

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістра»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Розпізнавання емоцій обличчя людини з використанням нейронних мереж»

Виконав студент групи ПЗмп-24-1
ЗАГЛАДА Мирослав Русланович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ЧИЖМОТРЯ Олексій Володимирович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«3» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЗАГЛАДА Мирослав Русланович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Розпізнавання емоцій обличчя людини з використанням нейронних мереж»**,

затверджена Наказом закладу вищої освіти від «3» лютого 2026 р. №62/с

Вихідні дані для роботи: практичні аспекти та використання нейронних мереж для розпізнавання емоцій на обличчі людини.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	03.02.2026р.	20.02.2026р.
2	Єфремов Ю.М.	20.02.2026р.	15.03.2026р.
3	Єфремов Ю.М.	15.03.2026р.	20.03.2026р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «магістр» присвячена розпізнаванню емоцій обличчя людини з використанням нейронних мереж. Пояснювальна записка складається зі вступу, трьох розділів та списку використаної літератури. Пояснювальна записка до кваліфікаційної магістерської роботи містить 65 сторінок, 19 ілюстрацій, 3 таблиці, 1 додаток та 20 посилань на зовнішні джерела.

Метою роботи є тренування моделі нейронної мережі для класифікації емоцій обличчя та розробка програмного додатку із зручним графічним інтерфейсом для розпізнавання емоційних станів на зображеннях, у відеофайлах та в режимі реального часу.

В роботі описано процес аналізу предметної області та існуючих рішень, проектування та навчання згорткової нейронної мережі VGG13 на наборі даних FER+ з використанням методів PLD та CEL, а також розробку програмного додатку з графічним інтерфейсом. Досліджено ефективність двох стратегій навчання на краудсорсингових розподілах міток. Проведено тестування програми у всіх режимах роботи.

КЛЮЧОВІ СЛОВА: НЕЙРОННА МЕРЕЖА, РОЗПІЗНАВАННЯ ЕМОЦІЙ, VGG13, FER+, ГЛИБОКЕ НАВЧАННЯ, ONNX.

ABSTRACT

The master's thesis is devoted to facial emotion recognition using neural networks. The explanatory note consists of an introduction, three chapters and a list of used literature. The thesis contains 65 pages, 19 illustrations, 3 tables, 1 appendix and 20 references to external sources.

The purpose of the work is to train a neural network model for facial emotion classification and develop a software application with a user-friendly graphical interface for emotion recognition on images, video files and in real time.

The work describes the process of analyzing the subject area and existing solutions, designing and training the VGG13 convolutional neural network on the FER+ dataset using PLD and CEL methods, and developing a software application with a graphical interface. The effectiveness of two training strategies on crowd-sourced label distributions has been studied. The program has been tested in all operating modes.

KEY WORDS: NEURAL NETWORK, EMOTION RECOGNITION, VGG13, FER+, DEEP LEARNING, ONNX.

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	9
1.1 Аналіз проблем і особливостей розпізнавання емоцій.....	9
1.2 Аналіз наявних методів розпізнавання емоцій.....	12
1.3 Постановка задачі.....	17
1.4 Вибір засобів реалізації.....	18
Висновки до розділу 1.....	22
РОЗДІЛ 2. ПРОЕКТУВАННЯ АЛГОРИТМІВ ДЛЯ НАВЧАННЯ МЕРЕЖІ РОЗПІЗНАВАННЯ ЕМОЦІЙ.....	23
2.1 Визначення процесу навчання та архітектури моделі нейронної мережі.....	23
2.2 Аналіз набору даних для проведення навчання.....	27
2.3 Реалізація алгоритмів для навчання моделі.....	30
2.4 Порівняння результатів навчання.....	39
Висновки до розділу 2.....	41
РОЗДІЛ 3. ОПИС ПРОГРАМНОГО ПРОДУКТУ.....	43
3.1 Розробка графічної частини додатку для розпізнавання емоцій.....	43
3.2 Опис графічного інтерфейсу.....	47
3.3 Тестування роботи додатку.....	51
Висновки до розділу 3.....	56
ВИСНОВКИ.....	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	60
ДОДАТКИ.....	63

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

NN - (англ. neural network) - нейронна мережа

CNN - (англ. convolutional neural network) - Згорткова нейронна мережа

STD - (англ. standard deviation) - середнє квадратичне відхилення

GUI - (англ. graphical user interface) - графічний інтерфейс

ВСТУП

Актуальність теми. Розпізнавання емоцій людини за виразом обличчя є важливою науково-прикладною задачею, яка знаходиться на перетині комп'ютерного зору, штучного інтелекту та психології. Інтерес до цієї проблеми обумовлений широким спектром практичних застосувань: від систем безпеки та моніторингу до медичної діагностики, освітніх технологій та сфери обслуговування. Автоматичне визначення емоційного стану людини дозволяє створювати більш інтелектуальні системи взаємодії, що здатні адаптуватися до потреб користувача.

Протягом останніх десятиліть підходи до розпізнавання емоцій зазнали значної еволюції. Ранні методи, засновані на ручному конструюванні ознак, поступово поступаються місцем підходам на основі глибокого навчання. Нейронні мережі, зокрема згорткові архітектури, продемонстрували значно вищу точність та узагальнюючу здатність, оскільки вони здатні автоматично виявляти складні закономірності безпосередньо з вхідних зображень без необхідності попереднього ручного виділення ознак.

Попри значний прогрес у даній галузі, залишається потреба у створенні комплексних програмних рішень, які б поєднували можливості тренування моделі з її практичним використанням для розпізнавання емоцій у різних режимах роботи. Це зумовлює актуальність даного дослідження.

Мета роботи - тренування моделі нейронної мережі для розпізнавання емоцій обличчя людини та розробка програмного додатку який дозволяє використовувати навчену модель для аналізу емоційного стану на зображеннях, у відеофайлах та в режимі реального часу через веб-камеру.

Для досягнення поставленої мети необхідно вирішити такі основні завдання:

1. Провести аналіз існуючих методів та підходів до розпізнавання емоцій обличчя людини з використанням нейронних мереж.

2. Дослідити особливості набору даних FER+ та методи навчання на основі краудсорсингових розподілів міток.

3. Реалізувати архітектуру згорткової нейронної мережі VGG13 та навчити модель на відповідному наборі даних.

4. Розробити модуль детекції облич та класифікації емоцій для обробки зображень, відеофайлів та відеопотоку з веб-камери.

5. Створити графічний інтерфейс користувача для зручного використання розробленого програмного додатку.

Об'єктом дослідження є процес автоматизованого розпізнавання емоційних станів людини за виразом обличчя із застосуванням методів глибокого навчання.

Предметом дослідження є методи та алгоритми навчання згорткових нейронних мереж на основі краудсорсингових розподілів міток для класифікації емоцій обличчя людини.

В атестаційній роботі використано методи глибокого навчання та згорткових нейронних мереж, методи обробки зображень, імовірнісні методи навчання на розподілах міток, методи оптимізації (стохастичний градієнтний спуск, адаптивна оптимізація), методи валідації та оцінки якості моделей класифікації, методи об'єктно-орієнтованого проектування та програмування.

Наукова новизна роботи полягає у застосуванні методів імовірнісного вибіркового навчання (PLD) та перехресної ентропійної функції втрат (CEL) для тренування згорткової нейронної мережі архітектури VGG13 на наборі даних FER+ із реалізацією повного циклу від навчання моделі до її практичного застосування у вигляді програмного додатку з графічним інтерфейсом. Особливістю роботи є інтеграція детекції облич з класифікацією емоцій у єдиний конвеєр обробки, що забезпечує роботу в режимі реального часу із підтримкою восьми класів емоцій.

Практичне значення одержаних результатів полягає в розробці програмного додатку який дозволяє тренувати моделі розпізнавання емоцій обличчя та використовувати їх для аналізу зображень, відеофайлів і

відеопотоку з веб-камери в режимі реального часу. Програмний продукт може бути застосований у системах моніторингу емоційного стану, дослідженнях у галузі психології та поведінкових наук, системах автоматизованого контролю якості обслуговування, а також як навчальний інструмент для вивчення методів глибокого навчання. Експорт навченої моделі у формат ONNX забезпечує портативність та можливість інтеграції в інші програмні системи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз проблем і особливостей розпізнавання емоцій

Емоції є фундаментальним аспектом людського спілкування та соціальної взаємодії. Здатність правильно інтерпретувати емоційний стан співрозмовника відіграє ключову роль у повсякденному житті, професійній діяльності та міжособистісних стосунках. Міжкультурне дослідження проведене у 1971 році, продемонструвало, що представники різних культур включаючи ізольовані племена Папуа Нової Гвінеї, однаково пов'язують певні вирази обличчя з відповідними емоціями, що стало основою теорії універсальних базових емоцій [1]. На основі цих спостережень було сформульовано теорію базових емоцій яка виділяє шість універсальних категорій: гнів, страх, сум, радість, огида та подив [2]. Кожна з цих емоцій характеризується унікальним набором фізіологічних реакцій, специфічними виразами обличчя та типовими ситуаціями виникнення. Ця класифікація стала загальноприйнятою у дослідженнях з автоматичного розпізнавання емоцій, хоча деякі автори розширюють її додатковими категоріями, такими як зневага чи нейтральний стан.

Незважаючи на наявність чіткої теоретичної бази, автоматичне розпізнавання емоцій за виразом обличчя стикається з цілою низкою практичних проблем. Комплексний огляд сучасних підходів до глибокого розпізнавання емоцій систематизує основні виклики цієї галузі [7], серед яких виділяються наступні: по-перше, суттєвий вплив чинить варіативність умов зйомки. Зміни освітлення, різні ракурси та часткове перекриття обличчя (окуляри, волосся, тощо) значно ускладнюють задачу класифікації. Модель, навчена на фронтальних фотографіях з рівномірним освітленням, може демонструвати значно нижчу точність у реальних умовах де ці фактори присутні одночасно. По-друге, серйозною проблемою є суб'єктивність анотації. Сприйняття емоцій за виразом обличчя є індивідуальним процесом - один і той самий вираз різні анотатори можуть інтерпретувати по-різному. Дослідження

показують, що при використанні малої кількості анотаторів узгодженість з еталонною міткою є вкрай низькою, і лише залучення множинних анотацій дозволяє суттєво підвищити якість навчальних даних [5, 6]. Третьою важливою проблемою є дисбаланс класів у наборах даних. Зображення з нейтральним виразом обличчя та радістю зустрічаються значно частіше, ніж зображення із зневагою чи огидою, що призводить до зміщення моделі в бік більш представлених класів [3]. Окрім цього, варто зазначити й культурні відмінності у прояві емоцій, вікові особливості міміки, а також складність розпізнавання мікроекспресій, тобто короточасних виразів обличчя тривалістю менше 500 мілісекунд.

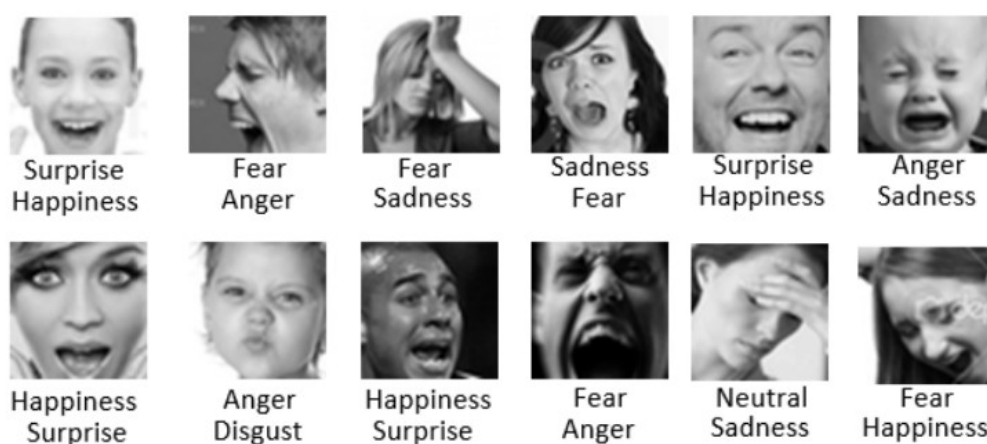


Рис. 1.1. Приклади зображень з набору даних FER+

Усі ці фактори роблять задачу автоматичного розпізнавання емоцій доволі складною у галузі комп'ютерного зору. Для її вирішення сучасні дослідники все частіше звертаються до методів глибокого навчання, зокрема до згорткових нейронних мереж. Згорткові нейронні мережі це клас глибоких нейронних мереж спеціально розроблених для вирішення завдань обробки та розпізнавання зображень. Вони відіграли ключову роль у розвитку комп'ютерного зору дозволивши комп'ютерам розуміти та інтерпретувати візуальні дані з надзвичайною точністю [7]. CNN зробили революцію в різних сферах: від автономного водіння до медичної візуалізації, що зробило їх наріжним каменем сучасного штучного інтелекту. Однією з визначальних характеристик CNN є їхня здатність автоматично вивчати ієрархічні ознаки з

необроблених значень пікселів, що має вирішальне значення для виявлення значущих патернів на різних рівнях абстракції, вони досягають цього за допомогою серії шарів, кожен з яких слугує певній меті.

Згорткові шари застосовують згорткові фільтри до вхідних зображень, дозволяючи мережі виявляти такі особливості як краї, кути і текстури. Згортання передбачає поелементне множення фільтра на локальні ділянки зображення з подальшим підсумовуванням. Кілька фільтрів вивчають різні особливості на кожному рівні. Шари об'єднання (pooling) зменшують просторові розміри карт ознак, знижуючи обчислювальну складність та підвищуючи стійкість до просторових зсувів. Наприклад, максимальне об'єднання (max pooling) вибирає максимальне значення з групи сусідніх пікселів, виділяючи найбільш характерні ознаки. Повнозв'язні шари у фінальній частині мережі згладжують карти ознак та виконують безпосередньо задачу класифікації або регресії на основі вивчених ознак. Вихідний шар зазвичай використовує функцію softmax для багатокласової класифікації, надаючи ймовірності для кожного класу.

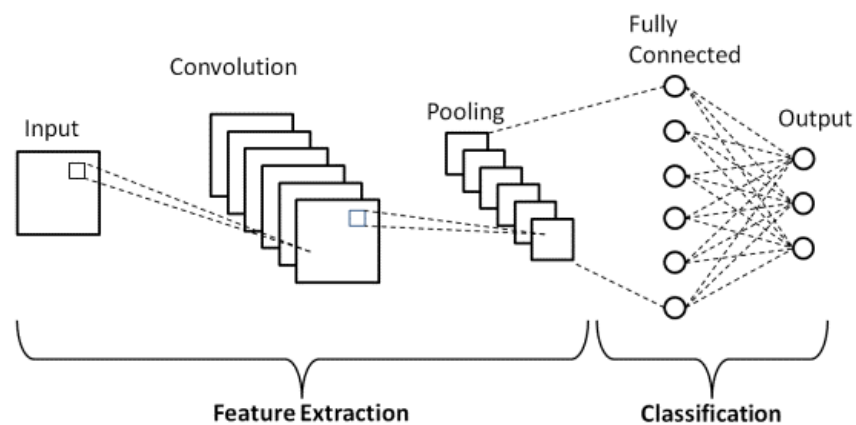


Рис. 1.2. Структура типової згорткової нейронної мережі

Основною концепцією згорткових шарів є спільне використання параметрів, що значно зменшує кількість параметрів які потрібно навчити. До всього зображення застосовується один фільтр, який навчається виявляти певну ознаку незалежно від її розташування. Ця властивість робить CNN інваріантними до переміщень та здатними розпізнавати патерни незалежно від їхнього положення на зображенні. Згорткові навчаються за допомогою

зворотного поширення похибки - процесу, який передбачає ітеративне налаштування параметрів мережі для мінімізації різниці між прогнозованими та фактичними результатами. Градієнти функції втрат обчислюються та використовуються для оновлення ваг за допомогою алгоритмів оптимізації, таких як стохастичний градієнтний спуск (SGD) або Adam. Щоб покращити узагальнення та запобігти перенавчанню, часто використовують аугментацію даних: застосування перетворень (обертання, обрізання, відображення) для штучного збільшення навчального набору. Методи регуляризації, такі як dropout, який випадковим чином деактивує нейрони під час навчання, також сприяють запобіганню перенавчанню.

Саме згорткові нейронні мережі є оптимальним вибором для задачі розпізнавання емоцій, оскільки вони здатні автоматично виділяти характерні мімічні ознаки з зображень облич без необхідності ручного конструювання ознак яке вимагали традиційні підходи.

1.2 Аналіз наявних методів розпізнавання емоцій

На сьогоднішній день існує ряд комерційних та відкритих рішень, що надають можливість розпізнавання емоцій за виразом обличчя. Аналіз цих рішень дозволяє виявити їх сильні та слабкі сторони та визначити вимоги до розроблюваної системи.

Amazon Rekognition це хмарний сервіс Amazon Web Services для аналізу зображень та відео включає функцію розпізнавання емоцій як частину ширшого набору інструментів комп'ютерного зору [13]. Сервіс класифікує вирази обличчя за 8 категоріями (радість, сум, гнів, здивування, огида, страх, спокій, зневага) і повертає рівень впевненості для кожної емоції.

Він підтримує обробку як окремих зображень, так і відеопотоків, що робить його одним із найбільш функціональних хмарних рішень. Інтеграція з іншими сервісами AWS (S3, Lambda, Kinesis) дозволяє будувати складні конвеєри обробки даних. Однак сервіс працює виключно через хмарний API, що вимагає постійного з'єднання з Інтернетом та створює залежність від

інфраструктури провайдера. Код сервісу є закритим, а вартість залежить від обсягу оброблених зображень, що може бути суттєвим фактором при великих обсягах даних.

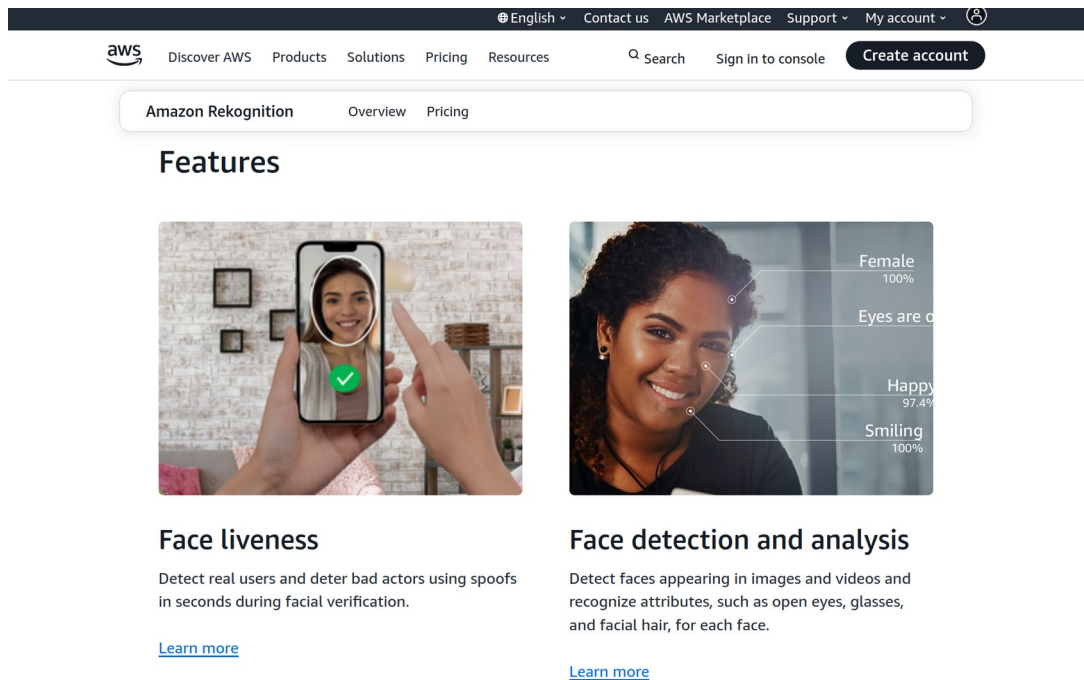


Рис. 1.3. Сервіс Amazon Rekognition

Face++ (Megvii) - китайська платформа комп'ютерного зору Face++ пропонує API для розпізнавання емоцій, який аналізує вирази обличчя та повертає оцінки впевненості для 7 базових емоцій [14]. Платформа є одним із найпопулярніших рішень на азійському ринку та активно використовується в комерційних продуктах.

Face++ забезпечує високу швидкість обробки та підтримку різних мов програмування, включаючи Python, Java та C#. Платформа надає безкоштовний тарифний план з обмеженою кількістю запитів, що дозволяє розробникам протестувати функціональність перед повномасштабним впровадженням. Проте Face++ працює виключно як хмарний сервіс, орієнтований переважно на обробку окремих зображень, і не надає вбудованої функціональності для обробки відеофайлів чи потокового відео з веб-камери.

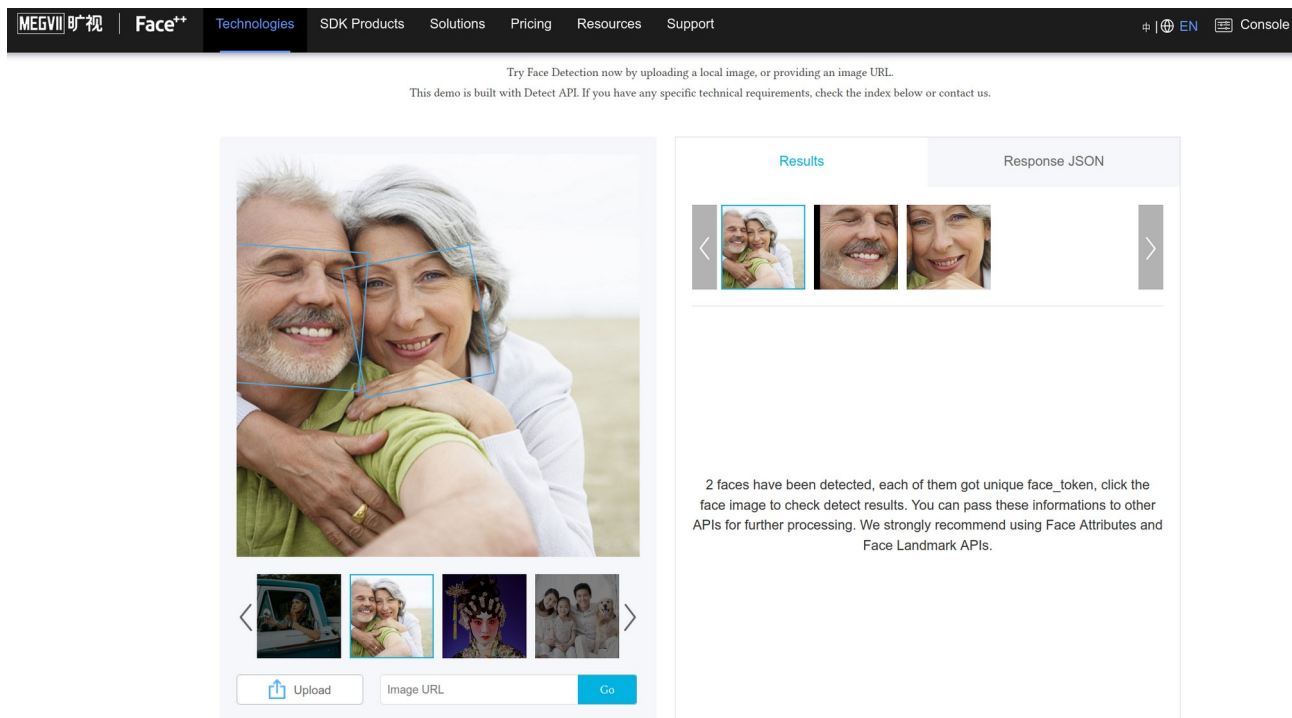


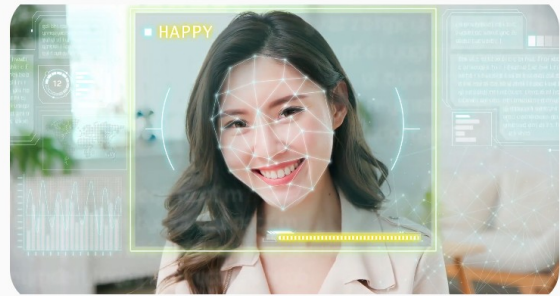
Рис. 1.4. Сервіс Face++

Affectiva (iMotions) є одним із лідерів у галузі афективних обчислень та розпізнавання емоцій, і вважається золотим стандартом у сфері автоматизованого аналізу виразів обличчя [15]. Їхній SDK здатен розпізнавати 7 базових емоцій, а також два додаткових стани: сентиментальність та розгубленість.

Система базується на глибоких нейронних мережах, навчених на масштабному наборі даних із зображень тисяч учасників з різних країн, що забезпечує високу точність розпізнавання для різних демографічних груп. Affectiva широко використовується в академічних дослідженнях, маркетинговому тестуванні рекламних матеріалів та системах моніторингу водіїв. Однак продукт є комерційним із закритим кодом, ліцензія на використання SDK має значну вартість, а автономна робота без підключення до серверів ліцензування не підтримується. Користувачі також не мають можливості тренувати власні моделі або модифікувати існуючі.

Affectiva Facial Coding SDK

Incorporate Real-Time Emotion Sensing In Digital Experiences and Applications



Develop Emotion Sensing Solutions with Facial Coding SDK

Affectiva Facial Coding SDK is the industry-leading solution for real-time facial expression analysis, enabling businesses, researchers, and developers to decode human emotions with precision. Leveraging deep learning models trained on thousands of diverse participants, Affectiva Facial Coding SDK provides highly accurate emotion detection, outperforming competitors in real-world conditions.

Whether you're conducting ad testing, user experience research, driver monitoring, or healthcare studies, Affectiva Facial Coding SDK delivers scalable, cross-platform Facial Coding AI that integrates seamlessly into your workflow.

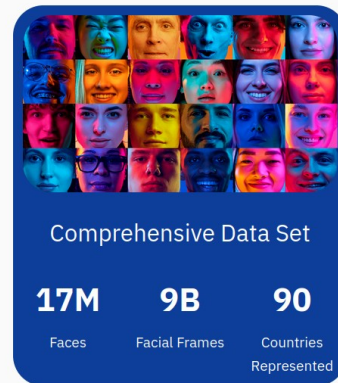


Рис. 1.5. Сервіс Affectiva

DeerFace це відкрита бібліотека на Python для розпізнавання облич та аналізу атрибутів обличчя, включаючи вік, стать, расу та емоції [16]. Бібліотека є одним із найпопулярніших відкритих рішень у цій галузі та інтегрує декілька попередньо навчених моделей для детекції та розпізнавання облич (VGG-Face, FaceNet, ArcFace тощо).

DeerFace класифікує емоції за 7 категоріями та забезпечує повністю локальну роботу без необхідності хмарних з'єднань. Відкритий вихідний код дозволяє розробникам вивчати внутрішню логіку роботи та адаптувати бібліотеку під власні потреби. Однак DeerFace не надає графічного інтерфейсу для кінцевого користувача, не підтримує пакетну обробку відеофайлів «з коробки» та, що найважливіше не дозволяє тренувати власні моделі класифікації емоцій на нових наборах даних оскільки тут використовуються лише попередньо навчені ваги.



Рис. 1.6. Бібліотека DeepFace

Результати порівняльного аналізу існуючих рішень наведено у таблиці 1.1.

Таблиця 1.1

Порівняльна характеристика існуючих рішень для розпізнавання емоцій

Параметр	Amazon Rekognition	Face++	Affectiva (iMotions)	DeepFace
Кількість емоцій	8	7	9	7
Режим реального часу	Так	Так	Так	Так
Обробка відео-файлів	Так	Ні	Так	Ні
Локальна робота (offline)	Ні	Ні	Ні	Так
Відкритий код	Ні	Ні	Ні	Так
Власне тренування моделі	Ні	Ні	Ні	Ні
Безкоштовний доступ	Частково	Частково	Ні	Так
Експорт моделі (ONNX)	Ні	Ні	Ні	Ні

З наведеної таблиці видно, що існуючі рішення мають ряд суттєвих обмежень. Комерційні хмарні сервіси забезпечують високу якість розпізнавання, проте вимагають постійного підключення до Інтернету, мають

закритий код та не дозволяють тренування власних моделей. Спеціалізоване рішення Affectiva є дорогим та закритим. Відкрита бібліотека DeepFace забезпечує локальну роботу, але не підтримує обробку відеофайлів та не надає можливості навчання на власних даних.

1.3 Постановка задачі

На підставі проведеного аналізу предметної області та існуючих рішень можна сформулювати основні завдання які необхідно вирішити в рамках даної роботи.

Головною метою роботи є тренування моделі нейронної мережі для розпізнавання емоцій обличчя людини та розробка програмного додатку, який дозволяє використовувати навчену модель для аналізу емоційного стану на зображеннях, у відеофайлах та в режимі реального часу через веб-камеру.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Реалізувати архітектуру згорткової нейронної мережі на основі якої буде відбуватись тренування.
2. Реалізувати два методи навчання на мітках: імовірнісне вибіркове навчання (PLD), при якому мітка для кожного зображення щоепохи випадково вибирається з розподілу анотаторів, та перехресну ентропійну функцію втрат (CEL), що безпосередньо використовує розподіл міток як цільовий вектор.
3. Підготувати конвеєр обробки набору даних, що включає завантаження зображень та міток, нормалізацію, аугментацію даних та розбиття на навчальну, валідаційну та тестову вибірки.
4. Навчити модель з використанням GPU-прискорення та оцінити її ефективність на тестовій вибірці, порівнявши результати методів PLD та CEL.
5. Реалізувати модуль розпізнавання облич на основі попередньо натренованих моделей та модуль класифікації емоцій, що здійснює передобробку виявлених регіонів облич та їх класифікацію навченою моделлю.

6. Розробити графічний інтерфейс користувача, що забезпечує зручне використання системи у трьох режимах: аналіз окремих зображень, обробка відеофайлів та розпізнавання у реальному часі з веб-камери.

7. Реалізувати експорт навченої моделі у формат ONNX [13] для забезпечення портативності та можливості інтеграції в інші програмні системи.

Система повинна класифікувати вирази обличчя за 8 категоріями емоцій: нейтральний стан, радість, подив, сум, гнів, огида, страх та зневага. При цьому для кожного виявленого обличчя на виході надається мітка найбільш імовірної емоції та відповідний рівень впевненості моделі.

1.4 Вибір засобів реалізації

Вибір інструментальних засобів для реалізації системи зумовлений специфікою задачі глибокого навчання та необхідністю забезпечення ефективної обробки зображень у реальному часі. Система охоплює повний цикл роботи з нейронною мережею, починаючи від підготовки навчальних даних та тренування моделі і завершуючи розпізнаванням емоцій на зображеннях, у відеопотоці та через веб-камеру в режимі реального часу. Для кожного з цих етапів необхідні спеціалізовані інструменти які забезпечують високу продуктивність, зручність розробки та можливість гнучкого налаштування всіх компонентів системи.

Основною мовою програмування для реалізації проекту обрано Python яка є провідною мовою у сфері машинного навчання та штучного інтелекту завдяки лаконічному синтаксису, великій кількості спеціалізованих бібліотек та активній спільноті розробників, а також через її простоту, що забезпечує швидке прототипування та зручну інтеграцію різномірних компонентів системи: від навчання нейронних мереж до обробки зображень та створення графічного інтерфейсу користувача. Важливою перевагою Python є його кросплатформенність, що дозволяє розгорнути розроблену систему на різних операційних системах без значних модифікацій коду та наявності пакетних

менеджерів таких як `pip` суттєво спрощує встановлення та управління залежностями проекту.

Для реалізації та навчання нейронної мережі обрано фреймворк `PyTorch`, високопродуктивну бібліотеку глибокого навчання з імперативним стилем програмування та динамічним обчислювальним графом [10]. `PyTorch` забезпечує ефективне GPU-прискорення через технологію `CUDA`, підтримує автоматичне диференціювання та надає зручний програмний інтерфейс для роботи з тензорами. Фреймворк надає гнучкі засоби для побудови мережі довільної складності включаючи згорткові, повнозв'язні шари, шари об'єднання та регуляризації, а також забезпечує зручний механізм ініціалізації вагових коефіцієнтів для стабільного навчання глибоких мереж.

Модуль `torchvision`, що входить до екосистеми `PyTorch` використовується для аугментації навчальних даних. Під час тренування застосовуються випадкові перетворення до тренувальних даних які включають обертання зображень на випадковий кут, горизонтальне зміщення, випадкове масштабування та випадкове горизонтальне відзеркалення. Такі перетворення суттєво збільшують ефективний обсяг навчальної вибірки та підвищують здатність моделі до узагальнення, що є критично важливим для роботи в реальних умовах із різноманітними ракурсами та умовами зйомки.

Для обробки зображень та відео використовується бібліотека `OpenCV` яка є відкритою бібліотекою комп'ютерного зору, що надає широкий набір функцій для захоплення та обробки відеопотоку, маніпуляцій із зображеннями, малювання анотацій та конвертації між колірними просторами [11]. Вона надає можливості читання та запис зображень і відеофайлів, зміну розмірів зображень облич до вхідного формату мережі, перетворення кольорових зображень у градації сірого, а також візуалізацію результатів розпізнавання шляхом нанесення обмежувальних рамок та текстових міток з назвами емоцій та відсотками впевненості. Для обробки відеопотоку з веб-камери `OpenCV` забезпечує захоплення кадрів у реальному часі та відображення анотованих кадрів. Бібліотека також підтримує завантаження та виконання моделей

нейронних мереж через модуль DNN, що використовується для роботи детектора облич у форматі Caffe.

Для експорту навченої моделі обрано формат ONNX (Open Neural Network Exchange) який є відкритим стандартом представлення моделей глибокого навчання, що забезпечує кросплатформенність між різними фреймворками та цільовими платформами [12]. На етапі розпізнавання емоцій прискорення моделі забезпечує бібліотека ONNX Runtime яка виконує оптимізоване завантаження моделей формату ONNX з підтримкою як CPU так і GPU через відповідні провайдери виконання. Застосування ONNX Runtime замість PyTorch на етапі розпізнавання дозволяє значно зменшити обсяг залежностей та підвищити швидкість обробки, що є суттєвою перевагою для розгортання системи у виробничому середовищі.

Для етапу визначення облич на зображенні реалізовано модуль детекції з підтримкою декількох бекендів, що забезпечує гнучкість та надійність системи. За замовчуванням використовується модель RFB-320 - легкий детектор облич на основі архітектури Receptive Field Block [8], який забезпечує оптимальне співвідношення швидкості та точності для роботи в реальному часі. У разі недоступності нейромережових моделей система автоматично переключасться на класичний метод Haar Cascade [9], вбудований у бібліотеку OpenCV, що забезпечує базову функціональність детекції облич навіть у обмежених умовах. Усі детектори повертають координати обмежувальних рамок виявлених облич, які потім обрізаються, масштабуються до потрібного розміру та подаються на вхід класифікатору емоцій.

Графічний інтерфейс користувача реалізовано з використанням стандартної бібліотеки tkinter яка входить до складу дистрибутива Python та не потребує встановлення додаткових залежностей. Інтерфейс побудований на основі віджетів ttk (themed tkinter), що забезпечує сучасний зовнішній вигляд на різних операційних системах. Головне вікно містить дві вкладки: вкладку розпізнавання емоцій та вкладку дослідження набору даних. Вкладка розпізнавання включає панель попереднього перегляду для відображення

зображень та відео, панель налаштувань моделі з можливістю вибору ONNX-моделі та пристрою обчислень, панель вибору джерела вхідних даних із підтримкою зображень, відеофайлів та веб-камери, а також панель результатів детекції. Обробка відео та потоку з веб-камери виконується в окремому потоці для забезпечення плавності роботи інтерфейсу.

Система також надає інтерфейс командного рядка (CLI) реалізований за допомогою стандартного модуля `argparse` який дозволяє виконувати навчання моделі та розпізнавання емоцій без графічного інтерфейсу. CLI підтримує дві основні команди: `train` для навчання моделі з можливістю налаштування гіперпараметрів таких як кількість епох, розмір батчу, швидкість навчання та метод навчання, та `recognize` для розпізнавання емоцій у трьох режимах: обробка окремих зображень, відеофайлів та потоку з веб-камери в реальному часі.

Для візуалізації процесу навчання використовується бібліотека `matplotlib` яка дозволяє будувати графіки функції втрат та точності класифікації на тренувальній і валідаційній вибірках протягом усіх епох навчання. Для числових обчислень та ефективних маніпуляцій з багатовимірними масивами даних використовується бібліотека `NumPy` яка забезпечує високопродуктивні операції над тензорами, обчислення функції `softmax`, а також операції з обмежувальними рамками під час детекції облич. Для індикації прогресу тривалих операцій, таких як навчання моделі та обробка відеофайлів використовується бібліотека `tqdm` яка відображає інтерактивний індикатор прогресу з оцінкою часу завершення.

Обраний набір інструментальних засобів забезпечує повний цикл розробки системи розпізнавання емоцій: від підготовки даних та навчання нейронної мережі до розгортання готового додатку з графічним інтерфейсом та підтримкою роботи в реальному часі. Поєднання `PyTorch` для навчання, `ONNX Runtime` для прискорення, `OpenCV` для обробки зображень та `tkinter` для побудови інтерфейсу створює ефективну та модульну архітектуру, кожен компонент якої може бути замінений або оновлений незалежно від інших.

Висновки до розділу 1

В ході виконання першого розділу було проведено аналіз предметної області розпізнавання емоцій обличчя людини. Розглянуто теоретичні засади класифікації та теорію базових емоційних станів. Визначено основні проблеми та виклики, що постають перед автоматизованими системами розпізнавання: варіативність умов зйомки, суб'єктивність анотації, дисбаланс класів у навчальних даних.

Проведено аналіз існуючих рішень для розпізнавання емоцій, як комерційних хмарних сервісів так і відкритих бібліотек. Порівняльний аналіз показав, що жодне з існуючих рішень не поєднує одночасно всі необхідні характеристики: локальну автономну роботу, можливість тренування власної моделі, підтримку всіх режимів обробки та графічний інтерфейс. На основі проведеного аналізу сформульовано постановку задачі дослідження та обрано засоби реалізації програмного продукту: мову Python, фреймворк PyTorch, бібліотеку OpenCV та формат моделей ONNX.

РОЗДІЛ 2. ПРОЕКТУВАННЯ АЛГОРИТМІВ ДЛЯ НАВЧАННЯ МЕРЕЖІ РОЗПІЗНАВАННЯ ЕМОЦІЙ

2.1 Визначення процесу навчання та архітектури моделі нейронної мережі

Процес розпізнавання емоцій за виразом обличчя з використанням нейронних мереж складається з кількох послідовних етапів. Спершу зображення подається на вхід системи де відбувається попередня обробка: перетворення у напівтоновий формат, масштабування до необхідного розміру та нормалізація інтенсивностей пікселів. Далі підготовлене зображення надходить до згорткової нейронної мережі, яка виконує класифікацію за 8 категоріями емоцій і в кінці на виході мережа повертає вектор ймовірностей де кожне значення відповідає впевненості моделі у приналежності вхідного зображення до конкретного класу емоцій. Для початку потрібно обрати архітектуру нейронної мережі яка підійде під дану задачу. На сьогодні існує низка архітектур згорткових нейронних мереж кожна з яких має свої характерні особливості, переваги та обмеження. Розглянемо найбільш відомі з них.

Архітектура AlexNet була запропонована у 2012 році та стала першою глибокою згортковою мережею яка здобула перемогу на змаганні ILSVRC із значним відривом від конкурентів - похибка top-5 становила 15,3% порівняно з 26,2% у найближчого суперника [18]. Мережа складається з 5 згорткових та 3 повнозв'язних шарів загальною кількістю близько 60 мільйонів параметрів. AlexNet ввела в практику ряд важливих технічних рішень: функцію активації ReLU замість сигмоїди, регуляризацію методом випадкового відсіювання (Dropout) та аугментацію навчальних даних. Проте відносно невелика глибина мережі та використання великих згорткових фільтрів (11×11 , 5×5) обмежують її здатність виявляти дрібні мімічні ознаки, що є критичним для розпізнавання емоцій.

Далі розглянемо архітектуру GoogLeNet, представлена у 2015 році яка запровадила концепцію модулів Inception - паралельних гілок згортки з фільтрами різного розміру (1×1 , 3×3 , 5×5) результати яких конкатенуються [19]. Такий підхід дозволяє мережі одночасно аналізувати ознаки різного масштабу. GoogLeNet здобула перемогу на ILSVRC-2014 при загальній кількості параметрів лише близько 6,8 мільйонів, у 12 разів менше ніж у AlexNet. Однак складна багатогілкова архітектура Inception суттєво ускладнює модифікацію мережі під конкретну задачу та вимагає ретельного балансування між гілками.

В свою чергу архітектура ResNet яка була запропонована у 2016 році вирішила фундаментальну проблему деградації глибоких мереж шляхом введення механізму залишкових з'єднань (skip connections) які дозволяють градієнтам обходити проміжні шари під час зворотного поширення похибки [20]. Завдяки цьому стало можливим ефективне навчання мереж глибиною до 152 шарів. ResNet-152 здобула перемогу на ILSVRC-2015 з похибкою top-5 лише 3,57%. Проте надмірна глибина ResNet є надлишковою для задачі розпізнавання емоцій на зображеннях невеликого розміру (64×64), оскільки карти ознак швидко зменшуються до субпіксельних розмірів, а велика кількість параметрів створює ризик перенавчання на обмежених наборах даних.

Сімейство мереж VGG представлене у 2015 році продемонструвало, що послідовне нарощування глибини мережі з використанням виключно малих фільтрів згортки 3×3 дозволяє суттєво підвищити точність класифікації [17]. Ключова ідея VGG полягає в тому, що два послідовних шари з фільтрами 3×3 мають таке ж ефективне рецептивне поле, як один шар з фільтром 5×5 , але при цьому використовують значно менше параметрів та мають додатковий нелінійний перетворювач між шарами. Архітектура VGG відрізняється простотою та регулярністю структури, що робить її зручною для модифікації під конкретні задачі.

Для задачі розпізнавання емоцій у даній роботі обрано архітектуру VGG13 через декілька міркувань, найголовніше з них це те що регулярна та прозора структура VGG (послідовні блоки з парними згортковими шарами)

дозволяє легко адаптувати мережу під специфічний розмір вхідних зображень (64×64) та кількість класів (8 емоцій), крім того глибина у 13 шарів є достатньою для виявлення ієрархічних мімічних ознак на зображеннях обличчя, але при цьому не є надмірною, на відміну від ResNet-50/152 де значна частина обчислювальних ресурсів витрачається на ознаки, які не є інформативними для задачі даного масштабу.

Архітектура реалізованої мережі VGG13 приймає на вхід напівтонове зображення розміром 64×64 пікселі (1 канал) та складається з п'яти згорткових блоків, за якими йдуть два повнозв'язних шари та вихідний класифікаційний шар. Операція згортки, що є основою кожного блоку обчислюється за формулою:

$$y(i, j) = \sum_m \sum_n x(i+m, j+n) \cdot w(m, n) + b \quad (2.1)$$

де $y(i, j)$ - значення результуючої карти ознак у позиції з координатами i та j ; $x(i+m, j+n)$ - значення вхідної карти ознак (або вхідного зображення для першого шару) у позиції, зміщеній на m та n відносно поточної; $w(m, n)$ - ваги згорткового фільтра (ядра) розміром $m \times n$, які є параметрами, що навчаються під час тренування мережі; b - зміщення (bias), додатковий параметр, що навчається та дозволяє зсувати функцію активації. Кожен згортковий блок містить два послідовних шари згортки з фільтрами 3×3 та функцією активації випрямленого лінійного блоку (ReLU), після яких застосовується операція максимального об'єднання (max pooling) з ядром 2×2 для зменшення просторових розмірів та шар випадкового відсіювання (Dropout2d) з імовірністю 0.25 для регуляризації. Кількість фільтрів зростає від 64 у першому блоці до 512 у четвертому та п'ятому блоках. Після п'яти етапів максимального об'єднання (max pooling) просторовий розмір карт ознак зменшується з 64×64 до 2×2 , що дає $512 \times 2 \times 2 = 2048$ значень. Ці значення передаються до повнозв'язної частини мережі, яка складається з двох шарів по 1024 нейрони з активацією ReLU та випадковим відсіюванням (Dropout) з імовірністю 0.5, та

вихідного лінійного шару з 8 виходами, що відповідають 8 категоріям емоцій. Детальна архітектура наведена у додатку А.1.

Реалізація архітектури мережі у фреймворку PyTorch виконана у вигляді класу VGG13 який успадковує базовий клас nn.Module. Нижче наведено фрагмент коду, що демонструє визначення першого згорткового блоку та повнозв'язної частини мережі:

```
class VGG13(nn.Module):
    def __init__(self, num_classes=8):
        super(VGG13, self).__init__()
        # First conv block: 2 conv layers with 64 filters
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Dropout2d(0.25),
        )
        # ... conv2-conv5 analogous with 128, 256, 512, 512 filters

        # Fully connected layers
        # After 5 max pooling: 64 -> 32 -> 16 -> 8 -> 4 -> 2
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(512 * 2 * 2, 1024),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(1024, 1024),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(1024, num_classes),
        )
```

Прямий прохід даних через мережу реалізовано у методі прямого проходу (forward), де зображення послідовно проходить через п'ять згорткових блоків та повнозв'язну частину:

```
def forward(self, x):
    x = self.conv1(x)
    x = self.conv2(x)
    x = self.conv3(x)
    x = self.conv4(x)
    x = self.conv5(x)
    x = self.fc(x)
    return x
```

Ініціалізація ваг мережі виконується за методом Каймінга для згорткових шарів та нормальним розподілом з нульовим середнім та стандартним відхиленням 0.01 для повнозв'язних шарів. Така ініціалізація забезпечує стабільність градієнтів на початкових етапах навчання та запобігає проблемі

зникаючих або вибухаючих градієнтів. Для оптимізації параметрів мережі використовується стохастичний градієнтний спуск (SGD) з моментом 0.9 та початковою швидкістю навчання 0.05. Розклад швидкості навчання реалізовано у вигляді ступінчатого зниження: в перші 20 епох початкова швидкість, епохи 20-39 мають половину початкової і від 40-ї епохи вже десята частина початкової. Такий розклад дозволяє мережі спочатку швидко наблизитись до оптимуму, а потім точніше налаштовувати ваги. Для запобігання перенавчанню застосовується механізм ранньої зупинки який відстежує точність на валідаційній вибірці та зупиняє навчання якщо покращення не спостерігається протягом заданої кількості епох (за замовчуванням 100). Модель з найкращою точністю на валідаційній вибірці зберігається у вигляді контрольної точки для подальшого аналізу.

2.2 Аналіз набору даних для проведення навчання

Для навчання моделі використовується набір даних FER+ який є розширеною версією оригінального набору FER2013 [3]. Оригінальний набір було створено шляхом збору зображень обличч з мережі Інтернет за допомогою емоційно забарвлених ключових слів та подальшої фільтрації людськими анотаторами. Набір містить 35887 напівтонових зображень розміром 48×48 пікселів кожне з яких відображає обличчя людини з певним емоційним виразом. Зображення збиралися в неконтрольованих умовах тому вони характеризуються значною варіативністю освітлення, ракурсів, вікових груп та етнічної приналежності зображених осіб, що робить цей набір даних наближеним до реальних умов застосування системи розпізнавання емоцій.

Ключовою відмінністю FER+ від оригінального FER2013 є підхід до анотації. Замість однієї мітки від одного анотатора, кожне зображення у FER+ розмічене 10 незалежними анотаторами через платформу краудсорсингу [4]. Кожен анотатор обирав одну з 10 категорій: neutral, happiness, surprise, sadness, anger, disgust, fear, contempt, unknown та NF (not a face). Такий підхід дозволяє отримати розподіл міток для кожного зображення, що відображає

неоднозначність та суб'єктивність сприйняття емоцій, наприклад: зображення з ледь помітною посмішкою може отримати голоси як за категорію happiness, так і за neutral, що точніше моделює реальну невизначеність людського сприйняття порівняно з однозначною жорсткою міткою. Саме ця властивість множинної анотації робить FER+ особливо цінним для навчання моделей здатних працювати з неоднозначними виразами обличчя.

Набір даних розділено на три підмножини: навчальну вибірку (Training) з приблизно 28000 зображень, валідаційну вибірку (PublicTest) з приблизно 3500 зображень та тестову вибірку (PrivateTest) з приблизно 3500 зображень. Таке розбиття успадковане від оригінального FER2013 та є стандартним для досліджень у цій галузі, що забезпечує можливість коректного порівняння результатів із іншими опублікованими роботами. Навчальна вибірка слугує для тренування вагових коефіцієнтів нейронної мережі, валідаційна для контролю процесу навчання, підбору гіперпараметрів та реалізації механізму ранньої зупинки, а тестова для фінальної незалежної оцінки якості навченої моделі на раніше не бачених даних. Вихідні дані зберігаються у форматі CSV-файлів де кожен рядок оригінального файлу fer2013.csv містить значення пікселів зображення у вигляді текстового рядка з 2304 числами (48×48), розділеними пробілами, а також позначку належності до однієї з трьох підмножин. Файл анотацій fer2013new.csv, що відповідає FER+ доповнює кожне зображення десятьма голосами анотаторів за відповідними категоріями емоцій. Перед початком навчання ці CSV-файли конвертуються у набір PNG-зображень за допомогою спеціалізованого скрипту генерації який зчитує текстові рядки пікселів, перетворює їх на двовимірні масиви розміром 48×48 , формує об'єкти зображень та зберігає їх без стиснення у відповідних каталогах для навчальної, валідаційної та тестової вибірок. Приклад даних даних з датасету FER+ можна побачити на рисунку 2.1.

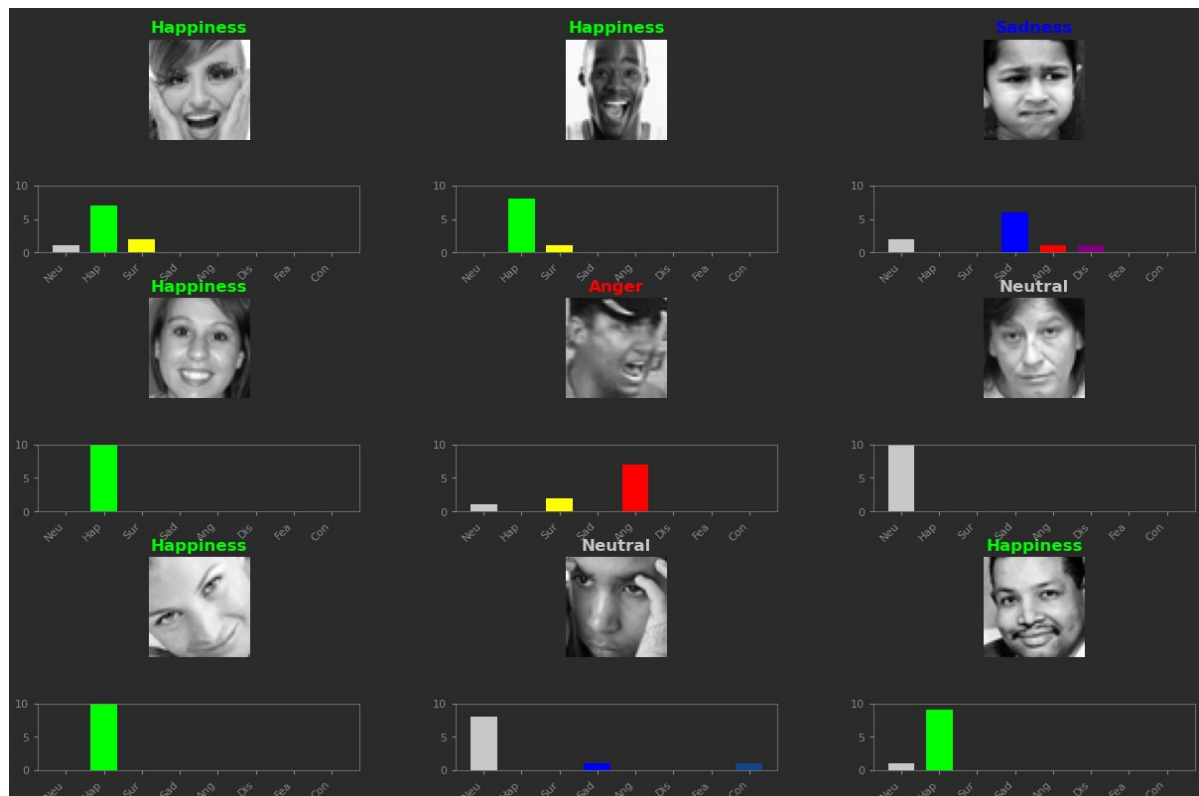


Рис. 2.1. Приклади зображень з набору FER+ та їх розподіл міток

Підготовка даних для навчання включає декілька послідовних етапів передобробки. Спершу зображення конвертуються у напівтоновий формат та масштабуються з оригінального розміру 48×48 до 64×64 пікселів шляхом білінійної інтерполяції. Збільшення розміру вхідних зображень дозволяє нейронній мережі зберігати достатню просторову роздільність карт ознак після кількох послідовних операцій об'єднання (pooling), кожна з яких зменшує просторові розміри вдвічі. Далі виконується гістограмне вирівнювання, що є процедурою яка нормалізує розподіл інтенсивностей пікселів та підвищує контрастність зображення. Алгоритм вирівнювання обчислює гістограму яскравості зображення, визначає кумулятивну функцію розподілу та відображає вихідні значення пікселів у нормалізований діапазон. Після цього виконується стандартизація: приведення значень до нульового середнього та одиничної дисперсії, що стабілізує процес навчання та прискорює збіжність оптимізатора. У випадках коли стандартне відхилення зображення є надзвичайно малим, стандартизація обмежується лише центруванням відносно середнього, що запобігає числовій нестабільності при діленні на наближені до нуля значення.

Для навчальної вибірки додатково застосовується аугментація даних для штучного розширення набору за допомогою геометричних перетворень. Конвеєр аугментації включає випадкові перетворення зображення серед яких поворот, горизонтальний та вертикальний зсув, а також масштабування в обмеженому діапазоні. Окрім цього застосовується випадкове горизонтальне відображення, що є природним перетворенням для зображень облич оскільки дзеркально відображене обличчя зберігає ту саму емоцію. Для валідаційної та тестової вибірок аугментація не застосовується, зображення проходять лише нормалізацію, що забезпечує стабільність та відтворюваність оцінки якості моделі. Окрім того, завантажувач даних перемішує навчальну вибірку перед кожною епохою, що забезпечує рівномірне представлення різних класів у межах кожного батчу та запобігає можливим артефактам пов'язаним із послідовним порядком зображень у наборі даних.

2.3 Реалізація алгоритмів для навчання моделі

Реалізація системи навчання побудована на модульній архітектурі де кожен компонент відповідає за окрему функціональну область. Основні модулі системи розміщено у директорії `src/` та включають: `model.py` - визначення архітектури мережі VGG13, `dataset.py` - завантаження та підготовка набору даних FER+, `train.py` - реалізація процесу навчання, `recognize.py` - модуль розпізнавання емоцій. Точкою входу для навчання через командний рядок є файл `main.py`.

Ключовим етапом підготовки даних є гістограмне вирівнювання з подальшою стандартизацією. Функція `histogram_equalization` обчислює кумулятивну функцію розподілу (CDF) інтенсивностей пікселів, нормалізує значення до діапазону $[-1, 1]$, а потім виконує z-стандартизацію приведення до нульового середнього та одиничної дисперсії. Ця процедура усуває відмінності в освітленні між зображеннями та підвищує контрастність мімічних ознак:

```
def histogram_equalization(img_array):  
    img_flat = img_array.flatten().astype(np.int32)  
    # Compute histogram
```

```

hist = np.bincount(img_flat, minlength=256)
# Compute CDF and normalize to [-1, 1]
cdf = hist.cumsum().astype(np.float64)
cdf = cdf * (2.0 / cdf[-1]) - 1.0
# Apply histogram equalization
img_eq = cdf[img_flat]
# Standardization (zero mean, unit variance)
mean = img_eq.mean()
std = img_eq.std()
if std > 1e-6:
    img_eq = (img_eq - mean) / std
else:
    img_eq = img_eq - mean
return img_eq.reshape(img_array.shape).astype(np.float32)

```

Перевірка стандартного відхилення на мінімальне значення запобігає помилці ділення на нуль у випадку абсолютно однорідних зображень. Математично процедура стандартизації описується формулою:

$$\hat{x} = \frac{(x - \mu)}{\sigma} \quad (2.2)$$

де $\hat{x}$ - нормалізоване значення пікселя яке подається на вхід нейронної мережі; x - значення пікселя після застосування гістограмного вирівнювання, що знаходиться у діапазоні $[-1, 1]$; μ - середнє арифметичне значення усіх пікселів зображення після гістограмного вирівнювання, що обчислюється як сума всіх значень, поділена на кількість пікселів; σ - стандартне відхилення значень пікселів яке характеризує ступінь розкиду інтенсивностей навколо середнього. Результуюче зображення має нульове середнє ($\mu = 0$) та одиничну дисперсію ($\sigma = 1$), що забезпечує однорідність масштабу вхідних даних для нейронної мережі та стабілізує процес оптимізації ваг.

Конфігурація аугментації для навчальної та валідаційної вибірок реалізована у функції `get_transforms`. Для навчальної вибірки застосовуються випадкові перетворення (поворот, зсув, масштабування), горизонтальне відображення, перетворення у тензор та нормалізація. Для валідаційної та тестової вибірок аугментація не застосовується, виконується лише перетворення у тензор та нормалізація:

```

def get_transforms(split='train'):
    if split == 'train':
        return transforms.Compose([
            transforms.RandomAffine(
                degrees=15,                # Random rotation

```

```

        translate=(0.1, 0.1), # Random translation
        scale=(0.9, 1.1),    # Random scaling
    ),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5], std=[0.5])
])
else:
    return transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5], std=[0.5])
    ])

```

Метод `_load_data` класу `FERPlusDataset` зчитує файл `label.csv` та виконує валідацію кожного запису. Для кожного зображення витягуються голоси 10 анотаторів за 8 основними категоріями емоцій (категорії `unknown`, `NF` а також записи з нульовою сумою голосів відкидаються). Тут виконується фільтрація шуму анотації: голоси з кількістю менше одного обнуляються, після чого вектор голосів нормалізується до суми 1:

```

def _load_data(self):
    label_file = os.path.join(self.split_dir, 'label.csv')
    with open(label_file, 'r') as f:
        reader = csv.reader(f)
        for row in reader:
            filename = row[0].strip()
            emotion_votes = [int(row[i]) for i in range(2, 12)]
            # Extract first 8 emotions (neutral through contempt)
            label_dist = np.array(emotion_votes[:8], dtype=np.float32)
            valid_votes = label_dist.sum()
            if valid_votes == 0:
                continue # All votes are unknown/NF
            # Outlier rejection: reset emotions with < 1 vote
            label_dist[label_dist < 1] = 0
            total = label_dist.sum()
            if total > 0:
                label_dist /= total # Normalize to probability
                majority_label = np.argmax(label_dist)
                self.data.append({
                    'image_path': image_path,
                    'label_dist': label_dist,
                    'majority_label': majority_label
                })

```

Метод `__getitem__` реалізує повний конвеєр підготовки одного зображення для подачі в нейронну мережу. Зображення завантажується у напівтоновому форматі, масштабується до 64×64 пікселів, проходить гістограмне вирівнювання, після чого застосовуються трансформації (аугментація для навчання або лише нормалізація для валідації). Цільовий

вектор формується методом `_process_target` відповідно до обраного режиму навчання:

```
def __getitem__(self, idx):
    sample = self.data[idx]
    image = Image.open(sample['image_path']).convert('L')
    image = image.resize((64, 64), Image.BILINEAR)
    img_array = np.array(image, dtype=np.uint8)
    # Apply histogram equalization
    img_preprocessed = histogram_equalization(img_array)
    # Convert back to PIL for torchvision transforms
    img_for_transform = Image.fromarray(
        ((img_preprocessed - img_preprocessed.min()) /
         (img_preprocessed.max() - img_preprocessed.min() + 1e-8)
         * 255).astype(np.uint8), mode='L')
    if self.transform:
        image = self.transform(img_for_transform)
    else:
        image = torch.FloatTensor(img_preprocessed).unsqueeze(0)
    target = self._process_target(sample['label_dist'])
    return {
        'image': image,
        'label_dist': torch.FloatTensor(sample['label_dist']),
        'majority_label': torch.LongTensor([sample['majority_label']])[0],
        'target': torch.FloatTensor(target)
    }
```

Метод повертає словник з чотирма полями: `image` - підготовлений тензор зображення, `label_dist` - оригінальний розподіл міток анотаторів, `majority_label` - мітка більшості (для оцінки точності), `target` - цільовий вектор для функції втрат (залежить від режиму навчання).

Метод `_process_target` визначає спосіб формування цільового вектора залежно від обраного методу навчання. У режимі `probability` (PLD) виконується стохастична вибірка однієї мітки з розподілу анотаторів та формується одnogарячий (one-hot) вектор. У режимі `crossentropy` (CEL) повертається повний нормалізований розподіл:

```
def _process_target(self, label_dist):
    if self.training_mode == 'crossentropy':
        # CEL: return full distribution for soft cross-entropy
        return label_dist.copy()
    elif self.training_mode == 'probability':
        # PLD: sample one label from distribution each time
        idx = np.random.choice(len(label_dist), p=label_dist)
        target = np.zeros_like(label_dist)
        target[idx] = 1.0
        return target
```

Критично важливою особливістю PLD є те, що вибірка виконується при кожному зверненні до зображення через `__getitem__`, тобто на кожній епісі

модель отримує потенційно різну мітку для одного й того ж зображення, що дозволяє неявно апроксимувати справжній розподіл емоцій.

SoftCrossEntropyLoss є спільною функцією втрат для обох методів навчання. На першому кроці обчислюється нормалізація виходів моделі за допомогою функції softmax, яка перетворює вектор довільних дійсних чисел у розподіл ймовірностей:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad (2.3)$$

де $\text{softmax}(z_j)$ - нормалізована ймовірність належності вхідного зображення до i -го класу емоцій значення якої знаходиться в діапазоні $(0, 1)$; z_j - вихід моделі для i -го класу (логіт); знаменник - сума експонент усіх логітів по всіх класах $j = 0, 1, \dots, 7$, яка слугує нормалізуючим множником та забезпечує, що сума усіх $\text{softmax}(z_i)$ дорівнює 1. Функція softmax перетворює вектор довільних дійсних чисел у коректний розподіл ймовірностей, зберігаючи порядок величин: більший логіт відповідає вищій ймовірності. Далі обчислюється м'яка перехресна ентропія між цільовим розподілом та логарифмом передбаченого розподілу:

$$L = -\frac{1}{N} \sum_n \sum_i t_{ni} \cdot \log(\text{softmax}(z_{ni})) \quad (2.4)$$

де L - значення функції втрат яке мережа намагається мінімізувати під час навчання; N - кількість зображень у поточному пакеті (batch), за яким усереднюється значення втрат; n - індекс зображення в пакеті, що приймає значення від 1 до N ; i - індекс класу емоцій, що приймає значення від 0 до 7 (всього 8 класів); t_{ni} - значення цільового вектора для n -го зображення та i -го класу, при методі PLD це 1 для вибіркового класу і 0 для решти, при методі CEL це нормалізована частка голосів анотаторів за i -й клас; z_{ni} - відповідний логіт моделі (вихід до застосування softmax) для n -го зображення та i -го класу; $\text{softmax}(z_{ni})$ - передбачена ймовірність i -го класу для n -го зображення, обчислена за формулою (2.3). Знак мінус перед виразом обумовлений тим, що логарифм ймовірності завжди від'ємний (оскільки ймовірність менша за 1), а

функція втрат повинна бути додатною. Реалізація функції втрат у PyTorch має наступний вигляд:

```
class SoftCrossEntropyLoss(nn.Module):
    def forward(self, outputs, targets):
        log_probs = torch.log_softmax(outputs, dim=1)
        loss = -(targets * log_probs).sum(dim=1).mean()
        return loss
```

Функція `train_epoch` реалізує один повний прохід по навчальній вибірці. Модель переводиться у режим `train()`, що активує відсіювання (Dropout) та пакетну нормалізацію (BatchNorm). Для кожного пакету виконується прямий прохід, обчислення втрат, зворотне поширення похибки та оновлення ваг. Точність обчислюється відносно міток більшості анотаторів для забезпечення узгодженості оцінки:

```
def train_epoch(model, dataloader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for batch in tqdm(dataloader, desc='Training'):
        images = batch['image'].to(device)
        majority_labels = batch['majority_label'].to(device)
        targets = batch['target'].to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += majority_labels.size(0)
        correct += predicted.eq(majority_labels).sum().item()
    epoch_loss = running_loss / len(dataloader)
    epoch_acc = 100. * correct / total
    return epoch_loss, epoch_acc
```

Точність на кожній епісі обчислюється як відношення кількості правильно класифікованих зображень до загальної кількості зображень у вибірці:

$$Accuracy = \frac{1}{N} \cdot \sum_n [\operatorname{argmax}(\hat{y}_n) = y_n] \cdot 100\% \quad (2.5)$$

де Accuracy - точність класифікації, виражена у відсотках; N - загальна кількість зображень у вибірці (навчальній, валідаційній або тестовій); n - індекс конкретного зображення; $\hat{y}_n$ - вектор передбачених ймовірностей для n-го зображення, що складається з 8 значень (по одному на кожен клас емоцій),

обчислених функцією softmax на виході мережі; $\text{argmax}(\hat{y}_n)$ - індекс класу з найбільшою передбаченою ймовірністю, тобто клас який модель вважає найбільш імовірним; y_n - мітка більшості анотаторів для n-го зображення, тобто клас емоцій, за який проголосувала найбільша кількість з 10 анотаторів; оператор $[\cdot]$ - дужки Айверсона, що повертають 1, якщо умова всередині виконується (передбачений клас збігається з міткою більшості), та 0 в іншому випадку. Точність завжди оцінюється відносно мітки більшості незалежно від методу навчання (PLD чи CEL), що забезпечує узгодженість та порівнюваність результатів між різними експериментами.

Функція validate оцінює якість моделі на валідаційній або тестовій вибірці. Модель переводиться у режим оцінювання eval(), обчислення градієнтів вимикається за допомогою torch.no_grad() для економії пам'яті та прискорення виведення (inference). Валідація завжди використовує повний розподіл міток (CEL) для забезпечення консистентної метрики незалежно від методу навчання:

```
def validate(model, dataloader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch in tqdm(dataloader, desc='Validation'):
            images = batch['image'].to(device)
            label_dist = batch['label_dist'].to(device)
            majority_labels = batch['majority_label'].to(device)
            outputs = model(images)
            loss = criterion(outputs, label_dist)
            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += majority_labels.size(0)
            correct += predicted.eq(majority_labels).sum().item()
    val_loss = running_loss / len(dataloader)
    val_acc = 100. * correct / total
    return val_loss, val_acc
```

Оптимізатор стохастичного градієнтного спуску (SGD) з моментом оновлює ваги мережі за наступними формулами:

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla L(\theta_t) \quad (2.6)$$

$$\theta_{t+1} = \theta_t - v_t \quad (2.7)$$

де v_t - вектор накопиченого моменту на кроці t який зберігає інформацію про напрямок попередніх оновлень ваг; γ - коефіцієнт моменту, що дорівнює 0.9 та визначає, яка частка попереднього моменту зберігається на кожному кроці (значення 0.9 означає, що 90% попереднього напрямку враховується); v_{t-1} - значення моменту на попередньому кроці; η - швидкість навчання (learning rate), що визначає розмір кроку оновлення ваг; $\nabla L(\theta_t)$ - градієнт функції втрат L по вектору параметрів θ на кроці t , який вказує напрямок найшвидшого зростання функції втрат; θ_t - вектор усіх параметрів мережі (ваги згорткових фільтрів, зміщення, ваги повнозв'язних шарів) на кроці t ; θ_{t+1} - оновлений вектор параметрів. Накопичення моменту дозволяє згладжувати оновлення ваг та прискорювати збіжність у напрямку стійкого градієнта, зменшуючи осциляції у напрямках з високою кривизною функції втрат. Швидкість навчання зменшується за ступінчастим розкладом за допомогою планувальника LambdaLR - на 20-й епосі швидкість зменшується вдвічі, на 40-й - вдесятеро. Механізм ранньої зупинки (early stopping) відстежує точність на валідаційній вибірці та зберігає контрольну точку (checkpoint) найкращої моделі:

```
# Learning rate schedule: step decay
def lr_lambda(epoch):
    if epoch < 20:
        return 1.0
    elif epoch < 40:
        return 0.5
    else:
        return 0.1

optimizer = optim.SGD(model.parameters(), lr=base_lr/batch_size,
momentum=0.9)
scheduler = optim.lr_scheduler.LambdaLR(optimizer, lr_lambda)

# Training loop with early stopping
best_val_acc = 0.0
patience_counter = 0
for epoch in range(1, args.epochs + 1):
    train_loss, train_acc = train_epoch(
        model, train_loader, criterion, optimizer, device)
    val_loss, val_acc = validate(
        model, val_loader, val_criterion, device)
    scheduler.step()
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        patience_counter = 0
        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
```

```

        'val_acc': val_acc, 'train_acc': train_acc,
    }, checkpoint_dir / 'best_model.pth')
else:
    patience_counter += 1
    if patience_counter >= args.early_stopping_patience:
        break # Early stopping triggered

```

Контрольна точка зберігає не лише ваги моделі (`model_state_dict`), але й стан оптимізатора (`optimizer_state_dict`), що дозволяє відновити навчання з точки зупинки. Після завершення навчального циклу найкраща модель завантажується з контрольної точки та оцінюється на тестовій вибірці.

Після завершення навчання найкраща модель експортується у формат ONNX [12] за допомогою функції `torch.onnx.export`. Модель переводиться у режим `eval()`, формується фіктивний вхідний тензор для трасування обчислювального графу, та виконується експорт з динамічними осями для розміру пакету:

```

model.eval()
dummy_input = torch.randn(1, 1, 64, 64).to(device)
torch.onnx.export(
    model, dummy_input, args.output_model,
    export_params=True,
    opset_version=18,
    do_constant_folding=True,
    input_names=['input'],
    output_names=['output'],
    dynamic_axes={
        'input': {0: 'batch_size'},
        'output': {0: 'batch_size'}
    }
)

```

Параметр згортання констант (`do_constant_folding`) увімкнено для оптимізації обчислювального графу. Динамічні осі дозволяють обробляти довільну кількість зображень за один прохід. Разом з моделлю зберігається JSON-файл метаданих:

```

metadata = {
    'model_type': 'VGG13',
    'training_method': args.method,
    'num_classes': 8,
    'emotion_labels': get_emotion_labels(),
    'train_acc': checkpoint['train_acc'],
    'val_acc': checkpoint['val_acc'],
    'test_acc': test_acc,
    'training_date': datetime.now().isoformat(),
    'hyperparameters': {
        'learning_rate': initial_lr,
        'batch_size': args.batch_size,
        'optimizer': 'SGD', 'momentum': 0.9
    }
}

```

```
    }  
}  
metadata_path = Path(args.output_model).with_suffix('.json')  
with open(metadata_path, 'w') as f:  
    json.dump(metadata, f, indent=2)
```

Метадані включають тип моделі, метод навчання, кількість класів, відображення класів емоцій, точність на всіх вибірках, дату навчання та гіперпараметри. Ця інформація використовується при завантаженні моделі для розпізнавання, дозволяючи перевірити сумісність та відтворити умови навчання.

2.4 Порівняння результатів навчання

Для оцінки ефективності реалізованих методів навчання було проведено серію експериментів з використанням обох стратегій PLD та CEL. Навчання виконувалось на GPU (NVIDIA CUDA) з пакетом (batch) розміром 64, оптимізатором SGD з моментом 0.9 та початковою швидкістю навчання 0.05. Максимальна кількість епох 500, з механізмом ранньої зупинки при відсутності покращення протягом 20 епох.

Метод імовірнісного вибіркового навчання (PLD) формує цільовий вектор шляхом випадкової вибірки однієї мітки з розподілу анотаторів для кожного зображення на кожній епосі. Завдяки стохастичній природі вибірки, протягом багатьох епох модель отримує різні мітки для кожного зображення, що дозволяє неявно навчитися розподілу емоцій.

Метод перехресної ентропійної функції втрат з розподілом міток (CEL) використовує повний нормалізований розподіл голосів анотаторів як цільовий вектор. На відміну від PLD, де цільовий вектор є одногарячим (one-hot), CEL подає на вхід функції втрат м'який розподіл ймовірностей, що безпосередньо кодує невизначеність анотації.

Результати навчання, отримані при виконанні даної роботи, наведено у таблиці 2.2.

Порівняння результатів навчання методами PLD та CEL

Метод	Train Acc	Val Acc	Test Acc	Епохи	Час навчання
PLD	81.15%	82.64%	~80.81%	100	36 хв
CEL	82.34%	81.78%	~81.98%	95	28 хв

Як видно з таблиці, обидва методи демонструють досить близькі результати, що свідчить про подібність їх ефективності для даної задачі. Метод CEL показав дещо вищу точність на тестовій вибірці (81,98%) порівняно з PLD (80,81%), водночас PLD продемонстрував кращу точність на валідаційній вибірці (82,64% проти 81,78%). Різниця між методами є незначною і становить приблизно 1%, що вказує на те, що обидва підходи до роботи з розподілами міток анотаторів є рівноцінними для задачі розпізнавання емоцій.



Рис. 2.2. Графіки навчання моделі методом PLD



Рис. 2.3. Графіки навчання моделі методом CEL

Аналіз графіків навчання показує, що обидва методи демонструють стабільну збіжність. Функція втрат монотонно зменшується протягом перших 30-40 епох після чого стабілізується. Точність на навчальній вибірці зростає швидше ніж на валідаційній, що є типовим для задач класифікації зображень. Ступінчасте зниження швидкості навчання на 20-й та 40-й епохах дозволяє моделі більш точно налаштувати ваги у завершальній фазі навчання. Варто зазначити, що метод CEL навчався швидше - 95 епох за 28 хвилин порівняно з 100 епохами за 36 хвилин у PLD, це пояснюється тим, що CEL використовує детерміністичний цільовий вектор, тоді як PLD вносить додатковий стохастичний шум через випадкову вибірку міток, що може уповільнювати збіжність. За результатами експериментів метод CEL обрано як основний для фінальної моделі, оскільки він забезпечує дещо вищу точність на тестовій вибірці та потребує менше часу на навчання.

Висновки до розділу 2

У другому розділі було описано проектування та реалізацію алгоритмів навчання нейронної мережі для розпізнавання емоцій обличчя. Проведено порівняльний аналіз відомих архітектур згорткових нейронних мереж та обґрунтовано вибір архітектури VGG13 як оптимальної для задачі класифікації емоцій на зображеннях розміром 64×64 пікселі. Визначено детальну архітектуру мережі VGG13, що складається з п'яти згорткових блоків та двох повнозв'язних шарів, з описом кожного компонента та параметрів регуляризації.

Проведено аналіз набору даних FER+, описано процес передобробки зображень (масштабування, гістограмне вирівнювання, стандартизація) та аугментації даних (перетворення, горизонтальне відображення, тощо).

Реалізовано два методи навчання: PLD та CEL, та описано модульну структуру коду, що включає компоненти для визначення моделі, завантаження даних, навчання та експорту в формат ONNX. Проведено порівняння результатів навчання яке показало, що обидва методи демонструють подібну

ефективність із різницею близько 1 відсоткового пункту, при цьому CEL показав дещо вищу точність на тестовій вибірці та швидше навчання.

РОЗДІЛ 3. ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1 Розробка графічної частини додатку для розпізнавання емоцій

Графічний інтерфейс користувача розроблено з використанням бібліотеки Tkinter яка є стандартною складовою мови Python та забезпечує кросплатформенну побудову віконних додатків на операційних системах Windows, macOS та Linux. Для стилізації елементів інтерфейсу використовується модуль ttk (themed Tkinter widgets) який надає сучасний зовнішній вигляд віджетів та підтримку системних тем оформлення. Додатково для роботи із зображеннями у вікні програми застосовується бібліотека Pillow, а для обробки відеопотоку та маніпуляцій із зображеннями бібліотека OpenCV [11]. Для візуалізації розподілу міток у вкладці дослідника набору даних використовується бібліотека matplotlib з бекендом TkAgg, що дозволяє вбудовувати графіки безпосередньо у вікно Tkinter.

Архітектура графічного інтерфейсу побудована на основі двох вкладок реалізованих за допомогою віджету ttk.Notebook. Перша вкладка «Recognition» містить основний функціонал розпізнавання емоцій з підтримкою трьох режимів роботи: обробка зображень, відеофайлів та потоку з веб-камери. Друга вкладка «Dataset Explorer» яка надає інструмент для візуального дослідження набору даних FER+ із відображенням випадкових зображень та розподілів міток анотаторів у вигляді стовпчастих діаграм.

Головний клас додатку EmotionRecognitionApp успадковує ttk.Frame та інкапсулює всю логіку інтерфейсу. При ініціалізації створюються змінні стану (шлях до моделі, обраний пристрій, тип детектора облич, поріг впевненості, режим відлагодження та ідентифікатор камери), формується структура вкладок та викликаються методи побудови окремих секцій інтерфейсу. Точкою входу для графічного інтерфейсу є файл gui.py, який створює головне вікно та розміщує в ньому екземпляр EmotionRecognitionApp:

```
def main():  
    root = tk.Tk()
```

```

root.title("Emotion Recognition")
root.geometry("960x700")
root.minsize(800, 600)
style = ttk.Style()
available_themes = style.theme_names()
if 'clam' in available_themes:
    style.theme_use('clam')
app = EmotionRecognitionApp(root)
app.pack(fill=tk.BOTH, expand=True)
root.mainloop()

```

Для забезпечення відгуку інтерфейсу під час тривалих операцій (обробка відео, потік з веб-камери) використовується багатопоточна модель на основі модуля `threading` стандартної бібліотеки Python. Кожна операція обробки запускається в окремому потоці, що дозволяє головному потоку Tkinter продовжувати обробку подій інтерфейсу, реагувати на натискання кнопок, переміщення вікна та інші дії користувача. Оновлення елементів інтерфейсу з робочого потоку виконується через метод `self.after(0, callback)`, який планує виконання функції у головному потоці, що є необхідною умовою для безпечної роботи з віджетами Tkinter:

```

def _start_processing(self):
    if not self._validate_inputs():
        return
    self.running = True
    self.start_btn.config(state=tk.DISABLED)
    self.stop_btn.config(state=tk.NORMAL)
    source = self.source_var.get()
    if source == "image":
        thread = threading.Thread(
            target=self._process_image, daemon=True)
    elif source == "video":
        thread = threading.Thread(
            target=self._process_video, daemon=True)
    else:
        thread = threading.Thread(
            target=self._process_webcam, daemon=True)
    thread.start()

```

Перед запуском обробки метод `_validate_inputs` перевіряє коректність усіх введених параметрів: наявність файлу моделі ONNX, наявність вхідного файлу (для режимів Image та Video), а також доступність файлів RFB-моделі при обраному нейромеревовому детекторі. У разі виявлення помилки виводиться діалогове вікно з описом проблеми, а обробка не запускається:

```

def _validate_inputs(self):
    if not self.model_path.get():
        messagebox.showerror("Error",
            "Please select an emotion model (.onnx)")
    return False

```

```

if not os.path.exists(self.model_path.get()):
    messagebox.showerror("Error",
        f"Model file not found: {self.model_path.get()}")
    return False
source = self.source_var.get()
if source in ("image", "video"):
    if not self.input_path.get():
        messagebox.showerror("Error",
            f"Please select an input {source}")
        return False
    if not os.path.exists(self.input_path.get()):
        messagebox.showerror("Error",
            f"Input file not found: {self.input_path.get()}")
        return False
if self.face_detector_var.get() == "rfb":
    if (self.rfb_model_path.get() and
        not os.path.exists(self.rfb_model_path.get())):
        messagebox.showerror("Error",
            f"RFB model not found")
        return False
return True

```

Клас `EmotionRecognizer` з модуля `recognize.py` є центральним компонентом конвеєру обробки зображень. Він інкапсулює завантаження моделі ONNX через бібліотеку `onnxruntime`, детекцію облич обраним детектором (RFB-320 за замовчуванням або Haar Cascade як резервний варіант), передобробку виявлених регіонів обличчя (конвертація у напівтоновий формат, масштабування до 64×64 пікселів, нормалізація до діапазону $[-1, 1]$), виведення моделі та анотацію результатів на зображенні. Ініціалізація розпізнавача виконується наступним чином:

```

class EmotionRecognizer:
    def __init__(self, model_path, device='cpu',
        confidence_threshold=0.5,
        face_detector='rfb', ...):
        # Load ONNX emotion model
        providers = ['CUDAExecutionProvider',
            'CPUExecutionProvider']
        if device == 'cuda' else ['CPUExecutionProvider']
        self.session = ort.InferenceSession(
            model_path, providers=providers)
        # Load face detector with fallback
        if face_detector == 'rfb':
            try:
                self.face_detector = RFBFaceDetector(...)
            except Exception as e:
                print(f"Falling back to Haar Cascade")
                self._load_haar_cascade()
        else:
            self._load_haar_cascade()

```

Метод `process_image` виконує повний конвеєр обробки одного кадру від детекції облич до отримання результатів класифікації, повертаючи список

кортежів (координати рамки, словник з емоцією, впевненістю та ймовірностями):

```
def process_image(self, image):
    faces = self.detect_faces(image)
    results = []
    for (x, y, w, h) in faces:
        face_img = image[y:y+h, x:x+w]
        emotion_data = self.predict_emotion(face_img)
        results.append(((x, y, w, h), emotion_data))
    return results
```

Метод `predict_emotion` виконує передобробку виділеного регіону обличчя (конвертація у напівтоновий формат, масштабування до 64×64, нормалізація) та запускає інференс через ONNX Runtime. Результат проходить через функцію `softmax` для отримання розподілу ймовірностей:

```
def predict_emotion(self, face_img):
    input_tensor = self.preprocess_face(face_img)
    outputs = self.session.run(
        None, {'input': input_tensor})[0]
    probs = self._softmax(outputs[0])
    emotion_id = np.argmax(probs)
    confidence = probs[emotion_id]
    return {
        'emotion': self.emotion_labels[emotion_id],
        'confidence': float(confidence),
        'probabilities': {
            self.emotion_labels[i]: float(probs[i])
            for i in range(len(probs))
        }
    }
```

Метод `annotate_image` наносить на зображення кольорові обмежувальні рамки, мітки емоцій та рівні впевненості. Для кожного виявленого обличчя малюється прямокутна рамка кольором відповідної емоції, фоновий прямокутник для тексту та мітка з назвою емоції та відсотком впевненості. У режимі відлагодження додатково виводяться ймовірності всіх класів:

```
def annotate_image(self, image, faces_emotions, debug=False):
    annotated = image.copy()
    if len(faces_emotions) == 0:
        cv2.putText(annotated, "No faces detected",
            (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
            1.0, (0, 0, 255), 2)
        return annotated
    for (x, y, w, h), emotion_data in faces_emotions:
        emotion = emotion_data['emotion']
        confidence = emotion_data['confidence']
        if confidence < self.confidence_threshold:
            continue
        color = self.emotion_colors.get(
            emotion, (255, 255, 255))
```

```

cv2.rectangle(annotated,
               (x, y), (x+w, y+h), color, 2)
label = f"{emotion} {confidence*100:.1f}%"
cv2.putText(annotated, label, (x, y - 5),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6,
            (255, 255, 255), 2)
return annotated

```

Вкладка «Dataset Explorer» реалізована у вигляді окремого класу DatasetExplorerTab у файлі dataset_explorer.py. Вкладка містить панель інструментів для вибору шляху до набору даних та кнопку оновлення, а також область відображення на основі вбудованого matplotlib.Figure. При натисканні кнопки «Refresh» функція load_dataset_entries сканує всі підмножини набору (train, valid, test), завантажує файли label.csv та формує список записів. З цього списку випадковим чином обирається 9 зображень, які відображаються у сітці 3×3 з фотографією обличчя, кольоровою міткою домінуючої емоції та стовпчастою діаграмою голосів анотаторів:

```

class DatasetExplorerTab(ttk.Frame):
    def _render_grid(self):
        self._fig.clf()
        k = min(9, len(self._entries))
        samples = random.sample(self._entries, k)
        gs = GridSpec(6, 3, figure=self._fig,
                      height_ratios=[3, 2, 3, 2, 3, 2])
        for idx, entry in enumerate(samples):
            row = idx // 3
            col = idx % 3
            ax_img = self._fig.add_subplot(gs[row*2, col])
            self._draw_image_cell(ax_img, entry)
            ax_bar = self._fig.add_subplot(gs[row*2+1, col])
            self._draw_bar_chart(ax_bar, entry)
        self._canvas_widget.draw()

```

3.2 Опис графічного інтерфейсу

Графічний інтерфейс вкладки «Recognition» розділено на дві основні панелі: ліву (область попереднього перегляду) та праву (панель налаштувань та керування). Ліва панель займає більшу частину вікна та містить компонент Canvas пікселів з темним фоном для відображення оброблених зображень або кадрів відеопотоку, рядок стану з інформацією про поточну операцію та показник кадрів на секунду, а також індикатор прогресу для відстеження обробки відеофайлів. Права панель має фіксовану ширину та містить послідовно розташовані секції налаштувань.

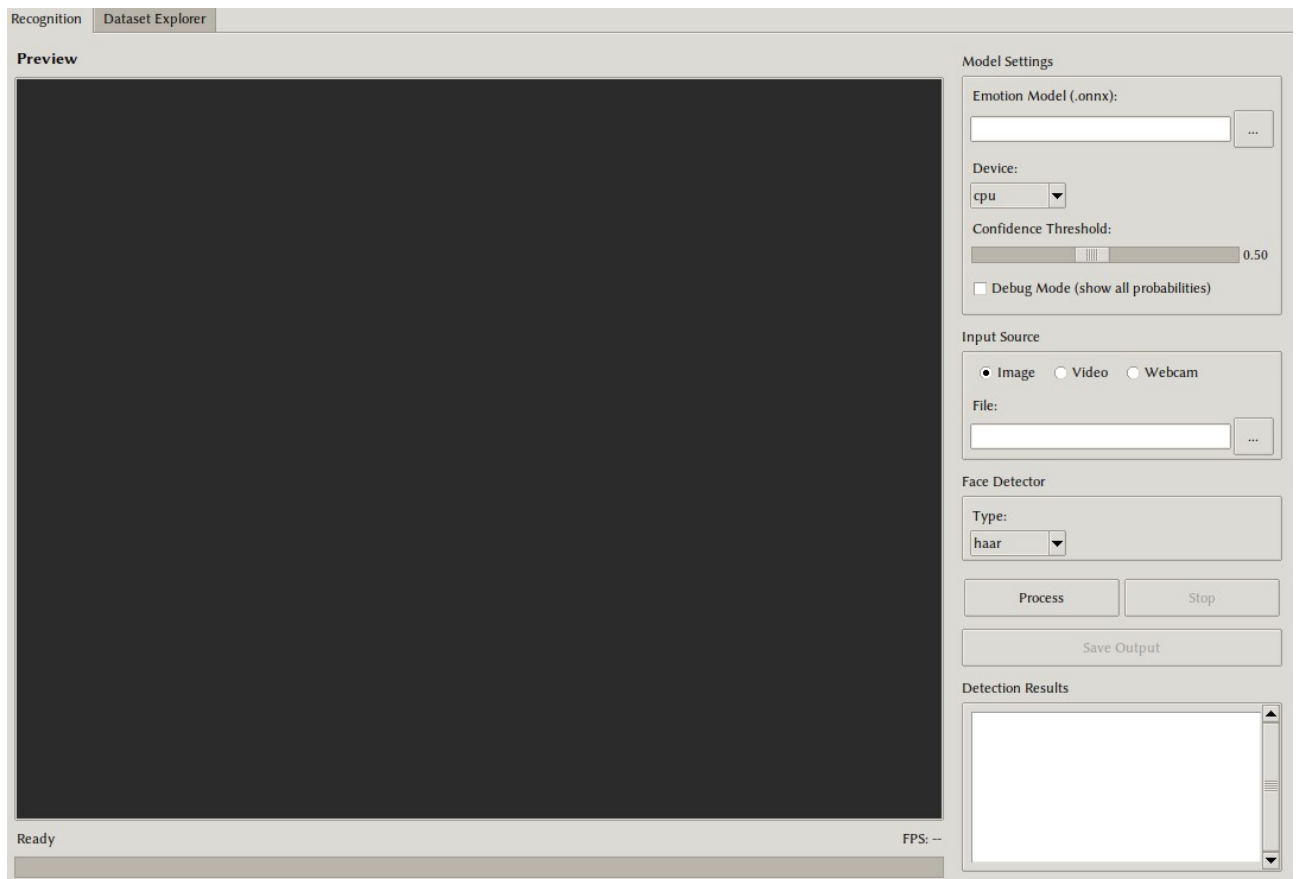


Рис. 3.1. Головне вікно додатку - вкладка Recognition

Секція «Model Settings» оформлена як `ttk.LabelFrame` і дозволяє користувачу обрати файл натренованої моделі у форматі ONNX через діалог вибору файлу, визначити пристрій обчислення (CPU або CUDA) через випадаючий список, налаштувати поріг впевненості за допомогою повзунка з діапазоном від 0.1 до 1.0 та числовим індикатором поточного значення, а також увімкнути режим відлагодження, який відображає повний розподіл ймовірностей для всіх 8 класів емоцій під кожним виявленим обличчям.

Секція «Input Source» містить три перемикачі для вибору режиму роботи: Image, Video та Webcam. Зміна вибору керується обробником `_on_source_change` який динамічно приховує або відображає відповідні елементи: при обранні Image або Video з'являється поле вибору файлу із підтримкою форматів JPG, PNG, BMP для зображень та MP4, AVI, MOV, MKV для відео, а при обранні Webcam - лічильник для вибору Camera ID від 0 до 10.

Секція «Face Detector» дозволяє обрати тип детектора обличч через випадаючий список: Haar або RFB. При виборі RFB обробник

`_on_detector_change` динамічно відображає додаткові поля для вказання шляхів до файлів `.caffemodel` та `.prototxt`. Якщо ці файли не вказані, використовуються шляхи за замовчуванням.

Секція керування містить три кнопки: «Process» для запуску обробки, «Stop» для зупинки поточної операції та «Save Output» для збереження результату. Стан кнопок змінюється динамічно: під час обробки кнопка Process стає неактивною, а Stop активною і після завершення обробки активується кнопка Save Output. При збереженні зображення відкривається діалог вибору формату, в той час як для відео файл зберігається автоматично з суфіксом «_emotion».

Секція «Detection Results» відображає текстові результати розпізнавання у вигляді текстового компонента з вертикальною прокруткою. Для кожного виявленого обличчя виводиться порядковий номер, розпізнана емоція та відсоток впевненості. При увімкненому режимі відлагодження додатково виводяться ймовірності всіх 8 класів у порядку спадання. Текстове поле знаходиться у стані disabled для запобігання редагуванню користувачем та тимчасово переключається у normal для оновлення вмісту.

Область попереднього перегляду реалізована методом `_display_image`, який конвертує зображення з формату BGR (OpenCV) у RGB, масштабує його під поточний розмір Canvas із збереженням пропорцій та центруванням, перетворює у `PhotoImage` через `Pillow` та розміщує на полотні:

```
def _display_image(self, image):
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    canvas_width = self.canvas.winfo_width()
    canvas_height = self.canvas.winfo_height()
    if canvas_width <= 1 or canvas_height <= 1:
        canvas_width, canvas_height = 640, 480
    img_h, img_w = image_rgb.shape[:2]
    scale = min(canvas_width / img_w,
                canvas_height / img_h)
    new_w = int(img_w * scale)
    new_h = int(img_h * scale)
    image_resized = cv2.resize(
        image_rgb, (new_w, new_h))
    pil_image = Image.fromarray(image_resized)
    photo = ImageTk.PhotoImage(pil_image)
    def update():
        self.canvas.delete("all")
        x = (canvas_width - new_w) // 2
```

```

y = (canvas_height - new_h) // 2
self.canvas.create_image(
    x, y, anchor=tk.NW, image=photo)
self.canvas._photo = photo
self.after(0, update)

```

Для кожного виявленого обличчя метод `annotate_image` класу `EmotionRecognizer` наносить кольорову обмежувальну рамку, фон для текстової мітки та саму мітку з назвою емоції та відсотком впевненості. Кольорове кодування використовує наступну схему: зелений (0, 255, 0) для радості, червоний (0, 0, 255) для гніву, синій (255, 0, 0) для суму, жовтий (0, 255, 255) для подиву, сірий (200, 200, 200) для нейтрального стану, фіолетовий (128, 0, 128) для огиди, помаранчевий (255, 165, 0) для страху та коричневий (139, 69, 19) для зневаги.

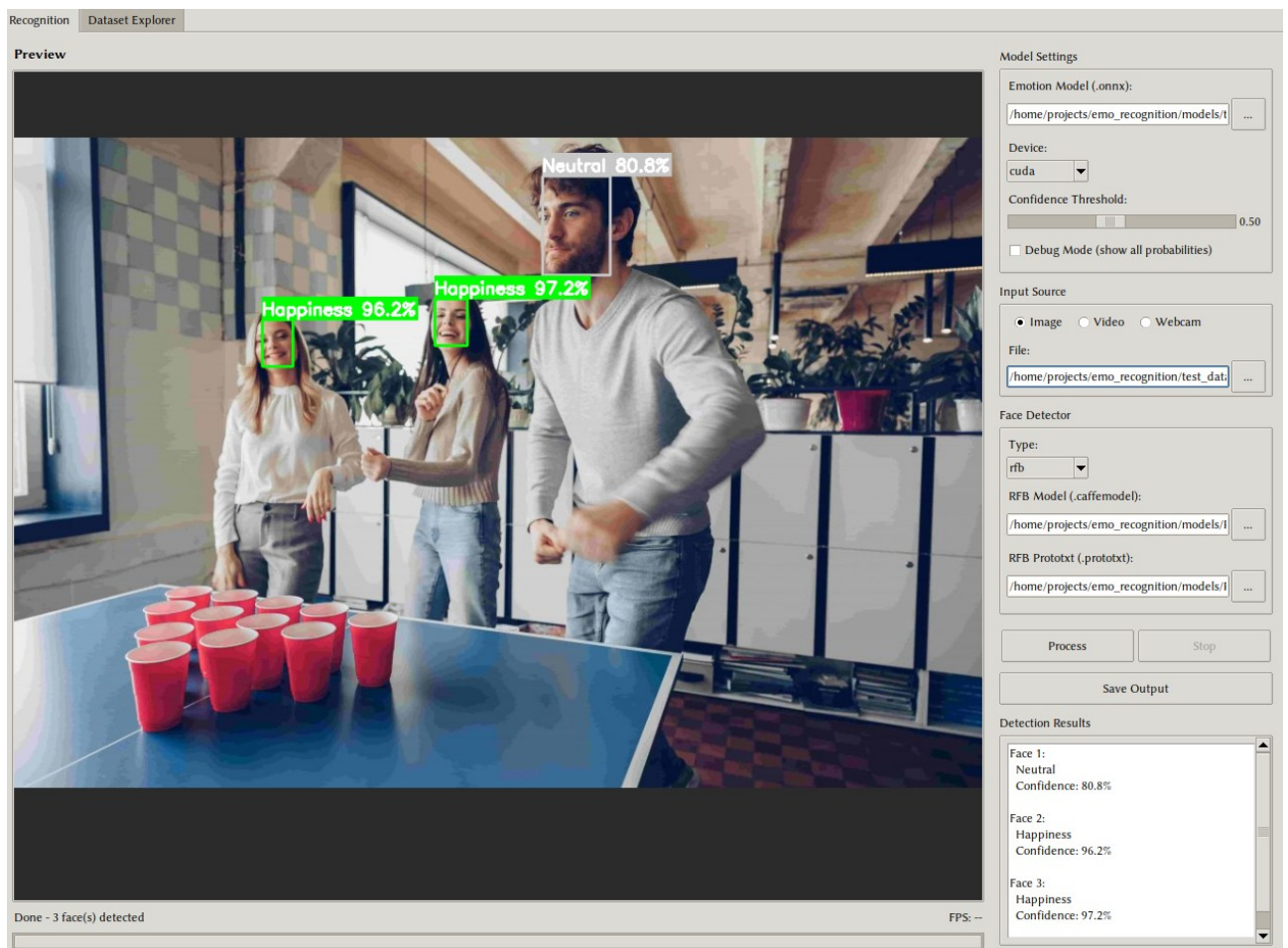


Рис. 3.2. Результат обробки зображення з декількома обличчями

Вкладка «Dataset Explorer» у верхній частині містить панель інструментів з полем для введення шляху до директорії набору даних (за замовчуванням «dataset»), кнопкою «Browse» для вибору директорії та кнопкою «Refresh» для

завантаження та відображення нової вибірки. Основну частину вкладки займає вбудований matplotlib.Figure з темним фоном, де у сітці 3×3 відображаються випадкові зображення. Для кожного зображення у верхній комірці відображається фотографія обличчя з кольоровим підписом домінуючої емоції, а у нижній стовпчаста діаграма голосів анотаторів із шкалою від 0 до 10, де кольори стовпчиків відповідають кольоровій схемі емоцій, а по осі X скорочені назви емоцій (Neu, Har, Sur, Sad, Ang, Dis, Fea, Con)



Рис. 3.3. Вкладка Dataset Explorer

3.3 Тестування роботи додатку

Тестування розробленого додатку проводилось у трьох основних режимах роботи: обробка окремих зображень, обробка відеофайлів та розпізнавання у реальному часі з веб-камери. Окрім перевірки основного функціоналу, було проведено тестування обробки помилкових ситуацій та граничних випадків. Для тестування використовувалась модель VGG13 натренована методом CEL та експортована у формат ONNX.

Для перевірки коректності роботи системи було використано тестові зображення із різними умовами: фронтальні фотографії з одним обличчям, групові фотографії з кількома обличчями, зображення з різним освітленням та ракурсами зйомки. Система коректно виявляє обличчя та класифікує емоції з відповідним рівнем впевненості. При обробці зображення з одним обличчям час відгуку з детектором RFB-320 становить менше 0.5 секунди на CPU. Для групових фотографій з 3-5 обличчями час обробки зростає пропорційно кількості виявлених облич, але залишається в межах секунди.

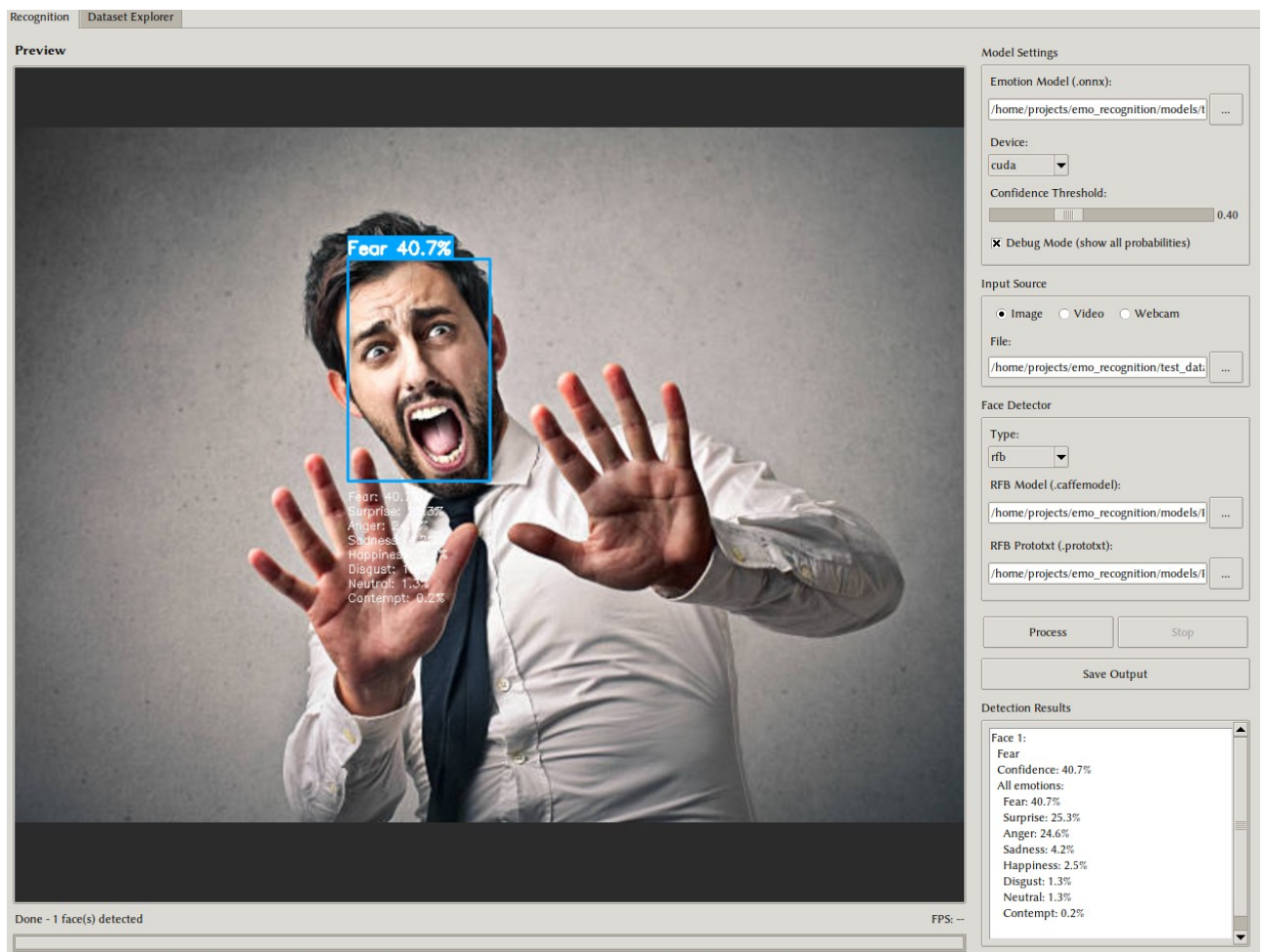


Рис. 3.4. Приклад розпізнавання емоцій на зображенні в режимі дебагу

Обробка відеофайлів виконується покадрово з відображенням прогресу на індикаторі та попереднім переглядом кожного п'ятого кадру в області Preview. Результатом є новий відеофайл з анотаціями, який автоматично зберігається з суфіксом «_emotion». При обробці відео довжиною 30 секунд (720p, 30 FPS) загальний час обробки з RFB-320 становить приблизно одну

хвилину. Користувач може зупинити обробку в будь-який момент натисканням кнопки «Stop», при цьому вже оброблена частина відео зберігається коректно в новий файл. Швидкість обробки на CPU з детектором RFB-320 складає приблизно 8-12 кадрів на секунду для відео роздільністю 720р.

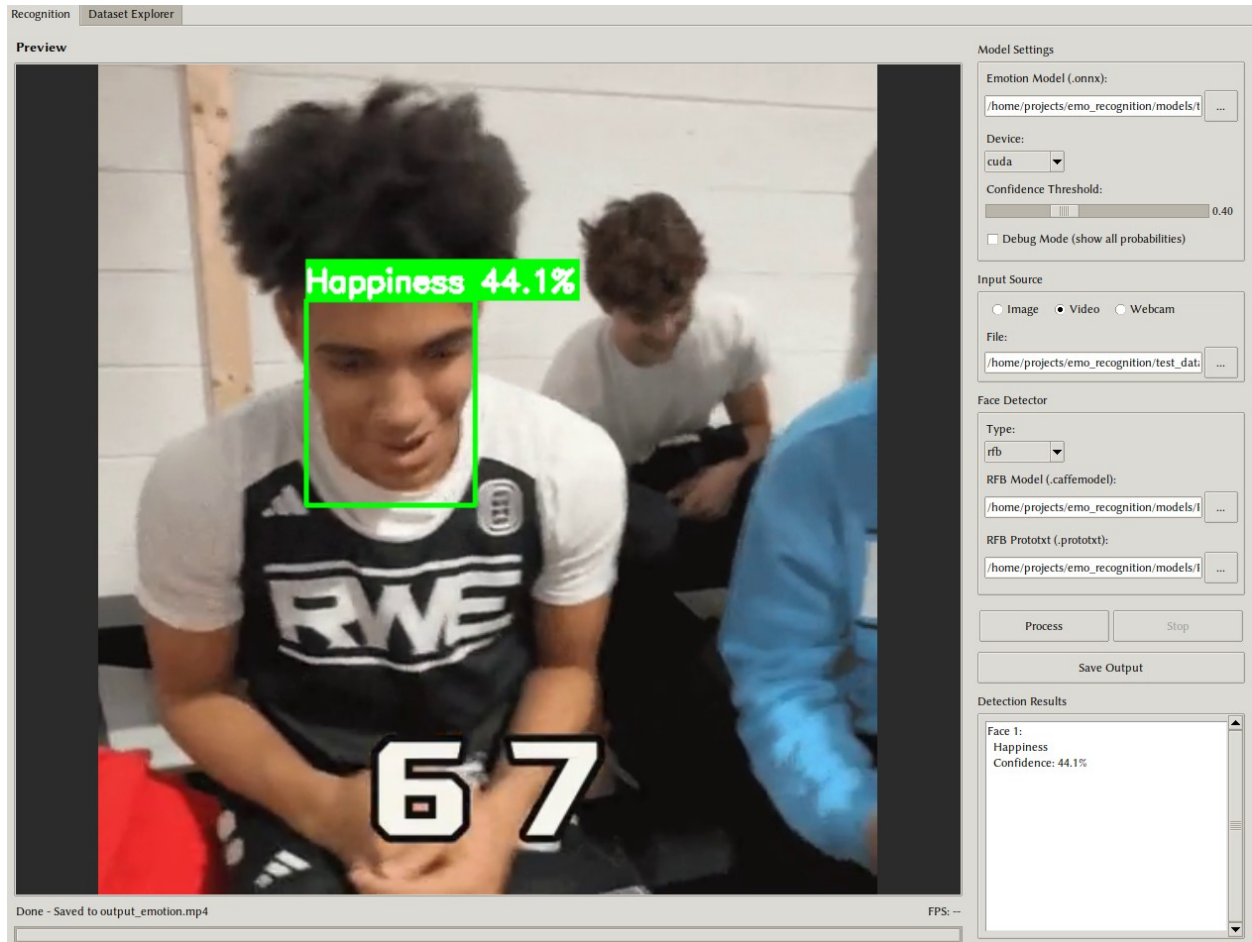


Рис. 3.5. Режим обробки відеофайлу

Режим роботи з веб-камерою забезпечує захоплення відеопотоку та розпізнавання емоцій у реальному часі. На кожному кадрі виконується детекція облич, класифікація та відображення результатів з обчисленням FPS. З детектором RFB-320 досягається швидкість 10-30 FPS на CPU, що забезпечує достатньо плавне оновлення для інтерактивного використання. Система коректно відстежує зміни виразу обличчя: при переході від нейтрального стану до посмішки мітка змінюється з «Neutral» на «Happiness» протягом 1-2 кадрів.

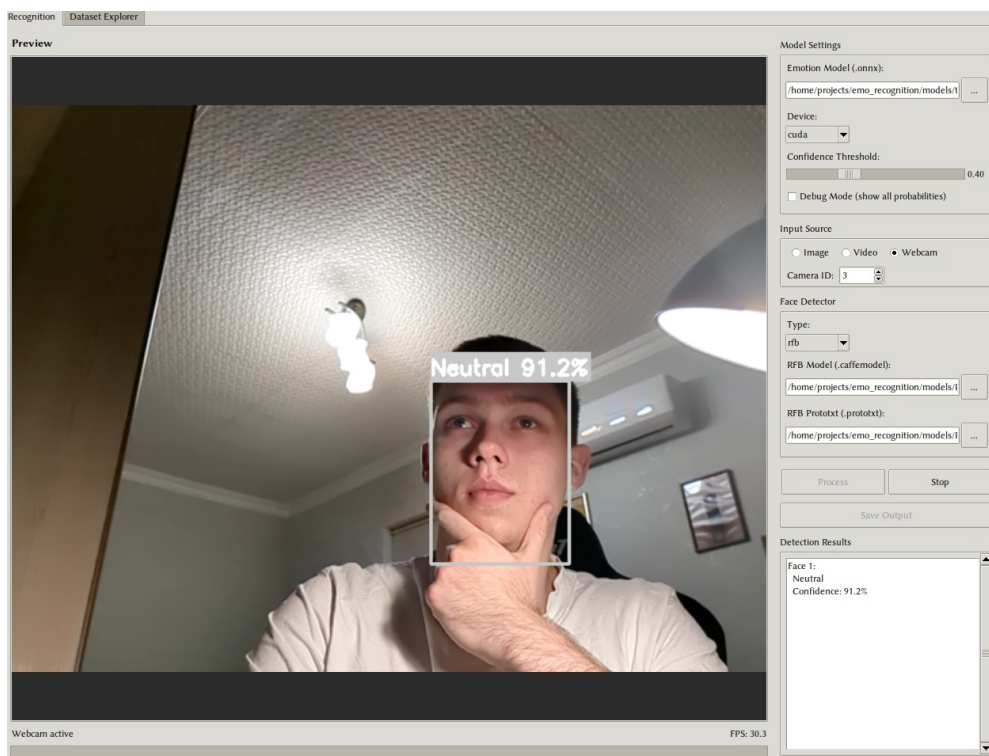


Рис. 3.6. Режим розпізнавання з веб-камери

Для забезпечення надійності додатку було проведено тестування поведінки системи при некоректних вхідних даних та помилкових діях користувача. При спробі запуску обробки без вибору файлу моделі ONNX система виводить діалогове вікно з повідомленням «Please select an emotion model (.onnx)» та не розпочинає обробку. Аналогічно при вказанні неіснуючого шляху до моделі виводиться повідомлення «Model file not found» з вказанням шляху, що дозволяє користувачу одразу зрозуміти причину помилки.

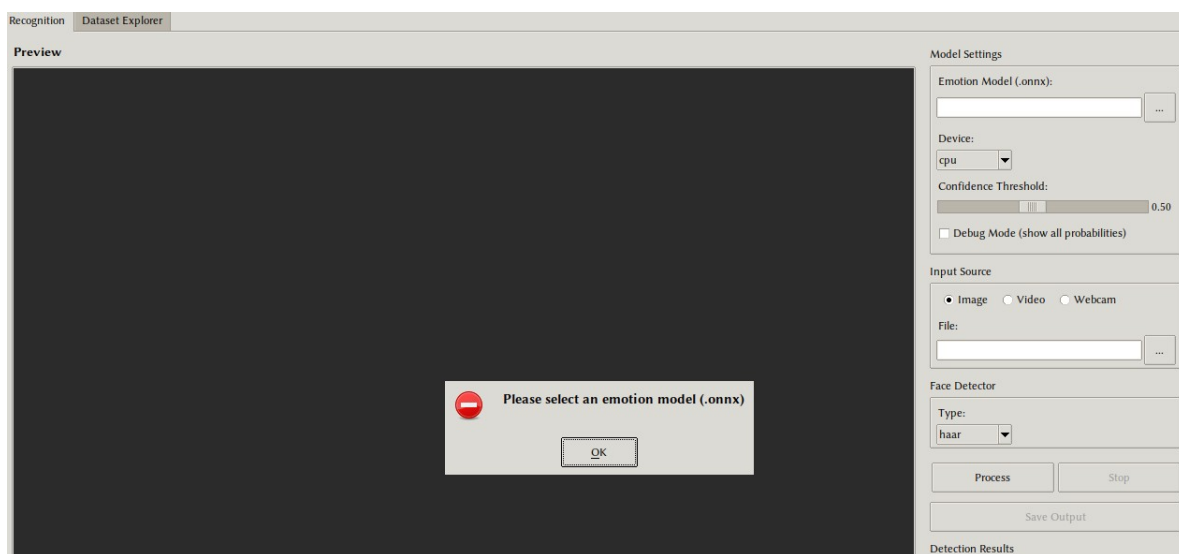


Рис. 3.7. Обробка помилки про відсутність файлу моделі

При спробі обробки зображення або відео без вибору вхідного файлу система повідомляє «Please select an input image» або «Please select an input video» залежно від обраного режиму. При вказанні файлу некоректного формату (наприклад, текстового файлу замість зображення) функція `cv2.imread` повертає `None`, що перехоплюється обробником винятків і користувачу виводиться повідомлення «Failed to load image» з деталями помилки.

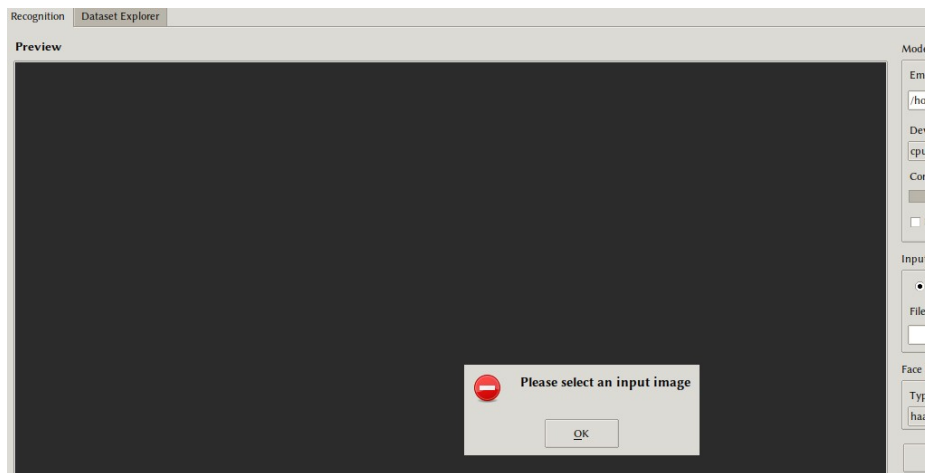


Рис. 3.8. Обробка помилки про відсутність вхідного файлу для обробки

При виборі детектора RFB з вказанням некоректних шляхів до файлів моделі система повідомляє «RFB model not found» та не запускає обробку, також якщо файли RFB-моделі не знайдені під час ініціалізації програми, система автоматично переключається на детектор Haar Cascade з виведенням попередження «Falling back to Haar Cascade detector» у консоль.

Також при спробі використання веб-камери з недоступним Camera ID система виводить повідомлення «Failed to open camera» з номером камери. При відключенні камери під час роботи цикл захоплення кадрів коректно завершується, а стан інтерфейсу повертається до початкового.

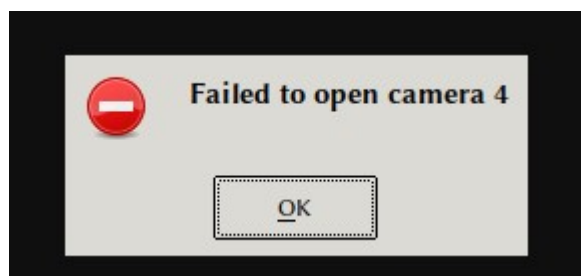
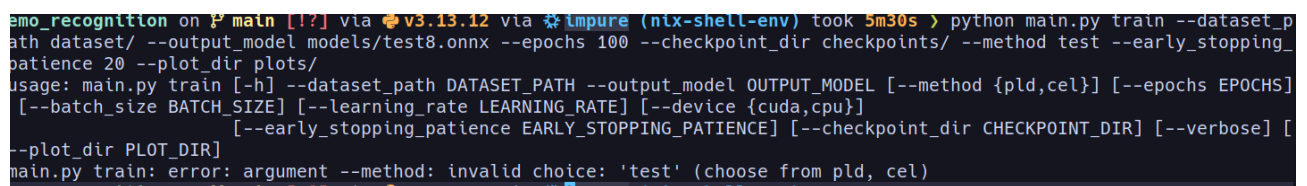


Рис. 3.9. Діалогове вікно помилки при відсутності камери за вказаним ID

Окрім графічного інтерфейсу, система підтримує роботу через командний рядок. CLI надає команди `train`, `image`, `video` та `webcam` з налаштуванням параметрів обробки:

```
python main.py image \  
  --model_path models/emotion_model.onnx \  
  --input photo.jpg \  
  --output result.jpg \  
  --face_detector rfb \  
  --confidence_threshold 0.5
```

При виклику CLI з некоректними аргументами модуль `argparse` виводить повідомлення про помилку із описом правильного використання.



```
emo_recognition on 🐍 main [!?] via 📶 v3.13.12 via 🌟 Lmpure (nix-shell-env) took 5m30s > python main.py train --dataset_p  
ath dataset/ --output_model models/test8.onnx --epochs 100 --checkpoint_dir checkpoints/ --method test --early_stopping_  
patience 20 --plot_dir plots/  
usage: main.py train [-h] --dataset_path DATASET_PATH --output_model OUTPUT_MODEL [--method {pld,cel}] [--epochs EPOCHS]  
  [--batch_size BATCH_SIZE] [--learning_rate LEARNING_RATE] [--device {cuda,cpu}]  
  [--early_stopping_patience EARLY_STOPPING_PATIENCE] [--checkpoint_dir CHECKPOINT_DIR] [--verbose] [  
  --plot_dir PLOT_DIR]  
main.py train: error: argument --method: invalid choice: 'test' (choose from pld, cel)
```

Рис. 3.10. Валідація командних аргументів

Тестування підтвердило стабільну роботу додатку у всіх режимах та коректну обробку помилкових ситуацій. Усі помилки супроводжуються інформативними повідомленнями, що дозволяють користувачу швидко ідентифікувати та усунути проблему.

Висновки до розділу 3

В третьому розділі було описано розробку графічної частини додатку для розпізнавання емоцій обличчя. Інтерфейс реалізовано на основі бібліотеки Tkinter з використанням модуля `ttk` для стилізації та віджету `Notebook` для організації двох вкладок та було застосовано багатопоточну модель обробки для забезпечення відгуку інтерфейсу під час тривалих операцій.

Детально описано структуру графічного інтерфейсу, що включає вкладку `Recognition` з п'ятьма секціями налаштувань (модель, джерело вхідних даних, детектор облич, кнопки керування, панель результатів) та вкладку `Dataset Explorer` для візуального дослідження набору даних FER+ з відображенням зображень та розподілів міток анотаторів.

Проведено тестування додатку у всіх режимах роботи, яке підтвердило коректність функціонування. Час обробки на CPU становить менше 1 секунди для окремих зображень та 10-30 FPS для потоку з веб-камери, що є достатнім для інтерактивного використання.

ВИСНОВКИ

У кваліфікаційній роботі було вирішено завдання розробки системи автоматичного розпізнавання емоцій обличчя людини з використанням нейронних мереж. В результаті проведених досліджень та практичної реалізації було створено програмний продукт, що забезпечує тренування моделі класифікації емоцій на краудсорсинговому наборі даних та розпізнавання емоційних станів на зображеннях, у відеофайлах та в режимі реального часу через веб-камеру.

Було проведено аналіз предметної області та порівняння існуючих рішень для розпізнавання емоцій який підтвердив відсутність системи, що поєднує локальну автономну роботу, можливість тренування власної моделі, підтримку всіх режимів обробки та зручний графічний інтерфейс. На основі порівняльного аналізу архітектур згорткових нейронних мереж було обґрунтовано вибір VGG13 як оптимальної для задачі класифікації емоцій на зображеннях невеликого розміру.

Реалізовано повний конвеєр навчання моделі на наборі даних FER+ із передобробкою, аугментацією та двома стратегіями роботи з краудсорсинговими розподілами міток. Дослідження методів PLD та CEL показало їх подібну ефективність із різницею близько одного відсоткового пункту, при цьому метод CEL досяг точності 81,98% на тестовій вибірці та продемонстрував швидшу збіжність.

Розроблений програмний додаток із графічним інтерфейсом забезпечує зручну роботу у трьох режимах та включає модуль детекції облич з автоматичним визначенням режиму роботи між нейромережевим детектором RFB-320 та класичним Haar Cascade. Тестування підтвердило стабільну роботу системи з часом обробки менше 0,5 секунди для окремих зображень та швидкістю 10-30 кадрів на секунду для потоку з веб-камери. Модель експортується у відкритий формат ONNX, що забезпечує її портативність та можливість інтеграції в інші програмні системи.

Отримані результати свідчать про ефективність застосування згорткових нейронних мереж у задачі розпізнавання емоцій обличчя, розроблена система має практичну цінність для моніторингу емоційного стану в освітньому процесі, аналізу реакцій клієнтів та досліджень у галузі психології. Перспективними напрямками подальшого розвитку щодо цього завдання є застосування архітектур на основі механізму уваги для підвищення точності та оптимізація моделі для більш широкого спектру емоцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ekman P., Friesen W. V. Constants across cultures in the face and emotion. *Journal of Personality and Social Psychology*. 1971. Vol. 17, No. 2. P. 124-129. DOI: <https://doi.org/10.1037/h0030377>
2. Ekman P. An argument for basic emotions. *Cognition and Emotion*. 1992. Vol. 6, No. 3-4. P. 169–200. DOI: <https://doi.org/10.1080/02699939208411068>
3. Goodfellow I. J., Erhan D., Carrier P. L. et al. Challenges in Representation Learning: A Report on Three Machine Learning Contests. *Neural Networks*. 2015. Vol. 64. P. 59-63. DOI: <https://doi.org/10.1016/j.neunet.2014.09.005>
4. Barsoum E., Zhang C., Canton Ferrer C., Zhang Z. Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution. *Proceedings of the 18th ACM International Conference on Multimodal Interaction (ICMI '16)*. 2016. P. 279-283. DOI: <https://doi.org/10.1145/2993148.2993165>
5. Wang K., Peng X., Yang J., Lu S., Qiao Y. Suppressing Uncertainties for Large-Scale Facial Expression Recognition. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2020)*. 2020. P. 6897-6906. DOI: <https://doi.org/10.1109/CVPR42600.2020.00693>
6. Li S., Deng W. Deep Facial Expression Recognition: A Survey. *IEEE Transactions on Affective Computing*. 2022. Vol. 13, No. 3. P. 1195-1215. DOI: <https://doi.org/10.1109/TAFFC.2020.2981446>
7. Sajjad M., Ullah F. U. M., Ullah M. et al. A comprehensive survey on deep facial expression recognition: challenges, applications, and future guidelines. *Alexandria Engineering Journal*. 2023. Vol. 68. P. 817-845. DOI: <https://doi.org/10.1016/j.aej.2023.01.017>
8. Liu S., Huang D., Wang Y. Receptive Field Block Net for Accurate and Fast Object Detection. *Computer Vision - ECCV 2018: Lecture Notes in Computer*

Science, vol. 11215. Springer, Cham. 2018. P. 385-400. DOI: https://doi.org/10.1007/978-3-030-01252-6_24

9. Viola P., Jones M. J. Robust Real-Time Face Detection. International Journal of Computer Vision. 2004. Vol. 57, No. 2. P. 137-154. DOI: <https://doi.org/10.1023/B:VISI.0000013087.49260.fb>

10. PyTorch: An Imperative Style, High-Performance Deep Learning Library [Электронный ресурс]. URL: <https://pytorch.org/>

11. OpenCV: Open Source Computer Vision Library [Электронный ресурс]. URL: <https://docs.opencv.org/>

12. ONNX: Open Neural Network Exchange [Электронный ресурс]. 2019. URL: <https://github.com/onnx/onnx>

13. Amazon Rekognition - Emotion Detection [Электронный ресурс]. URL: https://docs.aws.amazon.com/rekognition/latest/APIReference/API_Emotion.html

14. Face++ Emotion Recognition API [Электронный ресурс]. URL: <https://www.faceplusplus.com/emotion-recognition/>

15. Affectiva Facial Coding SDK (iMotions) [Электронный ресурс]. URL: <https://imotions.com/products/affectiva-facial-coding-sdk/>

16. DeepFace: A Lightweight Face Recognition and Facial Attribute Analysis Library for Python [Электронный ресурс]. URL: <https://github.com/serengil/deepface>

17. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015). 2015. arXiv: 1409.1556. URL: <https://arxiv.org/abs/1409.1556>

18. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing

Systems 25 (NeurIPS 2012). 2012. P. 1097-1105. DOI: <https://doi.org/10.1145/3065386>

19. Szegedy C., Liu W., Jia Y. et al. Going Deeper with Convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015). 2015. P. 1-9. DOI: <https://doi.org/10.1109/CVPR.2015.7298594>

20. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016). 2016. P. 770-778. DOI: <https://doi.org/10.1109/CVPR.2016.90>

ДОДАТКИ

ДОДАТОК А

Таблиця А.1

Архітектура мережі VGG13FER

Блок	Шари	Фільтри / Нейрони	Вихід
Conv Block 1	2×Conv3×3 + MaxPool + Dropout(0.25)	64	32×32
Conv Block 2	2×Conv3×3 + MaxPool + Dropout(0.25)	128	16×16
Conv Block 3	2×Conv3×3 + MaxPool + Dropout(0.25)	256	8×8
Conv Block 4	2×Conv3×3 + MaxPool + Dropout(0.25)	512	4×4
Conv Block 5	2×Conv3×3 + MaxPool + Dropout(0.25)	512	2×2
FC Layer 1	Linear + ReLU + Dropout(0.5)	1024	1024
FC Layer 2	Linear + ReLU + Dropout(0.5)	1024	1024
Output	Linear	8	8