobileNetV2 забезпечує стабільнішу якість розпізнавання.

4.2.2 Перехресне тестування

Перехресне тестування проводиться щоб оцінити узагальнювальну здатність моделей, тобто їх здатність зберігати якість розпізнавання при зміні розподілу вхідних даних. При перехресному тестуванні модель, що навчалась на одній версії датасету, тестується на іншій. Результати перехресного тестування моделей наведені на рисунках 4.9 – 4.12.

```
Модель: CRNN (trained on v0.01)
Тестовий датасет: v2
------------------------------------------------------------
Загальна кількість прикладів: 10187
Кількість помилок: 806
Accuracy: 0.9209
------------------------------------------------------------

Classification report:
```

	precision	recall	f1-score	support
bed	0.84	0.88	0.86	207
bird	0.97	0.86	0.91	185
cat	0.94	0.90	0.92	194
dog	0.94	0.86	0.90	220
down	0.92	0.88	0.90	406
eight	0.94	0.93	0.94	408
five	0.94	0.91	0.92	445
four	0.93	0.94	0.93	400
go	0.81	0.87	0.84	402
happy	0.99	0.92	0.95	203
house	1.00	0.90	0.94	191
left	0.90	0.95	0.92	412
marvin	0.98	0.93	0.95	195
nine	0.96	0.94	0.95	408
no	0.84	0.89	0.87	405
off	0.90	0.92	0.91	402
on	0.96	0.90	0.93	396
one	0.96	0.92	0.94	399
right	0.93	0.92	0.92	396
seven	0.96	0.97	0.97	406
sheila	0.98	0.92	0.95	212
six	0.94	0.97	0.95	394
stop	0.95	0.99	0.97	411
three	0.94	0.85	0.89	405
tree	0.79	0.89	0.83	193
two	0.85	0.96	0.90	424
up	0.84	0.96	0.90	425
wow	0.92	0.90	0.91	206
yes	0.99	0.95	0.97	419
zero	0.96	0.92	0.94	418
accuracy			0.92	10187
macro avg	0.93	0.92	0.92	10187
weighted avg	0.92	0.92	0.92	10187

Рисунок 4.9 – результати тестування моделі CRNN, тренованої на першій версії датасету, на тестовій вибірці другої версії датасету

```
Модель: MobileNetV2 (trained on v0.01)
Тестовий датасет: v2
------------------------------------------------------------
Загальна кількість прикладів: 10187
Кількість помилок: 595
Accuracy: 0.9416
------------------------------------------------------------

Classification report:

              precision    recall  f1-score   support

         bed       0.91      0.92      0.91       207
        bird       0.97      0.91      0.94       185
         cat       0.97      0.87      0.92       194
         dog       0.95      0.89      0.92       220
        down       0.84      0.96      0.89       406
       eight       0.98      0.95      0.97       408
        five       0.91      0.98      0.94       445
        four       0.98      0.94      0.96       400
          go       0.90      0.91      0.91       402
       happy       0.97      0.97      0.97       203
       house       0.97      0.94      0.96       191
        left       0.92      0.98      0.95       412
      marvin       0.98      0.94      0.96       195
        nine       0.96      0.97      0.97       408
          no       0.94      0.90      0.92       405
         off       0.97      0.92      0.94       402
          on       0.96      0.94      0.95       396
         one       0.95      0.93      0.94       399
       right       0.98      0.95      0.97       396
       seven       0.96      0.97      0.97       406
      sheila       1.00      0.92      0.96       212
         six       0.96      0.98      0.97       394
        stop       0.98      0.97      0.97       411
       three       0.94      0.89      0.92       405
        tree       0.87      0.85      0.86       193
         two       0.93      0.96      0.95       424
          up       0.90      0.94      0.92       425
         wow       0.90      0.93      0.92       206
         yes       0.99      0.95      0.97       419
        zero       0.89      0.97      0.93       418

    accuracy                           0.94     10187
   macro avg       0.94      0.94      0.94     10187
weighted avg       0.94      0.94      0.94     10187
```

Рисунок 4.10 – результати тестування моделі MobilenetV2, тренованої на першій версії датасету, на тестовій вибірці другої версії датасету

```
Модель: CRNN (trained on v0.02)
Тестовий датасет: v1
------------------------------------------------------------
Загальна кількість прикладів: 6835
Кількість помилок: 410
Accuracy: 0.9400
------------------------------------------------------------

Classification report:

              precision    recall  f1-score   support

    backward       0.00      0.00      0.00         0
         bed       0.89      0.92      0.91       176
        bird       0.91      0.94      0.92       158
         cat       0.93      0.97      0.95       166
         dog       0.94      0.93      0.94       180
        down       0.91      0.91      0.91       253
       eight       0.95      0.96      0.96       257
        five       0.93      0.97      0.95       271
      follow       0.00      0.00      0.00         0
     forward       0.00      0.00      0.00         0
        four       0.93      0.91      0.92       253
          go       0.88      0.92      0.90       251
       happy       0.99      0.98      0.98       180
       house       0.99      0.97      0.98       150
       learn       0.00      0.00      0.00         0
        left       0.96      0.96      0.96       267
      marvin       0.97      0.96      0.97       162
        nine       0.94      0.95      0.94       259
          no       0.91      0.92      0.91       252
         off       0.99      0.92      0.95       262
          on       0.96      0.96      0.96       246
         one       1.00      0.89      0.94       248
       right       0.97      0.89      0.93       259
       seven       0.97      0.97      0.97       239
      sheila       0.97      0.95      0.96       186
         six       0.96      0.97      0.97       244
        stop       0.97      0.95      0.96       249
       three       0.86      0.93      0.89       267
        tree       0.96      0.85      0.90       193
         two       0.91      0.97      0.94       264
          up       0.92      0.96      0.94       272
      visual       0.00      0.00      0.00         0
         wow       0.95      0.97      0.96       165
         yes       0.99      0.96      0.98       256
        zero       0.98      0.92      0.95       250

    accuracy                           0.94      6835
   macro avg       0.81      0.81      0.81      6835
weighted avg       0.95      0.94      0.94      6835
```

Рисунок 4.11 – результати тестування моделі CRNN, тренованої на другій версії датасету, на тестовій вибірці першої версії датасету

```
Модель: MobileNetV2 (trained on v0.02)
Тестовий датасет: v1
------------------------------------------------------------
Загальна кількість прикладів: 6835
Кількість помилок: 273
Accuracy: 0.9601
------------------------------------------------------------

Classification report:

              precision    recall  f1-score   support

    backward       0.00      0.00      0.00         0
         bed       0.99      0.93      0.96       176
        bird       0.97      0.96      0.97       158
         cat       0.91      0.99      0.95       166
         dog       0.98      0.98      0.98       180
        down       0.96      0.95      0.96       253
       eight       0.98      0.96      0.97       257
        five       0.98      0.97      0.98       271
      follow       0.00      0.00      0.00         0
     forward       0.00      0.00      0.00         0
        four       0.98      0.95      0.96       253
          go       0.93      0.94      0.93       251
       happy       0.99      0.99      0.99       180
       house       0.96      0.98      0.97       150
       learn       0.00      0.00      0.00         0
        left       0.98      0.97      0.98       267
      marvin       0.98      0.96      0.97       162
        nine       0.98      0.97      0.98       259
          no       0.94      0.95      0.94       252
         off       0.93      0.93      0.93       262
          on       0.98      0.97      0.98       246
         one       0.94      0.94      0.94       248
       right       0.98      0.94      0.96       259
       seven       0.97      0.99      0.98       239
      sheila       0.97      0.98      0.98       186
         six       0.98      0.97      0.98       244
        stop       0.97      0.99      0.98       249
       three       0.91      0.93      0.92       267
        tree       0.96      0.88      0.92       193
         two       0.92      0.98      0.95       264
          up       0.91      0.98      0.94       272
      visual       0.00      0.00      0.00         0
         wow       0.99      0.98      0.98       165
         yes       0.99      0.97      0.98       256
        zero       1.00      0.92      0.96       250

    accuracy                           0.96      6835
   macro avg       0.83      0.82      0.82      6835
weighted avg       0.96      0.96      0.96      6835
```

Рисунок 4.12 – результати тестування моделі MobilenetV2, тренованої на першій версії датасету, на тестовій вибірці другої версії датасету

Загальна точність моделі CRNN при перехресному тестування знизилась з 93,94% до 92,09%, що характерно для перехресного тестування. Для більшості базових команд (yes, stop, seven, six) зберігаються стабільні значення F1-міри. Для коротких та акустично подібних класів, таких як go, no, up показники

precision знижуються, що говорить про збільшення кількості хибних спрацювань. При переході до меншого набору класів модель зберігає хороший рівень якості.

Для моделі MobilenetV2 точність зменшилась з 95,42% до 94,16%. Модель показує стабільно високі значення precision та recall для більшості класів.якість розпізнавання акустично подібних команд краща, на що вказує збереження стабільно високих значень F1-міри. MobilenetV2 тренована на другій версії датасету при тестування на першій показує точність 96,01% та демонструє мінімальні втрати якості порівняно з прямим тестуванням. Саме MobilenetV2 у цьому експерименті оказує найкращу узагальнювальну здатність.

Обидві моделі показують хорошу узагальнювальну здатність на нових даних. MobilenetV2 є більш стійкою до зміни версії датасету, CRNN показує більшу чутливість до акустично схожих класів та зміни розподілу даних.

Таблиця 4.3 – Таблиця результатів перехресного тестування

Модель	CRNN (v0.01)	MobileNetV2 (v0.01)	CRNN (v0.02)	MobileNetV2 (v0.02)
Версія датасету для тестування	v0.02	v0.02	v0.01	v0.01
Кількість класів	30	30	30	30
Кількість тестових прикладів	10187	10187	6835	6835
Кількість правильних розпізнавань	9381	9592	6425	6562

Продовження таблиці 4.3

Кількість помилок	806	595	410	273
Точність	92,09%	94,16%	94%	96,01%
Середнє precision	93%	94%	95%	96%
Середнє recall	92%	94%	94%	96%
Середнє F1	92%	94%	94%	96%

У звітах перехресного тестування (рис. 4.9 – 4.12) частина класів (backward, follow, learn, visual) мають нульову кількість тестових прикладів, оскільки цих класів немає у Speech Commands v0.01.

Результати перехресного тестування (табл. 4.3) показує, що для обох архітектур точність знизилась, порівняно з прямим тестуванням. Це очікувано та пов’язано зі зміною розподілу даних. MobileNetV2 показала меншу втрату якості та більш стабільні значення середніх показників precision, recall, F1-score.

4.2.3 Тестування за класом

Програма має можливість проводити тестування за окремим класом, для більш детальної оцінки якості розпізнавання. Фонетично подібні слова часто викликають помилки та є особливо складними для автоматичного розпізнавання мовлення. Прикладами таких слів є three та tree. Для кожного з цих класів виконано пряме та перехресне тестування моделей CRNN та MobileNetV2. Результати представлені у таблицях 4.4 та 4.5.

Таблиця 4.4 – Порівняння якості розпізнавання класу three

Модель	Навчання	Тестування	Precision	Recall	F1-score	Кількість тестових прикладів
CRNN	v0.01	v0.01	100%	87%	93%	267
CRNN	v0.01	v0.02	100%	85%	92%	405
CRNN	v0.02	v0.01	100%	93%	96%	267
CRNN	v0.02	v0.02	100%	94%	97%	405
MobileNetV2	v0.01	v0.01	100%	91%	95%	267
MobileNetV2	v0.01	v0.02	100%	89%	94%	405
MobileNetV2	v0.02	v0.01	100%	93%	96%	267
MobileNetV2	v0.02	v0.02	100%	95%	97%	405

Для класу three всі моделі демонструють ідеальне значення precision. Це означає відсутність помилкових спрацювань на цей клас. Основна різниця між результатами моделей CRNN та MobileNetV2 помітна за показником recall, який вказує на здатність моделі знаходити всі відповідні приклади. Найнижчі показники recall зафіксовано у моделей, навчених на першій версії датасету та протестованих на другій. Найкращі значення зафіксовано у моделей, що вчились і тестувались на другій версії датасету. Результати тестування за класом tree наведені у таблиці 4.5.

Таблиця 4.5 – Порівняння якості розпізнавання класу tree

Модель	Навчання	Тестування	Precision	Recall	F1-score	Кількість тестових прикладів
CRNN	v0.01	v0.01	100%	91%	95%	193
CRNN	v0.01	v0.02	100%	89%	94%	193
CRNN	v0.02	v0.01	100%	85%	92%	193
CRNN	v0.02	v0.02	100%	82%	90%	193
MobileNetV2	v0.01	v0.01	100%	88%	94%	193
MobileNetV2	v0.01	v0.02	100%	85%	92%	193
MobileNetV2	v0.02	v0.01	100%	88%	94%	193
MobileNetV2	v0.02	v0.02	100%	85%	92%	193

Клас tree виявився складнішим для розпізнавання порівняно з класом three, що помітно через зниження показників. Не дивлячись на стабільно високе значення precision, показник recall знизився, особливо для моделей, навчених на другій версії датасету. Це означає, що команда tree часто плутається з іншими командами, зокрема фонетично подібними (three). Найнижчий результат продемонструвала CRNN, навчена і протестована на другій версії датасету де recall становить 82%, а F1-міра – 90% .

Висновок до розділу 4

У четвертому розділі проведено експериментальне дослідження ефективності розпізнавання мовних команд із використанням двох нейромережевих архітектур CRNN та MobileNetV2. Оцінка проводилась на основі двох версій датасету Speech Commands у режимах прямого та перехресного тестування.

За результатами прямого тестування виявлено, що обидві моделі мають високу точність розпізнавання. За показниками точності, precision, recall, F1-

міри MobileNetV2 показує стабільно вищі результати у порівнянні з CRNN. Найкращий результат зафіксовано для моделі MobileNetV2, навченої на Speech Commands v0.02 – точність 95%.

Результати перехресного тестування дозволяють зробити висновок про те, що моделі, навчені на другій версії датасету краще узагальнюють знання та є більш стійкими до змін у тестових даних. Моделі, що навчались на першій версії датасету, навпаки, частіше показували нижчу точність.

Проведення додаткового аналізу на основі розпізнавання окремих, фонетично подібних класів, показало, що схожі за вимовою слова складніші для розпізнавання моделлю. Навіть при високих значеннях precision, показник recall іноді був доволі низьким. Це говорить про пропуски правильних прикладів. Модель MobileNetV2 у більшості випадків мала кращу якість розпізнавання складник класів, особливо у режимі перехресного тестування.

Таким чином у результаті проведених експериментів встановлено, що вибір архітектури нейронної мережі та навчальної вибірки безпосередньо впливає на якість розпізнавання мовлення. На основі проведеного дослідження можна зробити висновок, що архітектура MobileNetV2 та датасет Speech Commands v0.02 є найкращим вибором для подальшого застосування у розробці систем розпізнавання мовлення.

ВИСНОВОК

У кваліфікаційній роботі проведено дослідження процесів автоматичного розпізнавання усного мовлення на основі методів усного інтелекту. Проаналізовано вплив архітектури нейронної мережа та наборів навчальних даних на показники точності розпізнавання.

Розглянуто сучасні підходи до розпізнавання мовлення та обґрунтовано доцільність використання нейромережевих моделей для задачі розпізнавання коротких мовних команд. На основі проведеного огляду обрано дві архітектури – CRNN та MobileNetV2, які мають різний принцип роботи.

Розроблено програмний засіб для проведення експериментального дослідження. ПЗ реалізує повний цикл препроцесингу та дозволяє виконувати навчання та тестування моделей. Система підтримує пряме, перехресне та покласове тестування.

Проведено експериментальне навчання та тестування моделей CRNN та MobileNetV2 з використанням датасетів Speech Commands v0.01 та Speech Commands v0.02. Встановлено, що використання другої, розширеної версії датасету дозволяє отримати вищі на 2-3% показники точності. Модель MobileNetV2 показує кращі результати розпізнавання порівняно з CRNN. Масксимальна точність моделі MobileNetV2 становить 95%, а середні показники precision, recall та F1-міри вищі за 95%. Результати перехресного тестування підтвердили, що моделі, навчені на більшому та різноманітнішому наборі даних мають кращу узагальнювальну здатність.

За отриманими результатами можна зробити висновок, що при розробці систем розпізнавання мовлення доцільніше використовувати модель MobileNetV2 та набір даних Speech Commands v0.02, особливо коли треба отримати стабільну якість розпізнавання за змінних умов.

Наукова новизна роботи полягає в уточненні та конкретизації результатів щодо ефективності нейромережевих засобів розпізнавання усного мовлення. Цей процес відбувався шляхом експериментального порівняння засобів штучного інтелекту. Отримані результати розширюють уявлення про вплив архітектури моделей та характеристик навчальних даних на показники точності та стабільності розпізнавання.

Практичне значення роботи полягає у можливості використання отриманих результатів для подальшої розробки систем розпізнавання мовлення. Результати можуть використовуватись розробниками програмного забезпечення як довідкові дані. Також результати роботи будуть корисні при вивченні дисциплін, пов’язаних з машинним навчанням та обробкою мовлення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пазика, К. С. Дослідження засобів штучного інтелекту для розпізнавання усного мовлення [Текст] / О. С. Куроп'ятник, К. С. Пазика // Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості та освіті : тези XIX міжнародна науково-практична конференція / Український державний університет науки і технологій. – Дніпро, 2025.
2. Білинський, Й. Й. Електронні системи [Текст] : навч. посіб. / Й. Й. Білинський, К. В. Огороднік, М. Й. Юкиш. – Вінниця : ВНТУ, 2011. – 208 с.
3. Ашихмін, А. В. Підвищення точності та швидкості обчислення миттєвого спектра гармонійних сигналів за допомогою детектора основного тону [Електронний ресурс] / А. В. Ашихмін. – 2008. – Режим доступу: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/fdc3443f-1ae0-4c85-a5ab-4f2f5844a9b7/content
4. Fourier Transform: History, Mathematics, and Applications - SOUL OF MATHEMATICS. Soul of mathematics. URL: https://soulofmathematics.com/index.php/fouriertransform-history-mathematics-and-applications/
5. Pyrovolakis, K. Multi-Modal Song Mood Detection with Deep Learning [Текст] / K. Pyrovolakis, P. Tzouveli, G. Stamou // Sensors. – 2022. – Vol. 22, Iss. 3. – Art. 1065. – Режим доступу: https://doi.org/10.3390/s22031065
6. Миколюк, І. О. Аналіз методів розпізнавання мовлення [Електронний ресурс] / І. О. Миколюк, О. І. Суприган. – Вінниця : ВНТУ, [рік]. – Режим доступу: file:///C:/Users/Admin/Downloads/5144-18819-1-PB.pdf
7. Мрозек, Є. Р. Аналіз сучасних підходів до розв'язання задач розпізнавання мовлення [Текст] / Є. Р. Мрозек // Інтелектуальні інформаційні технології та системи. – 2024. – № 4. – С. 39–49. – Режим доступу: https://doi.org/10.15407/csc.2024.04.039
8. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Electronic resource] / A. G. Howard, M. Zhu, B. Chen [et al.]. – 2017. – Mode of access: https://arxiv.org/abs/1704.04861
9. Розгон, Ю. В. Класифікація об'єктів за допомогою цифрових сигналів з використанням методів машинного навчання [Електронний ресурс] : кваліфікаційна робота бакалавра : спец. 122 "Комп'ютерні науки" / Ю. В. Розгон ; наук. кер. А. І. Дворниченко ; Сумський державний університет. – Суми, 2025. – Режим доступу: https://essuir.sumdu.edu.ua/server/api/core/bitstreams/5cf1a0f0-90d7-437f-9645-2875be876af3/content.

10. Warden, P. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition [Electronic resource] / Pete Warden // Google Brain. – 2018. – 12 April. – 11 p. – Access Mode : URL : https://arxiv.org/pdf/1804.03209
11. Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection [Electronic resource] / E. Cakir, G. Parascandolo, T. Heittola [et al.] // Tampere University of Technology. – 2017. – 10 p. – Access Mode : URL : https://arxiv.org/pdf/1702.06286.
12. Chung, J. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling [Electronic resource] / J. Chung, C. Gulcehre, K. Cho, Y. Bengio // Universite de Montreal. – 2014. – 9 p. – Access Mode : URL : https://arxiv.org/abs/1412.3555
13. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications [Electronic resource] / A. G. Howard, M. Zhu, B. Chen [et al.]. – 2017. – 9 p. – Access Mode : URL : https://arxiv.org/abs/1704.04861
14. Tang, R. Deep Residual Learning for Small-footprint Keyword Spotting [Electronic resource] / R. Tang, J. Lin // University of Waterloo. – 2018. – 5 p. – Access Mode : URL : https://arxiv.org/abs/1710.10341
15. Абросімов Є. О. Дослідження сучасних методів аугментації текстових даних [Електронний ресурс] / Є. О. Абросімов ; наук. кер. А. О. Дейнеко // Сучасні проблеми управління та автоматики в технічних системах : тези доп. конф. — Харків : ХНУРЕ, 2024. — Режим доступу: URL : https://openarchive.nure.ua/server/api/core/bitstreams/f25fa901-081e-461d-b5e5-dc5b55c6c705/content
16. Scikit-learn: Machine Learning in Python [Electronic resource] / F. Pedregosa, G. Varoquaux, A. Gramfort [et al.]. – 2018. – 6 p. – Access Mode : URL : https://arxiv.org/pdf/1201.0490
17. PyTorch: An Imperative Style, High-Performance Deep Learning Library [Electronic resource] / A. Paszke, S. Gross, F. Massa [et al.]. – 2019. – 12 p. – Access Mode : URL : https://arxiv.org/abs/1912.01703
18. McFee, B. librosa: Audio and Music Signal Analysis in Python [Electronic resource] / B. McFee, C. Raffel, D. Liang [et al.] // Proceedings of the 14th Python in Science Conference. – 2015. – P. 18–25. – Access Mode : URL : https://proceedings.scipy.org/articles/Majora-7b98e3ed-003.pdf
19. Tkinter — Python interface to Tcl/Tk [Electronic resource] / Python Software Foundation. – 2026. – Access Mode : URL : https://docs.python.org/3/library/tkinter.html

ДОДАТОК А
Технічне завдання

ЗАТВЕРДЖУЮ
Перший проректор Українського
державного
університету науки і технологій
Анатолій РАДКЕВИЧ

АНАЛІЗАТОР ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ РОЗПІЗНАВАННЯ УСНОГО МОВЛЕННЯ

Технічне завдання

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01530– 01 – ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олена КУРОП’ЯТНИК
Виконавець
_____Катерина ПАЗИКА
Нормоконтролер
_____Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО

44165850.01530 – 01 – ЛЗ

АНАЛІЗАТОР ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ РОЗПІЗНАВАННЯ УСНОГО МОВЛЕННЯ

Технічне завдання

Листів 14

2025

АНОТАЦІЯ

Документ 44165850.01530 – 01 – ЛЗ «Аналізатор засобів штучного інтелекту для розпізнавання усного мовлення» входить до складу програмної документації на програму, що реалізує додаток для проведення дослідження засобів штучного інтелекту для розпізнавання усного мовлення.

У даному документі представлене технічне завдання. Програма написана на мові Python. Об’єм пам’яті, що займає програма складає 6 ГБ. Конфігурація пристрою стандартна. Програма призначена для систем під керуванням операційної системи Windows.

Зміст

ВСТУП

Розпізнавання усного мовлення є однією з найбільш актуальних і швидкозростаючих галузей сучасності. Штучний інтелект і методи машинного навчання відкривають нові можливості для автоматизації процесу перетворення голосу на текст, що знаходить застосування в різних сферах: голосові асистенти, системи транскрипції, медичні додатки, автоматизація обслуговування клієнтів та системи безпеки.

Метою кваліфікаційної роботи є дослідження сучасних засобів штучного інтелекту для автоматичного розпізнавання усного мовлення, аналіз їх технічних характеристик та вибір оптимальних рішень для подальшого використання. Розроблена програма сприятиме досягненню цієї мети.

1. ПІДСТАВА ДЛЯ РОЗРОБКИ

Основою для розробки є наказ ректора Українського державного університету науки і технології Сухий К.М. «Про затвердження тем та призначення керівників дипломних проектів» №1401ст від 02.10.2025 року.

Тема проекту: “Аналізатор засобів штучного інтелекту для розпізнавання усного мовлення”.

Керівник дипломного проекту: Куроп’ятник Олена Сергіївна.

2. ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональним призначенням розробки є порівняння показників точності роботи та узагальнювальної здатності різних моделей розпізнавання мовлення, включаючи готові рішення та розроблені самостійно, з метою виявлення переваг та недоліків кожного підходу.

Експлуатаційним призначенням розробки є виявлення слабких місць і обмежень існуючих моделей з метою можливого подальшого вдосконалення або оптимізації алгоритмів;

3. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до функціональних характеристик

Програма повинна надавати можливість:

- підтримувати роботу з аудіофайлами у форматі wav;
- виконувати тренування та валідацію CRNN;
- виконувати тренування та валідацію MobileNetV2;
- обирати модель для дослідження;
- обирати версію датасету для навчання та тренування;
- виконувати розпізнавання слів у тестовому наборі даних;
- за результатами тестування визначати точність нейромережі;
- відображення результатів тестування в різних режимах: повний вивід, виведення результатів розпізнавання обраного класу.

Вхідними даними є:

Аудіофайли у форматі wav, які згруповані за класами, де кожен клас відповідає слову яке розпізнається, вибір моделі, вибір режиму відображення результатів тестування.

Вихідні дані: звіт за результатами тестування, що включає в себе значення точності, повноти розпізнавання, F1-міри, кількості тестових прикладів та список з перших 10 помилкових та правильних розпізнавань.

3.2 Вимоги до надійності

Вимоги до надійності наступні:

- відсутність збоїв при роботі програми;
- кількість помилок не повинна перевищувати однієї на 10000 операцій;
- повідомлення користувача про непередбачувані ситуації.

3.3 Вимоги експлуатації

Для роботи ПЗ необхідно

- температура повітря у приміщенні – 21-25 С, відносна вологість 40-60%;
- мінімальний досвід роботи із ПК.

3.4 Вимоги до складу та параметрів технічних засобів

Апаратне забезпечення необхідне для роботи:

- процесор з тактовою частотою 2 ГГЦ або вище;
- оперативна пам’ять 8 ГБ або більше;
- вільне місце на диску не менше 10 ГБ для розміщення програмного продукту, датасету та експериментальних даних.

3.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення необхідне для роботи:

- ОС Windows 10/11;
- Python 3.11.4;
- бібліотеки PyTorch, TensorFlow, torchaudio, Librosa, NumPy, Tkinter.

3.6 Вимоги до маркування і упаковки

Упаковка програмного продукту, включаючи документацію повинна бути захищена від пошкоджень різного роду (механічних, кліматичних).

Програма повинна мати етикетку, на якій вказано:

- назва програми;
- версія програми;
- ім'я розробника;
- контактна інформація розробника.

Приклад маркування :

3.7 Вимоги до транспортування та зберігання

Транспортування повинно проводитись в упаковці та забезпечувати цілісність продукту.

4. ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

- технічне завдання;
- пояснювальна записка;
- текст програми;
- керівництво користувача. Керівництво з дослідження засобів штучного інтелекту для розпізнавання усного мовлення.

Програмна документація повинна відповідати вимогам ДСТУ [1].

5. СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиця А.1 – Стадії та етапи розробки

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	10.10.2025 – 20.10.2025	
2	Огляд літератури та аналіз аналогів	20.10.2025 - 25.10.2025	
3	Розробка структур вхідних і вихідних даних, вимог до системи	26.10.2025 – 09.11.2025	
5	Узгодження та затвердження ТЗ, постановка задачі	10.11.2025 – 16.11.2025	30 %
6	Розробка та програмування логіки програми	17.11.2025 – 25.11.2025	
7	Розробка і реалізація інтерфейсу користувача	26.22.2025 – 07.12.2025	
8	Відлагодження програми	08.12.025 – 14.12.2025	60%
9	Розробка, узгодження та затвердження програмної документації	15.12.2025 – 04.01.2026	
10	Розробка демонстраційних матеріалів	05.01.2025 – 11.01.2025	100%
11	Подання кваліфікаційної роботи до кафедри	17.01.2026	
12	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	21.01.2026	

6. ПОРЯДОК І КОНТРОЛЬ ПРИЙМАННЯ

Контроль за виконанням роботи здійснює керівник дипломного проекту. Приймання здійснюється екзаменаційною комісією.

БІБІЛОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем: методичні вказівки до дипломного проектування та лабораторних робіт/уклад.: Ю.М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

ДОДАТОК Б
Текст програми

ЗАТВЕРДЖУЮ
Перший проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

АНАЛІЗАТОР ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ РОЗПІЗНАВАННЯ УСНОГО МОВЛЕННЯ

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.0150 – 01 12 01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олена КУРОП’ЯТНИК
Виконавець
_____Катерина ПАЗИКА
Нормоконтролер
_____Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО

44165850.01530 – 01 12 01

АНАЛІЗАТОР ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ РОЗПІЗНАВАННЯ УСНОГО МОВЛЕННЯ

Текст програми

Листів 21

2025

АНОТАЦІЯ

Документ 44165850.01530 – 01 12 01 «Аналізатор засобів штучного інтелекту для розпізнавання усного мовлення» входить до складу програмної документації на програму, що реалізує додаток для проведення дослідження засобів штучного інтелекту для розпізнавання усного мовлення.

Зміст

1 ОПИС ФАЙЛІВ

Розроблене програмне забезпечення складається з взаємопов’язаних модулів, кожен з яких відповідає за конкретний етап обробки даних, навчання або візуалізації.

- main.py – стартовий модуль системи. Здійснює запуск головного циклу обробки подій програми;
- gui.py – модуль графічного інтерфейсу користувача. Реалізований на базі бібліотекти Tkinter, керує візуальними компонентами;
- dataset_sc.py – модуль керування даними. Реалізує логіку сканування папок, розділення вибірок для тренування, валідації, тестування та механізм аугментації даних;
- crnn.py – модуль архітектури нейромережі CRNN;
- mobilenet.py – модуль архітектури нейромережі MobileNetv2;
- features.py – модуль цифрової обробки сигналів. Використовується для обчислення логарифмічної мел-спектрограми, нормалізації сигналу з використанням бібліотеки librosa;
- train.py – модуль логіки навчання. Реалізує цикл тренування та механізм ранньої зупинки для запобігання перенавчанню;
- tester.py – модуль тестування. Реалізує тестування моделей та генерує класифікаційний звіт.

2 ТЕКСТ ПРОГРАМИ

2.1 Текст програми файлу main.py

```
from gui import App

if __name__ == "__main__":
    app = App()
    app.mainloop()
```

2.2 Текст програми файлу dataset_sc.py

```
import os
import random
import torch
import numpy as np
import soundfile as sf
from torch.utils.data import Dataset
from features import compute_log_mel

IGNORE_FILES = {
    ".DS_Store", "LICENSE", "README.md",
    "testing_list.txt", "validation_list.txt"
}

class SpeechCommandsDataset(Dataset):
    def __init__(
        self,
        root_dir,
        split="train",              # train | val | test
        n_mels=128,
        n_frames=128,
        use_noise=True,             # шум застосовується ТІЛЬКИ якщо True
        noise_prob=0.3,
        labels_map=None,            # для перехресного тестування
        verbose=False
    ):
        self.root_dir = root_dir
        self.split = split
        self.n_mels = n_mels
        self.n_frames = n_frames
        self.use_noise = use_noise and split == "train"
        self.noise_prob = noise_prob
        self.verbose = verbose

        self.samples = []
        self.labels_map = labels_map if labels_map is not None else {}
        self.fixed_labels = labels_map is not None
        self.noise_samples = []

        self._load_split_lists()
        self._scan_dataset()
```

```
        self.idx_to_label = {v: k for k, v in self.labels_map.items()}

        if self.verbose:
            print(
                f"[Dataset:{split}] "
                f"samples={len(self.samples)}, "
                f"classes={len(self.labels_map)}, "
                f"noise={'on' if self.use_noise else 'off'}"
            )

    def _load_split_lists(self):
        def load_list(fname):
            path = os.path.join(self.root_dir, fname)
            if not os.path.exists(path):
                return set()
            with open(path) as f:
                return set(line.strip() for line in f)

        self.val_list = load_list("validation_list.txt")
        self.test_list = load_list("testing_list.txt")

    def _belongs_to_split(self, rel_path):
        if self.split == "train":
            return rel_path not in self.val_list and rel_path not in self.test_list
        if self.split == "val":
            return rel_path in self.val_list
        if self.split == "test":
            return rel_path in self.test_list
        raise ValueError("Unknown split")

    def _scan_dataset(self):
        label_id = len(self.labels_map)

        for entry in sorted(os.listdir(self.root_dir)):
            if entry in IGNORE_FILES:
                continue

            class_dir = os.path.join(self.root_dir, entry)
            if not os.path.isdir(class_dir):
                continue

            if entry == "_background_noise_" and self.use_noise:
                for fname in os.listdir(class_dir):
                    if fname.endswith(".wav"):
                        y, sr = sf.read(os.path.join(class_dir, fname))
                        self.noise_samples.append((y, sr))
                continue

            if self.fixed_labels:
                if entry not in self.labels_map:
                    continue
```

```
            else:
                if entry not in self.labels_map:
                    self.labels_map[entry] = label_id
                    label_id += 1

                class_idx = self.labels_map[entry]

                for fname in os.listdir(class_dir):
                    if not fname.endswith(".wav"):
                        continue

                    rel_path = f"{entry}/{fname}"
                    if not self._belongs_to_split(rel_path):
                        continue

                    full_path = os.path.join(class_dir, fname)
                    self.samples.append((full_path, class_idx))

    def _apply_noise(self, y):
        if not self.noise_samples or random.random() > self.noise_prob:
            return y

        noise, _ = random.choice(self.noise_samples)
        if len(noise) < len(y):
            noise = np.tile(noise, int(np.ceil(len(y) / len(noise))))
        noise = noise[:len(y)]

        alpha = random.uniform(0.1, 0.4)
        return np.clip(y + alpha * noise, -1.0, 1.0)

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        path, label = self.samples[idx]
        y, sr = sf.read(path)

        if self.use_noise:
            y = self._apply_noise(y)

        mel = compute_log_mel(
            y, sr,
            n_mels=self.n_mels,
            n_frames=self.n_frames
        )

        x = torch.tensor(mel, dtype=torch.float32).unsqueeze(0)
        return x, label
```

2.3 Текст програми файлу features.py

```
import numpy as np
import librosa
```

```
def compute_log_mel(
    y,
    sr,
    n_mels=128,
    n_fft=1024,
    hop_length=256,
    n_frames=128
):
    if y is None or len(y) == 0:
        raise ValueError("Empty audio signal")

    # Якщо аудіо багатоканальне — переводимо в mono
    if y.ndim > 1:
        y = librosa.to_mono(y.T)

    # Mel-спектрограма (нелінійна частотна шкала)
    S = librosa.feature.melspectrogram(
        y=y,
        sr=sr,
        n_fft=n_fft,
        hop_length=hop_length,
        n_mels=n_mels,
        power=2.0
    )

    # Логарифмічна шкала
    S_db = librosa.power_to_db(S, ref=np.max)

    # Приведення до фіксованої довжини
    t = S_db.shape[1]
    if t < n_frames:
        pad = n_frames - t
        S_db = np.pad(
            S_db,
            ((0, 0), (0, pad)),
            mode="constant",
            constant_values=(S_db.min(),)
        )
    elif t > n_frames:
        start = (t - n_frames) // 2
        S_db = S_db[:, start:start + n_frames]

    # Нормалізація
    mean = S_db.mean()
    std = S_db.std()
    if std < 1e-6:
        std = 1.0

    S_norm = (S_db - mean) / std

    return S_norm.astype(np.float32)
```

2.4 Текст програми файлу gui.py

```
import tkinter as tk
from tkinter import ttk, messagebox
import threading

from tester import Tester
from train import Trainer
from dataset_sc import SpeechCommandsDataset
from crnn import CRNN
from mobilenet import MobileNetV2Wrapper

DATASET_V1 = r"C:\main\JetBrains\mag\speech_commands_v0.01"
DATASET_V2 = r"C:\main\JetBrains\mag\speech_commands_v0.02"


MODEL_REGISTRY = {
    "CRNN (trained on v0.01)": {
        "model": "CRNN",
        "train_ds": "v1",
        "path": "crnn_v1.pth"
    },
    "CRNN (trained on v0.02)": {
        "model": "CRNN",
        "train_ds": "v2",
        "path": "crnn_v2.pth"
    },
    "MobileNetV2 (trained on v0.01)": {
        "model": "MobileNetV2",
        "train_ds": "v1",
        "path": "mobilenet_v1.pth"
    },
    "MobileNetV2 (trained on v0.02)": {
        "model": "MobileNetV2",
        "train_ds": "v2",
        "path": "mobilenet_v2.pth"
    },
}


class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Speech Commands — Training & Evaluation")
        self.geometry("1150x800")
        self._build_ui()

    def _build_ui(self):
        train_frame = ttk.LabelFrame(self, text="Навчання моделі", padding=10)
        train_frame.pack(fill="x", padx=10, pady=5)

        self.train_model_var = tk.StringVar(value="CRNN")
```

```
self.train_ds_var = tk.StringVar(value="v2")

ttk.Label(train_frame, text="Архітектура").grid(row=0, column=0, sticky="w")
ttk.Combobox(
    train_frame,
    textvariable=self.train_model_var,
    values=["CRNN", "MobileNetV2"],
    state="normal",
    width=25
).grid(row=1, column=0)

ttk.Label(train_frame, text="Датасет навчання").grid(row=0, column=1, sticky="w")
ttk.Radiobutton(train_frame, text="Speech Commands v0.01",
                variable=self.train_ds_var, value="v1").grid(row=1, column=1, sticky="w")
ttk.Radiobutton(train_frame, text="Speech Commands v0.02",
                variable=self.train_ds_var, value="v2").grid(row=2, column=1, sticky="w")

ttk.Button(
    train_frame,
    text="Запустити навчання",
    command=lambda: threading.Thread(target=self.run_training, daemon=True).start()
).grid(row=1, column=2, padx=20)

test_frame = ttk.LabelFrame(self, text="Тестування моделей", padding=10)
test_frame.pack(fill="x", padx=10, pady=5)

self.test_model_var = tk.StringVar(value=list(MODEL_REGISTRY.keys())[0])
self.test_ds_var = tk.StringVar(value="v2")

ttk.Label(test_frame, text="Навчена модель").grid(row=0, column=0, sticky="w")
ttk.Combobox(
    test_frame,
    textvariable=self.test_model_var,
    values=list(MODEL_REGISTRY.keys()),
    state="readonly",
    width=40
).grid(row=1, column=0, sticky="w")

ttk.Label(test_frame, text="Тестовий датасет").grid(row=0, column=1, sticky="w")
ttk.Radiobutton(test_frame, text="Speech Commands v0.01",
                variable=self.test_ds_var, value="v1",
                command=self._update_test_type).grid(row=1, column=1, sticky="w")
ttk.Radiobutton(test_frame, text="Speech Commands v0.02",
                variable=self.test_ds_var, value="v2",
                command=self._update_test_type).grid(row=2, column=1, sticky="w")

self.test_type_label = ttk.Label(test_frame, text="Тип тестування: —")
self.test_type_label.grid(row=1, column=2, padx=15)

self.mode_var = tk.StringVar(value="all")
ttk.Label(test_frame, text="Режим тестування").grid(row=3, column=0, sticky="w")
ttk.Radiobutton(test_frame, text="Повний",
```

```
            variable=self.mode_var, value="all").grid(row=4, column=0, sticky="w")
        ttk.Radiobutton(test_frame, text="За класом",
            variable=self.mode_var, value="class").grid(row=5, column=0, sticky="w")

        self.class_entry = ttk.Entry(test_frame, width=15)
        self.class_entry.grid(row=5, column=1, sticky="w")

        ttk.Button(
            test_frame,
            text="Запустити тестування",
            command=lambda: threading.Thread(target=self.run_test, daemon=True).start()
        ).grid(row=4, column=2, padx=20)

        self.log = tk.Text(self)
        self.log.pack(expand=True, fill="both", padx=10, pady=10)

        self._update_test_type()

    def _log(self, msg):
        self.log.insert(tk.END, msg + "\n")
        self.log.see(tk.END)

    def run_training(self):
        self.log.delete("1.0", tk.END)

        model_name = self.train_model_var.get()
        ds = self.train_ds_var.get()

        data_root = DATASET_V1 if ds == "v1" else DATASET_V2

        train_ds = SpeechCommandsDataset(data_root, split="train", use_noise=True)

        val_ds = SpeechCommandsDataset(
            data_root,
            split="val",
            use_noise=False,
            labels_map=train_ds.labels_map
        )

        n_classes = len(train_ds.labels_map)

        if model_name == "CRNN":
            model = CRNN(n_classes)
            save_path = f"crnn_{ds}.pth"
        else:
            model = MobileNetV2Wrapper(n_classes)
            save_path = f"mobilenet_{ds}.pth"

        self._log(f"Навчання {model_name} на датасеті {ds}")

        trainer = Trainer(model, train_ds, val_ds)
        trainer.train(
```

```
        epochs=30,
        batch_size=32,
        patience=5,
        save_path=save_path,
        log_fn=self._log
    )

def _update_test_type(self):
    key = self.test_model_var.get()
    train_ds = MODEL_REGISTRY[key]["train_ds"]
    test_ds = self.test_ds_var.get()

    if train_ds == test_ds:
        t = "Пряме"
    else:
        t = "Перехресне"

    self.test_type_label.config(text=f"Тип тестування: {t}")

def run_test(self):
    self.log.delete("1.0", tk.END)

    key = self.test_model_var.get()
    cfg = MODEL_REGISTRY[key]

    model_name = cfg["model"]
    model_path = cfg["path"]
    train_ds = cfg["train_ds"]
    test_ds = self.test_ds_var.get()

    data_root = DATASET_V1 if test_ds == "v1" else DATASET_V2

    self._log(f"Модель: {key}")
    self._log(f"Тестовий датасет: {test_ds}")
    self._log("-" * 60)

    tester = Tester(
        data_root=data_root,
        model_name=model_name,
        model_path=model_path
    )

    mode = self.mode_var.get()
    focus = self.class_entry.get() if mode == "class" else None

    mode = self.mode_var.get()
    focus = self.class_entry.get().strip() if mode == "class" else None

    if mode == "class":
        if not focus:
            messagebox.showerror(
                "Помилка",
```

```
                "Будь ласка, введіть назву класу для режиму «За класом»."
            )
            return

        # класи, які знає модель
        valid_labels = tester.labels

        if focus not in valid_labels:
            messagebox.showerror(
                "Помилка",
                f"Клас '{focus}' не існує для цієї моделі.\n"
                f"Доступні класи:\n{', '.join(valid_labels)}"
            )
            return

    results = tester.run(mode=mode, focus_label=focus)
    tester.print_results(results)

    self._log(f"Загальна кількість прикладів: {results['total']}")
    self._log(f"Кількість помилок: {results['errors']}")
    self._log(f"Accuracy: {results['accuracy']:.4f}")
    self._log("-" * 60)

    self._log("\nClassification report:\n")
    self._log(results["report_text"])

    self._log("\nCorrect examples:")
    for p, lbl, conf in results["correct"]:
        self._log(f"[OK] {p} | {lbl} | {conf:.3f}")

    self._log("\nMistakes:")
    for p, t, pr, conf in results["mistakes"]:
        self._log(f"[ERR] {p} | true={t} pred={pr} | {conf:.3f}")
```

2.5 Текст програми файлу crnn.py

```
import torch
import torch.nn as nn

class CRNN(nn.Module):
    def __init__(self, n_classes, n_mels=128, n_frames=128, gru_hidden=256, gru_layers=2,
bidirectional=True, dropout=0.3):
        super().__init__()

        # CNN екстрактор (з batchnorm)
        self.cnn = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1), nn.BatchNorm2d(32), nn.ReLU(),
            nn.MaxPool2d((2,2)),  # n_mels/2, n_frames/2

            nn.Conv2d(32, 64, kernel_size=3, padding=1), nn.BatchNorm2d(64), nn.ReLU(),
            nn.MaxPool2d((2,2)),  # n_mels/4, n_frames/4

            nn.Conv2d(64, 128, kernel_size=3, padding=1), nn.BatchNorm2d(128), nn.ReLU(),
```

```
            nn.MaxPool2d((2,2)),  # n_mels/8, n_frames/8
        )

        freq_p = n_mels // 8
        time_p = n_frames // 8
        self.freq_p = freq_p
        self.time_p = time_p

        self.gru_input_size = 128 * freq_p
        self.gru_hidden = gru_hidden
        self.bidirectional = bidirectional

        self.gru = nn.GRU(input_size=self.gru_input_size,
                          hidden_size=self.gru_hidden,
                          num_layers=gru_layers,
                          batch_first=True,
                          bidirectional=bidirectional,
                          dropout=0.2 if gru_layers > 1 else 0.0)

        # класифікатор
        linear_in = self.gru_hidden * (2 if bidirectional else 1)
        self.classifier = nn.Sequential(
            nn.Linear(linear_in, 256),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(256, n_classes)
        )

    def forward(self, x):
        B = x.size(0)
        x = self.cnn(x)  # -> (B, 128, freq_p, time_p)
        # переставимо у (B, time_p, channels*freq_p)
        x = x.permute(0, 3, 1, 2).contiguous()  # (B, time_p, C, freq_p)
        B, T, C, Fp = x.shape
        x = x.view(B, T, C * Fp)  # (B, time_p, feat)
        out, _ = self.gru(x)      # (B, time_p, hidden*dirs)
        out = out[:, -1, :]       # беремо останній timestep
        logits = self.classifier(out)
        return logits
```

2.6 Текст програми файлу mobilenet.py

```
import torch
from torch import nn
from torchvision.models import mobilenet_v2


class MobileNetV2Wrapper(nn.Module):
    def __init__(self, n_classes):
        super().__init__()

        # Попередньо навчена модель на ImageNet
        self.base = mobilenet_v2(weights="IMAGENET1K_V1")
```

```
        # Заміна класифікатора під кількість класів мовлення
        self.base.classifier[1] = nn.Linear(
            self.base.classifier[1].in_features,
            n_classes
        )

    def forward(self, x):
        # Перетворення 1-канального спектрограмного зображення у 3 канали
        if x.shape[1] == 1:
            x = x.repeat(1, 3, 1, 1)

        return self.base(x)
```

2.7 Текст програми файлу tester.py

```
import torch
import soundfile as sf
from sklearn.metrics import classification_report, confusion_matrix

from dataset_sc import SpeechCommandsDataset
from features import compute_log_mel


class Tester:
    def __init__(self, data_root, model_name, model_path, device="cuda"):
        self.data_root = data_root
        self.model_name = model_name
        self.device = torch.device(device if torch.cuda.is_available() else "cpu")

        ckpt = torch.load(model_path, map_location=self.device)

        if "state_dict" not in ckpt or "labels_map" not in ckpt:
            raise RuntimeError(
                "Checkpoint has invalid format. Expected keys: state_dict, labels_map"
            )

        self.model_labels = ckpt["labels_map"]  # label -> idx
        self.idx_to_label = {v: k for k, v in self.model_labels.items()}
        self.labels = [self.idx_to_label[i] for i in range(len(self.idx_to_label))]

        self.model = self._build_model(n_classes=len(self.labels))
        self.model.load_state_dict(ckpt["state_dict"])
        self.model.to(self.device)
        self.model.eval()

        self.test_ds = SpeechCommandsDataset(
            root_dir=data_root,
            split="test",
            labels_map=self.model_labels
        )

    def _build_model(self, n_classes):
```

```
    if self.model_name == "CRNN":
        from crnn import CRNN
        return CRNN(n_classes)
    elif self.model_name == "MobileNetV2":
        from mobilenet import MobileNetV2Wrapper
        return MobileNetV2Wrapper(n_classes)
    else:
        raise ValueError(f"Unknown model: {self.model_name}")

def _prepare_input(self, path):
    y, sr = sf.read(path)
    mel = compute_log_mel(y, sr, n_mels=128, n_frames=128)
    x = torch.tensor(mel, dtype=torch.float32).unsqueeze(0).unsqueeze(0)
    return x.to(self.device)

def run(self, mode="all", focus_label=None, limit=10):
    y_true, y_pred = [], []
    correct, mistakes = [], []

    if len(self.test_ds) == 0:
        raise RuntimeError("Test dataset is empty")

    # --- перевірка класу ---
    if mode == "class":
        if focus_label not in self.model_labels:
            raise ValueError(
                f"Клас '{focus_label}' не існує. Доступні класи:\n"
                + ", ".join(self.labels)
            )
        focus_idx = self.model_labels[focus_label]

    for path, true_idx in self.test_ds.samples:
        if mode == "class" and true_idx != focus_idx:
            continue

        x = self._prepare_input(path)

        with torch.no_grad():
            logits = self.model(x)
            probs = torch.softmax(logits, dim=1)
            pred_idx = int(torch.argmax(probs, dim=1).item())
            conf = float(probs[0, pred_idx].item())

        y_true.append(true_idx)
        y_pred.append(pred_idx)

        if mode == "class":
            # true label завжди focus_label
            if pred_idx == true_idx and len(correct) < limit:
                correct.append((path, focus_label, conf))
            elif pred_idx != true_idx and len(mistakes) < limit:
                mistakes.append(
```

```
                (path, focus_label, self.idx_to_label[pred_idx], conf)
            )
        else:
            if pred_idx == true_idx and len(correct) < limit:
                correct.append((path, self.idx_to_label[true_idx], conf))
            elif pred_idx != true_idx and len(mistakes) < limit:
                mistakes.append(
                    (path,
                     self.idx_to_label[true_idx],
                     self.idx_to_label[pred_idx],
                     conf)
                )

    total = len(y_true)
    errors = sum(t != p for t, p in zip(y_true, y_pred))
    accuracy = (total - errors) / total if total > 0 else 0.0

    if mode == "class":
        report_dict = classification_report(
            y_true,
            y_pred,
            labels=[focus_idx],
            target_names=[focus_label],
            output_dict=True,
            zero_division=0
        )

        cls = report_dict[focus_label]
        report_text = (
            f"Class: {focus_label}\n"
            f"Precision: {cls['precision']:.2f}\n"
            f"Recall:    {cls['recall']:.2f}\n"
            f"F1-score:  {cls['f1-score']:.2f}\n"
            f"Support:   {int(cls['support'])}"
        )

        cm = confusion_matrix(y_true, y_pred, labels=[focus_idx])

    else:
        report_text = classification_report(
            y_true,
            y_pred,
            target_names=self.labels,
            zero_division=0
        )
        cm = confusion_matrix(y_true, y_pred)

    return {
        "accuracy": accuracy,
        "total": total,
        "errors": errors,
        "report_text": report_text,
```

```
        "confusion_matrix": cm,
        "correct": correct,
        "mistakes": mistakes
    }
```

2.8 Текст програми файлу train.py

```
import torch
from torch.utils.data import DataLoader
import torch.nn.functional as F
from torch.optim import Adam
import sys

class Trainer:
    def __init__(self, model, train_ds, val_ds, device="cuda"):
        self.model = model
        self.train_ds = train_ds
        self.val_ds = val_ds
        self.device = torch.device(device if torch.cuda.is_available() else "cpu")
        self.model.to(self.device)

    def train(
        self,
        epochs=30,
        batch_size=32,
        patience=5,
        save_path="model.pth",
        log_fn=print
    ):
        torch.manual_seed(42)

        train_loader = DataLoader(
            self.train_ds,
            batch_size=batch_size,
            shuffle=True
        )
        val_loader = DataLoader(
            self.val_ds,
            batch_size=batch_size,
            shuffle=False
        )

        optimizer = Adam(self.model.parameters(), lr=1e-3)

        best_val = -1.0
        patience_ctr = 0
        total_samples = len(self.train_ds)

        for epoch in range(1, epochs + 1):
            self.model.train()
            correct, total, loss_sum = 0, 0, 0.0
            processed = 0
```

```
print(f"\n--- Епоха {epoch}/{epochs} ---")

for x, y in train_loader:
    x, y = x.to(self.device), y.to(self.device)

    optimizer.zero_grad()
    logits = self.model(x)
    loss = F.cross_entropy(logits, y)
    loss.backward()
    optimizer.step()

    loss_sum += loss.item()
    preds = logits.argmax(dim=1)
    correct += (preds == y).sum().item()
    total += y.size(0)
    processed += y.size(0)

    percent = (processed / total_samples) * 100
    sys.stdout.write(
        f"\rПрогрес: {percent:.1f}% "
        f"({processed}/{total_samples}) "
        f"Loss: {loss.item():.4f}"
    )
    sys.stdout.flush()

train_acc = correct / total

self.model.eval()
val_correct, val_total = 0, 0

with torch.no_grad():
    for x, y in val_loader:
        x, y = x.to(self.device), y.to(self.device)
        logits = self.model(x)
        preds = logits.argmax(dim=1)
        val_correct += (preds == y).sum().item()
        val_total += y.size(0)

val_acc = val_correct / val_total

status_msg = (
    f"Epoch {epoch}: "
    f"loss={loss_sum / len(train_loader):.4f} "
    f"train_acc={train_acc:.4f} "
    f"val_acc={val_acc:.4f}"
)
print(f"\n{status_msg}")
log_fn(status_msg)

if val_acc > best_val:
    best_val = val_acc
    patience_ctr = 0
```

```
            torch.save(
                {
                    "state_dict": self.model.state_dict(),
                    "labels_map": self.train_ds.labels_map
                },
                save_path
            )

            log_fn(f" Модель збережена: {save_path}")
        else:
            patience_ctr += 1
            if patience_ctr >= patience:
                log_fn(f" Early stopping на епосі {epoch}")
                break
```

ДОДАТОК В
Керівництво користувача

ЗАТВЕРДЖУЮ
Перший проректор Українського державного
університету науки і технологій
Анатолій РАДКЕВИЧ

АНАЛІЗАТОР ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ РОЗПІЗНАВАННЯ УСНОГО МОВЛЕННЯ

Керівництво користувача. Керівництво з дослідження засобів штучного інтелекту для розпізнавання усного мовлення
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01530– 01 – І3 – 01

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Олена КУРОП’ЯТНИК
Виконавець
_____Катерина ПАЗИКА
Нормоконтролер
_____Світлана ВОЛКОВА

2025

ЗАТВЕРДЖЕНО

44165850.01530– 01 – І3 – 01

АНАЛІЗАТОР ЗАСОБІВ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ РОЗПІЗНАВАННЯ УСНОГО МОВЛЕННЯ

Керівництво користувача. Керівництво з дослідження засобів штучного інтелекту для розпізнавання усного мовлення

Листів 14

2025

Зміст

ВСТУП

Керівництво користувача описує порядок роботи з програмною системою «Аналізатор засобів штучного інтелекті для розпізнавання усного мовлення». Цей засіб призначений для проведення дослідження та порівняльного аналізу ефективності нейромережевих архітектур CRNN І MobileNetV2 та наборів даних Speech Commands. Програмний засіб дає можливість виконувати тренування нейромережевих моделей на різних наборах даних, розраховувати точність розпізнавання та формувати класифікаційний звіт за результатами тестування. Користувач повинен мати високу кваліфікацію та знати принципи навчання нейронних мереж і вміти інтерпретувати результати тестування. Для роботи з програмним засобом «Аналізатор засобів штучного інтелекті для розпізнавання усного мовлення» користувачу рекомендується ознайомитись з кваліфікаційною роботою та даним керівництвом користувача.

1 ПРИЗНАЧЕННЯ І УМОВИ ВИКОРИСТАННЯ

Програмний засіб призначений для автоматизації дослідницьких процесів у сфері машинного навчання. Основними автоматизованими функціями є налаштування параметрів та запуск процесу навчання моделей, автоматичний розрахунок показників ефективності, попередня обробка даних та формулювання текстового звіту тестування.

Умови використання: система функціонує на персональних комп'ютерах з операційною системою Microsoft Windows 10 або новіших версій, з процесором не AMD Ryzen 5 (або еквівалент), 8 ГБ оперативної пам'яті, 6 ГБ вільного дискового простору та графічним процесором. Користувач повинен мати базові навички роботи з Python та графічними інтерфейсами. Робота в приміщеннях з температурою 21-25 °C та відносною вологістю 40-60 % для забезпечення стабільності обладнання.

2 ПІДГОТОВКА ДО РОБОТИ

Роботу з програмним засобом «Аналізатор засобів штучного інтелекту для розпізнавання усного мовлення» почніть з розгортання середовища та встановлення компонентів. Встановіть IDE PyCharm та інтерпретатор Python 3.9. Далі обов’язково встановіть бібліотеки PyTorch, librosa, NumPy, Tkinter, Scikit-learn. Наступний крок – розархівувуйте папки з проектом у робочу директорію. Для коректного зчитування даних перевірте чи відповідають шляхи до папок датасетів фактичному розташуванню їх на диску.

Для перевірки працездатності запустіть програмний засіб. Поява вінка графічного інтерфейсу говорить про готовність системи до роботи. Після цього переходьте до тестування наявних моделей.

3 ОПИС ОПЕРАЦІЙ

Запустіть програму. При першому запуску програми відкриється стартове вікно (рис. В.1)

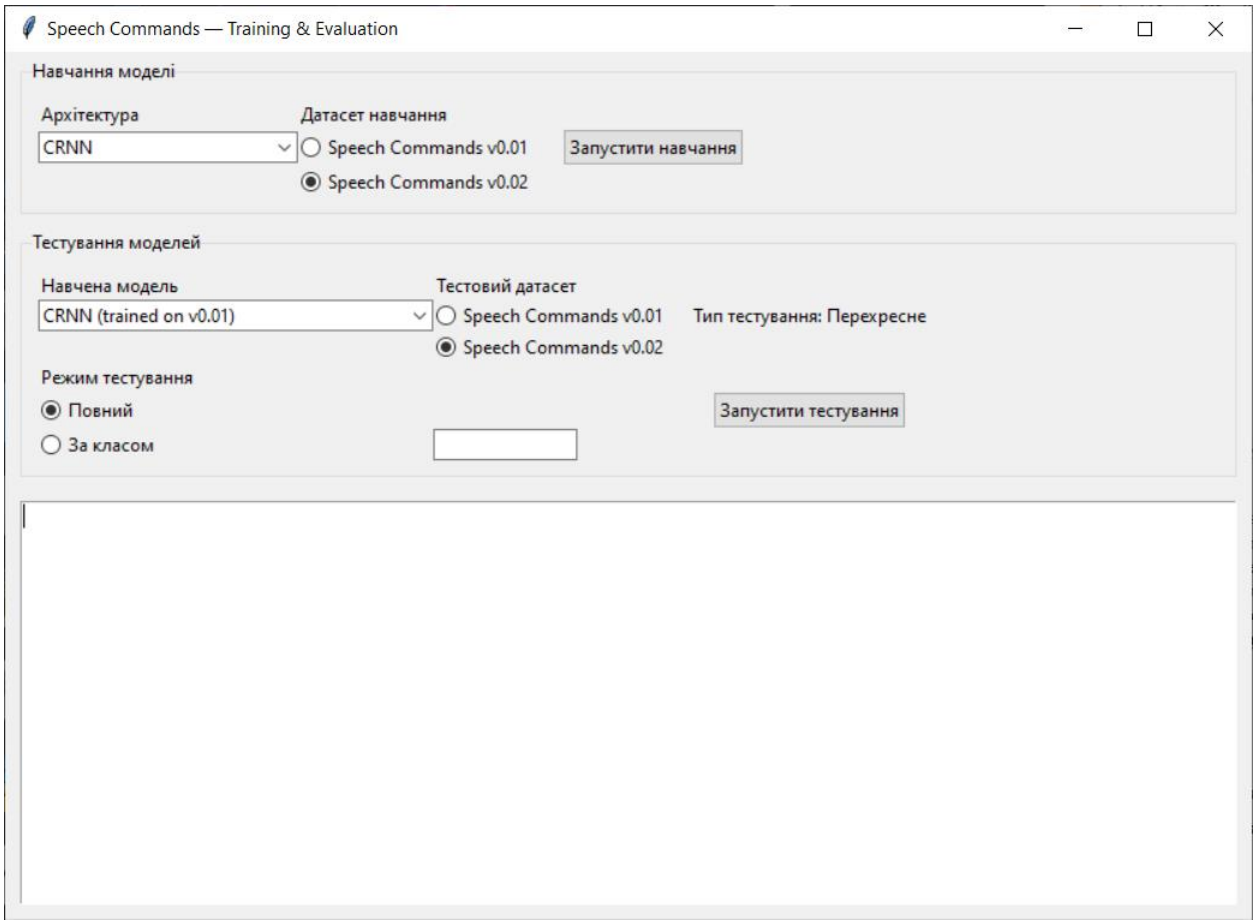

Рисунок В.1 – стартове вікно програми

Інтерфейс складається з:

- випадаючого списку для вибору архітектури моделі для навчання;
- перемикача версії датасету для навчання;
- кнопки запуску процесу навчання «Запустити навчання»;
- випадаючого списку для вибору архітектури моделі для тестування;
- перемикача версії датасету для тестування;
- перемикача режиму тестування (повний, за класом);
- текстового поля для введення назви класу, при використання режиму тестування за класами;
- кнопки запуску процесу тестування «Запустити тестування»;

- текстового поля режиму тестування, що динамічно змінюється в залежності від обраних параметрів;
- текстового поля для виведення результатів.

Наступний етап роботи це запуск операції навчання моделей (рис. В.2).

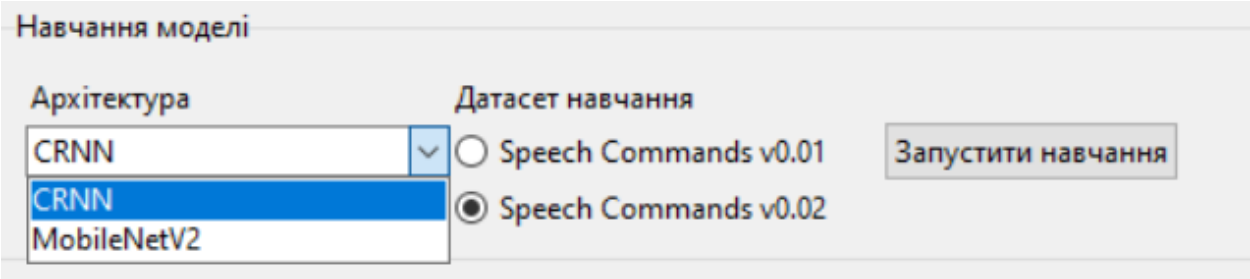

Рисунок В.2 – Параметри навчання моделі

Для початку операції навчання моделей оберіть з випадаючого списку архітектуру та встановіть покажчик вибору навпроти того датасету, який буде використовуватись при навчання. Після встановлення цих параметрів натисніть кнопку «Запустити навчання».

Після запуску операції навчання у текстове поле для виведення результатів буде виводитись процес перебігу тренування моделі (рис. В.3). Змінюючи параметри і проводячи навчання, можна отримати в результаті 4 моделі для розпізнавання мовних команд: CRNN та MobileNetV2 треновані відповідно на двох різних датасетах. Після завершення операції навчання та отримання чотирьох моделей переходьте до їх тестування.

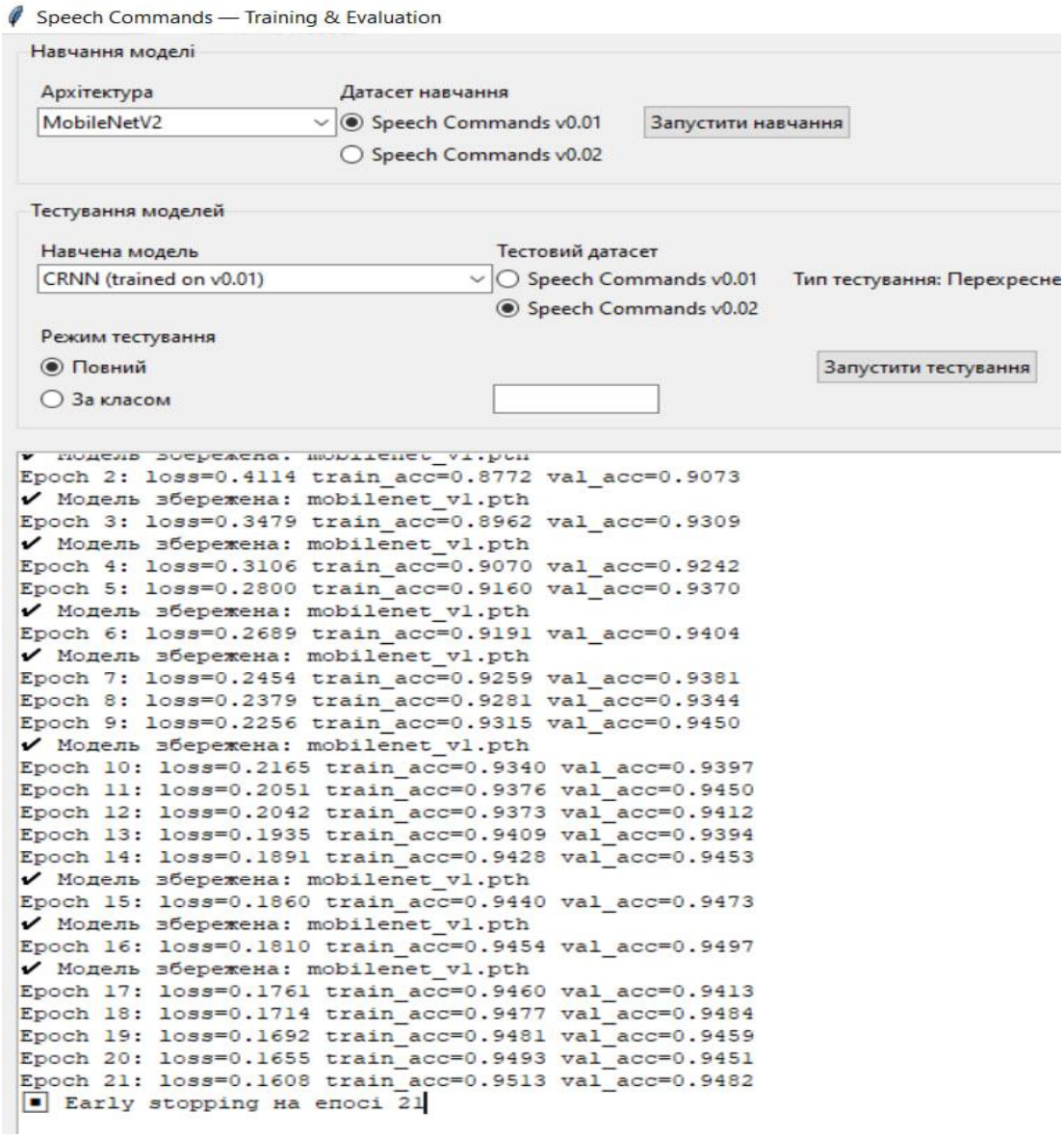

Рисунок В.3 – Процес навчання моделі

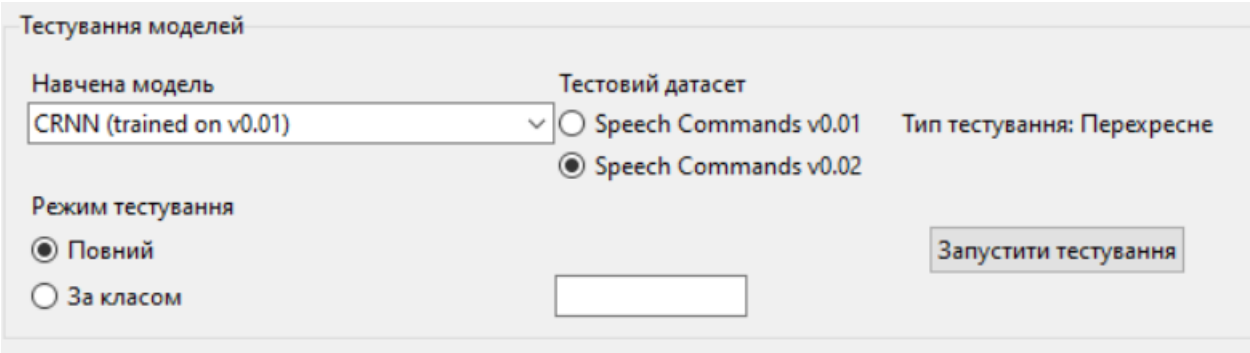

Рисунок В.4 – Параметри тестування моделей

Для початку операції тестування оберіть модель із запропонованих у випадаючому списку. Далі встановіть покажчик вибору навпроти того датасету, який буде використовуватись при тестуванні обраної моделі. Якщо датасет на якому модель навчалась не співпадає з обраним датасетом для тренування, поле «Тип тестування» змінює статус на «Перехресне», а якщо співпадають – «Пряме». Режим тестування встановлено за замовчуванням «Повний». Це означає, що користувач отримає повний звіт про результати тестування, що включає класифікаційний звіт та перші 10 прикладів правильного і неправильного розпізнавання.

Після встановлення всіх параметрів тестування натисніть кнопку «Запустити тестування» для початку тестування моделі. Після успішного закінчення тестування, у текстове поле для виведення результатів буде виведено класифікаційний звіт та приклади правильних і неправильних розпізнавань (рис.В.6).

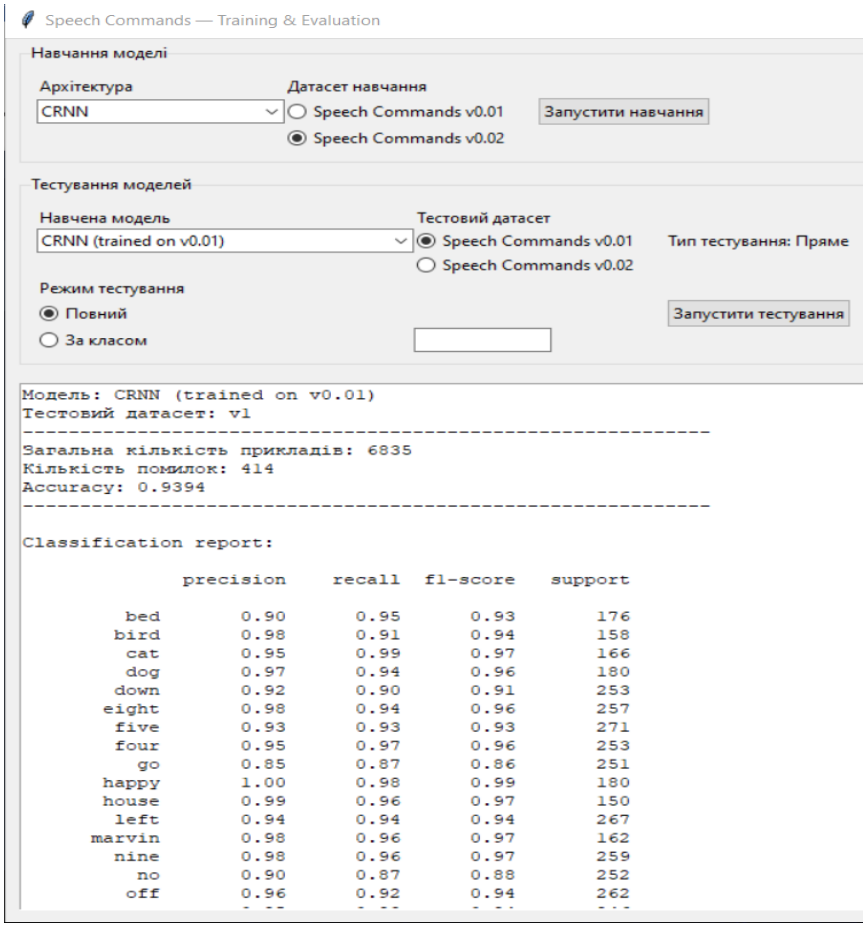

Рисунок В.6 – Результати повного тестування

Оберіть режим тестування «За класом». Уведіть назву класу в текстове поле поряд. Назва класу має входити до множини класів обраного датасету. У випадку якщо введено назву неіснуючого класу, виводиться помилка (рис. В.5).

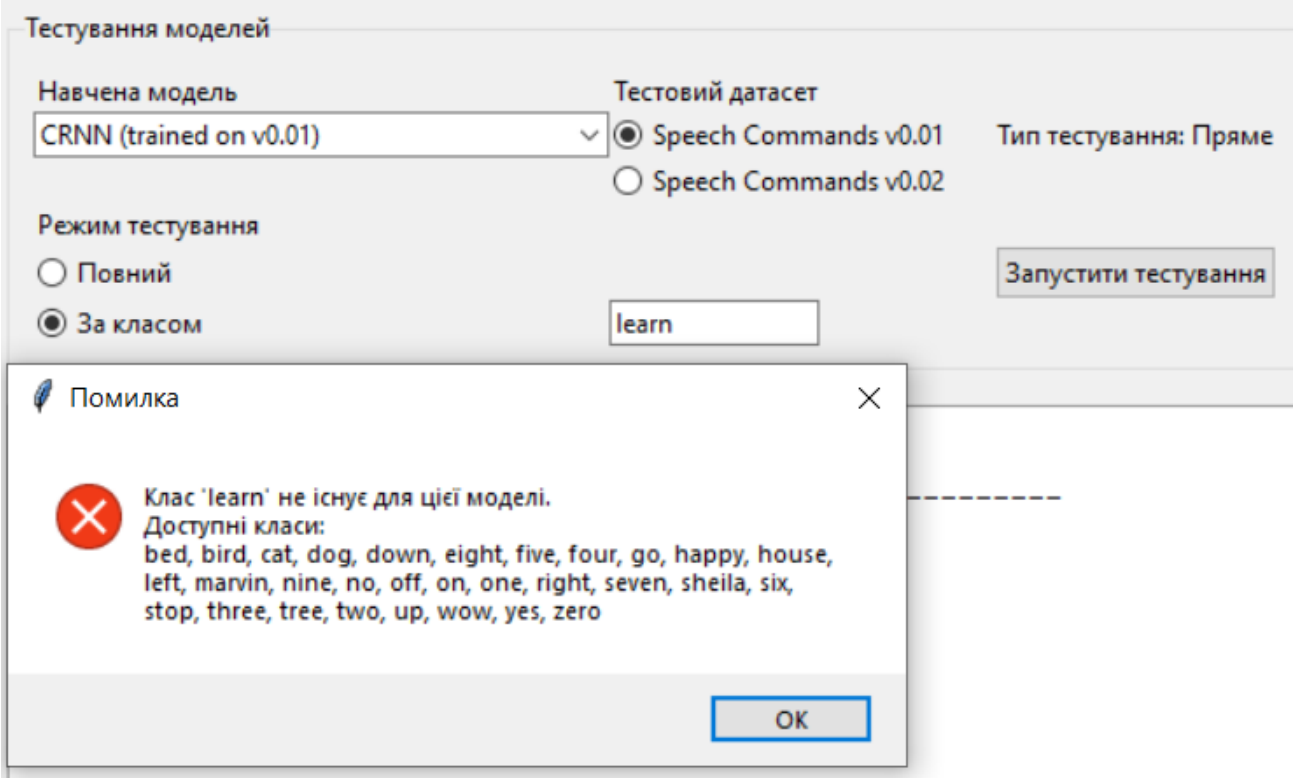

Рисунок В.5 – Помилка вибору класу

Після успішного закінчення тестування за класом, у текстове поле для виведення результатів буде виведено класифікаційний звіт по обраному класу та приклади правильних і неправильних розпізнавань (рис. В.7).

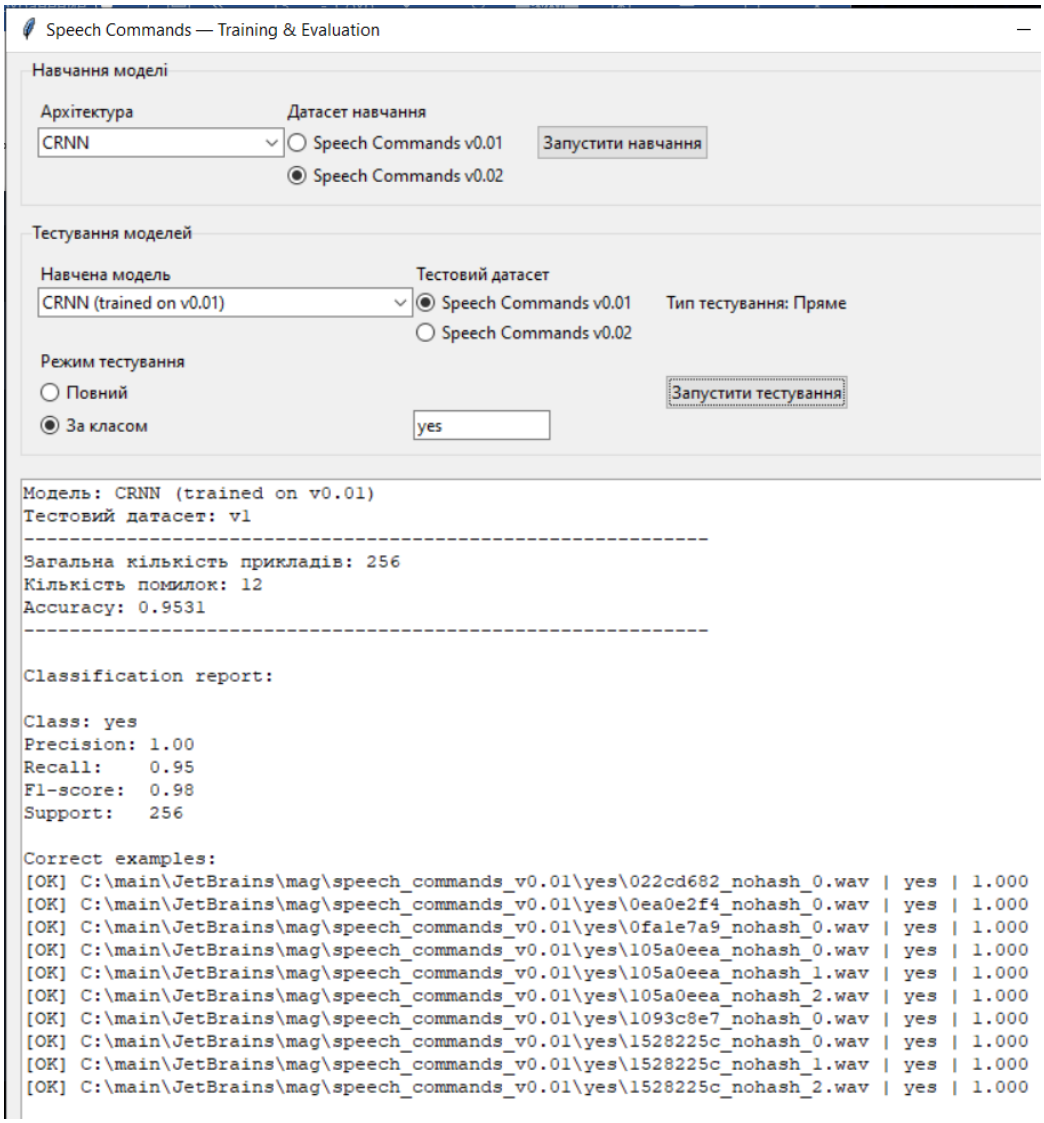

Рисунок В.7 – Результати тестування за класом

3 АВАРІЙНІ СИТУАЦІЇ

При пошкодженні файлів або випадкового видалення програмних модулів відновіть їх із резервної копії (архіву).

Тривалий процес навчання може призвести до сильного нагріву процесора. При спрацюванні теплового захисту зупиніть навчання, забезпечте додаткове охолодження ЕОМ та продовджуйте роботу після зниження температури.

При пошкодженні аудіофайлів датасету повторно завантежте набір даних Speech Commands з офіційного джерела.

3 РЕКОМЕНДАЦІЇ

Роботу з програмним забезпеченням починайте з етапу тестування наявних навчених моделей. Це дозволить швидко оцінити правильність роботи програми та ознайомитись із форматом вихідних звітів. Перед початком навчання нейромереж переконайтесь у наявності достатнього обсягу оперативної пам’яті та закрийте сторонні додатки для стабілізації роботи процесора.

ДОДАТОК Г


UKRAINIAN STATE UNIVERSITY OF SCIENCE AND TECHNOLOGIES
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УКРАЇНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ НАУКИ І ТЕХНОЛОГІЙ
ABSTRACTS OF THE XIX INTERNATIONAL CONFERENCE «MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
18-19, December, 2025
СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ І ОСВІТІ
ПРИСВЯЧЕНО ПАМ'ЯТІ ПРОФЕСОРА ІГОРЯ ЖУКОВИЦЬКОГО
ТЕЗИ
XIX МІЖНАРОДНОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ
18-19 ГРУДНЯ 2025
ДНІПРО
2025

Дослідження засобів штучного інтелекту для розпізнавання усного мовлення

Куроп'ятник О. С., Пазика К. С., Український державний університет науки і технологій, Україна

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням ролі штучного інтелекту у різних сферах діяльності людини. Одним із найбільш складних і водночас перспективних напрямів є автоматичне розпізнавання усного мовлення. ASR(Automatic Speech Recognition) лежить в основі таких популярних технологій як голосові асистенти, системи керування «розумним домом», автомобільна промисловість, альтернативне введення даних для людей з порушенням зору і багато інших. Актуальність досліджень у цій галузі зумовлена зростаючими вимогами до точності, швидкості та універсальності систем розпізнавання мовлення.

Розпізнавання усного мовлення є складним завданням, оскільки мовний сигнал має змінну природу. Акустичні характеристики одного й того самого слова або звуку ніколи не є ідентичними і постійно змінюються залежно від низки факторів, таких як індивідуальні особливості диктора, акцент, темп мовлення, інтонація, рівень шуму, якість запису та мовний контекст. До середини 2010-х років домінуючими методами у сфері розпізнавання мовлення були приховані марківські та гауссові сумішеві моделі. У таких системах мовний сигнал розглядався як послідовність станів, що змінюються у часі, а акустичні ознаки — наприклад, мел-кепстральні коефіцієнти (MFCC) — виділялися вручну за допомогою класичних алгоритмів цифрової обробки сигналів. Незважаючи на свою ефективність у контрольованих умовах, подібні підходи мали обмежену здатність адаптуватися до різноманітних дикторів, шумів та нестандартних умов запису.

З розвитком обчислювальних ресурсів дедалі більшого поширення набули end-to-end системи розпізнавання мовлення. На відміну від класичних підходів, такі моделі навчаються безпосередньо від акустичного сигналу до текстової транскрипції, мінімізуючи кількість ручних етапів обробки. У межах end-to-end підходів використовуються різні архітектури, зокрема рекурентні нейронні мережі (RNN), довготривала короткочасна пам'ять (LSTM), згорткові нейронні мережі (CNN), а також трансформерні моделі.

У межах дослідження проводиться аналіз різних архітектур нейронних мереж, які застосовуються для розпізнавання мовлення, зокрема комбінована згортково-рекурентна модель(CRNN). Запропонована модель поєднує згорткові шари, що виконують автоматичне виділення локальних спектральних ознак із мел-спектрограми мовного сигналу, та рекурентні шари, які дозволяють враховувати часову динаміку та послідовну структуру мовлення. Такий підхід забезпечує більш повне моделювання акустичних особливостей усного мовлення та підвищує точність розпізнавання в умовах варіативності сигналу та наявності шумів. Окремо розглянуто можливість використання попередньо натренованих моделей, що дає змогу зменшити обчислювальні витрати та скоротити час навчання.

Для експериментальної перевірки ефективності досліджуваних підходів використано відкритий набір аудіоданих, що містить записи коротких голосових команд. Оцінювання якості розпізнавання здійснювалося за стандартними метриками класифікації, такими як точність, повнота та F1-міра, що дозволяє комплексно оцінити роботу моделей. За отриманими результатами можна зробити висновки про доцільність застосування методів глибинного навчання для задач розпізнавання усного мовлення та їх перевагу над традиційними підходами.