МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра прикладної математики та моделювання складних систем

«До захисту допущено»

Завідувач кафедри

_____ Ігор КОПЛИК

___ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 133 Прикладна математика,

освітньо-професійної програми Наука про дані та моделювання складних систем

на тему: Вивчення існуючих моделей та нейромереж для розпізнавання образів з мінімізацією зусиль на додаткове тренування. Дослідження систем відеорозпізнавання, які не потребують навчання.

Здобувача групи ПМ-01 Івашини Андрія Володимировича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Андрій ІВАШИНА

Керівник доцент, кандидат наук, Аліна ДВОРНИЧЕНКО _____

Суми – 2024

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет **електроніки та інформаційних технологій**
Кафедра **прикладної математики та моделювання складних систем**
Рівень вищої освіти **перший**
Галузь знань **11 «Математика та статистика»**
Спеціальність **113 «Прикладна математика»**
Освітня програма **освітньо-професійна** «Наука про дані та моделювання складних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМтаМСС
Ігор КОПЛИК _____
«___» _____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧЕВІ ВИЩОЇ ОСВІТИ

Івашині Андрію Володимировичу

1. Тема роботи Вивчення існуючих моделей та нейромереж для розпізнавання образів з мінімізацією зусиль на додаткове тренування. Дослідження систем відеорозпізнавання, які не потребують навчання.

Керівник роботи доцент, кандидат наук, Аліна ДВОРНИЧЕНКО

затверджено наказом по факультету ЕлІТ від « 05 » квітня 2024 р. № 0349-VI

2. Термін подання роботи здобувачем « 10 » червня 2024 р.

3.Вихідні дані до роботи: набір даних «Тварини з атрибутами 2» (AwA2), що містить систематизовані зображеннями різних видів тварин та їх атрибути.

4. Зміст розрахунково-пояснювальної записки (перелік питань, для розроблення):

попередня обробка набору зображень, вибір гіперпараметрів, метрики оцінки моделей, програмна реалізація, аналіз результатів.

5. Перелік графічного матеріалу:

графік функцій активації, схема роботи нейронної мережі, візуалізація набору даних, зображення метрик, інфографіка результатів дослідження та їх аналізу.

6. Консультанти проєкту (роботи) із зазначенням розділів проєкту, що їх стосується

Розділ	Ім'я ПРІЗВИЩЕ та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання « 08 » квітня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання роботи	Примітка
1	Аналітичний огляд проблеми, збір необхідних даних та вибір стратегії дій	21.04.2024 р.	Дані у вигляді зображень
2	Використання вибраних методів у процесі машинного навчання	26.05.2024 р.	
3	Оцінка результатів проведеного дослідження та їх аналіз	02.06.2024 р.	Порівняльні діаграми

Здобувач вищої освіти _____ Івашина А. В.

Керівник роботи _____ Дворниченко А. В.

АНОТАЦІЯ

Кваліфікаційна робота: 72 с., 30 рисунків, 30 формул, 18 джерел.

Мета роботи: розглянути існуючі моделі та нейромережі для розпізнавання графічних образів та вилучення їх ознак для подальшої класифікації зображень з мінімізацією зусиль на додаткове тренування.

Об'єкт дослідження: сучасні нейронні мережі та методи розпізнавання образів.

Предмет дослідження: програмна реалізація згорткової нейронної мережі та застосування підходу навчання «з нуля» на реальному наборі систематизованих зображень.

Методи навчання: літературний аналіз, експериментальне моделювання, аналіз даних, апробація програмної реалізації.

У цій роботі досліджено методи розпізнавання та класифікації зображень за допомогою згорткових нейронних мереж (CNN) та навчання «з нуля» (ZSL). Розглянуто вибір активаційних функцій, оптимізаторів і метрик для оцінки моделей з метою досягнення найкращих результатів. Проведено підготовку та попередню обробку даних, машинне втілення CNN і ZSL, а також аналіз результатів експериментів. Показано ефективність обраних підходів для розпізнавання нових класів зображень без додаткового тренування. Виявлено переваги та недоліки кожного методу, що дозволяє окреслити напрямки для подальших досліджень і вдосконалення розглянутих моделей.

Умовні позначення:

ЗНМ – згорткова нейронна мережа

CNN – convolutional neural network

ZSL – zero-shot learning

AwA2 – animals with attributes 2

Ключові слова: ГЛИБОКЕ НАВЧАННЯ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, НАВЧАННЯ «З НУЛЯ», РОЗПІЗНАВАННЯ ОБРАЗІВ, КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, АНАЛІЗ ДАНИХ, АКТИВАЦІЙНА ФУНКЦІЯ, ОПТИМІЗАТОР, МЕТРИКА, PYTHON.

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій, нейронні мережі стають основою багатьох інноваційних рішень. Однією з найбільш значущих архітектур нейронних мереж є згорткова нейронна мережа, яка демонструє високу ефективність у задачах комп'ютерного зору, розпізнавання образів та інших сферах. ЗНМ стала важливим інструментом для аналізу візуальних даних, дозволяючи досягати нових висот в автоматизації та покращенні якості життя.

Інший важливий напрямок розвитку машинних технологій – навчання «з нуля», що дозволяє моделям класифікувати нові, невідомі класи, використовуючи мінімум додаткових даних. Це має величезний потенціал для багатьох галузей, де доступ до великого обсягу навчальних даних обмежений або неможливий.

Актуальність даного дослідження полягає в комплексному підході до аналізу і порівняння ефективності різних методів навчання нейронних мереж. Воно включає в себе детальний аналітичний огляд існуючих методів, вибір відповідного набору даних, попередню обробку даних та практичне втілення згорткової нейронної мережі та методу навчання «з нуля». Результати експерименту дозволять не тільки зрозуміти переваги та обмеження кожного з підходів, але й визначити найефективніші стратегії для їх застосування у реальних задачах.

Таким чином, дана робота сприяє глибшому розумінню сучасних методів машинного навчання, їх взаємодії та можливостей оптимізації, що є важливим кроком до подальшого розвитку інтелектуальних систем.

РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД ЗАВДАННЯ

У цьому розділі буде розглянуто принципи роботи ЗНМ та способу навчання «з нуля», їх архітектуру, налаштування навчання та оцінку результатів. Також будуть висвітлені основні переваги та недоліки цих методів.

1.1 Згорткова нейронна мережа (Convolutional Neural Network)

Згорткові нейронні мережі є потужним інструментом у сфері машинного навчання, який застосовується для аналізу візуальних даних, таких як зображення та відео. ЗНМ відрізняються від традиційних нейронних мереж тим, що вони можуть автоматично виявляти ієрархічні шаблони в даних за допомогою великої кількості шарів, що робить їх особливо ефективними для задач розпізнавання образів і класифікації. Кожен шар відіграє важливу роль у процесі обробки та аналізу даних, забезпечуючи можливість моделі навчатися та робити точні передбачення.

У цьому розділі я розгляну архітектурну будову сучасних згорткових нейронних мереж.

1.1.1 Згортковий шар (Convolutional Layer)

Згортковий шар є основним компонентом згорткових нейронних мереж, який відповідає за виявлення локальних ознак у вхідних даних. Основною ідеєю цього шару є застосування фільтрів для виявлення різних ознак, таких як краї, текстури або інші локальні структури у вхідному зображенні. Фільтри, які є невеликими матрицями, ковзають по всьому зображенню, застосовуючи операцію згортки до кожної підматриці вхідних даних. Розмір фільтра визначає масштаб виявлених ознак, наприклад, 3x3, 5x5 або 7x7.

Операція згортки включає накладення фільтра на вхідне зображення та обчислення скалярного добутку значень пікселів в області накладання та значень фільтра. Результат цієї операції утворює нове значення пікселя у вихідній карті ознак. Цей процес повторюється для кожної позиції фільтра по

всьому зображенню, створюючи карту ознак, яка представляє виявлені ознаки в різних позиціях. Кількість карт ознак відповідає кількості використаних фільтрів у згортковому шарі.

Параметри, які впливають на функціонування згорткового шару, включають крок, який визначає, наскільки сильно зміщується фільтр по зображенню після кожної операції згортки, та доповнення, що додає рамку з нульовими значеннями навколо вхідного зображення для контролю розмірності вихідних даних. Існують різні види доповнення, такі як «дійсне» доповнення, яке не додає додаткових пікселів і зменшує розмірність виходу, та «незмінне» доповнення, яке забезпечує, щоб вихідна карта ознак мала такий самий розмір, як і вхідне зображення.

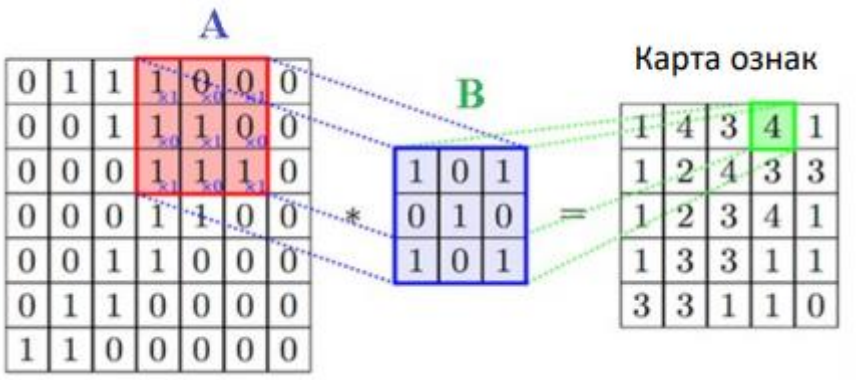

Рис. 1.1 – Візуалізація операції згортки (A – вхідне зображення, B – фільтр)

Згортковий шар виявляє локальні узори у вхідних даних, такі як краї, текстури або кути, які можуть бути комбіновані у наступних шарах для виявлення більш складних ознак і структур у даних. Наприклад, при застосуванні 3x3 фільтра до 5x5 вхідного зображення з кроком 1 і без доповнення, вихідна карта ознак буде мати розмір 3x3. Таким чином, згорткові шари є ключовими компонентами в побудові нейронних мереж для задач комп'ютерного зору, таких як класифікація зображень, виявлення об'єктів та сегментація зображень.

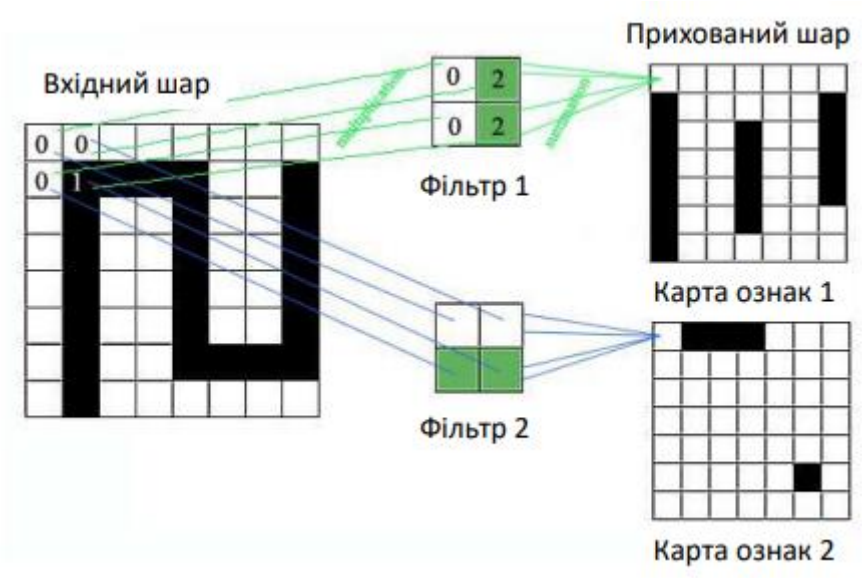

Рис. 1.2 – Візуалізація використання фільтрів

Згорткові шари зазвичай використовуються у комбінації з іншими шарами, такими як шари агрегації, шари активації і повноз’єднані шари, для побудови потужних нейронних мереж, здатних до вивчення ієрархічних ознак і розв'язання складних задач. Це робить згорткові нейронні мережі надзвичайно ефективними у задачах комп'ютерного зору, дозволяючи їм навчатися складних моделей зображень і успішно застосовувати їх у реальних завданнях.

1.1.2 Агрегувальний шар (Pooling Layer)

Агрегувальний шар є важливим компонентом згорткових нейронних мереж, який виконує функцію зменшення розмірності даних, зберігаючи при цьому найважливіші інформаційні ознаки. Цей шар допомагає зменшити обчислювальну складність моделі і зробити її більш стійкою до зміщень та інших варіацій у вхідних даних. Застосовуючи різні типи агрегування: максимізаційне або середнє, даний шар дозволяє моделі зберігати критично важливі ознаки, водночас зменшуючи об’єм даних для обробки.

Максимізаційне агрегування обирає найбільше значення в кожній підобласті, що дозволяє зберегти найбільш інтенсивні ознаки, які часто відповідають важливим локальним структурам. Наприклад, для вікна розміром 2x2 вибирається найбільше значення з цих чотирьох пікселів. А середнє агрегування відповідно обчислює середнє значення в кожній підобласті, що

дозволяє зберегти загальну інформацію про ознаки, розмиваючи локальні варіації. Глобальне об’єднання значно зменшує розмірність даних, оскільки кожна карта ознак зменшується до одного скалярного значення.

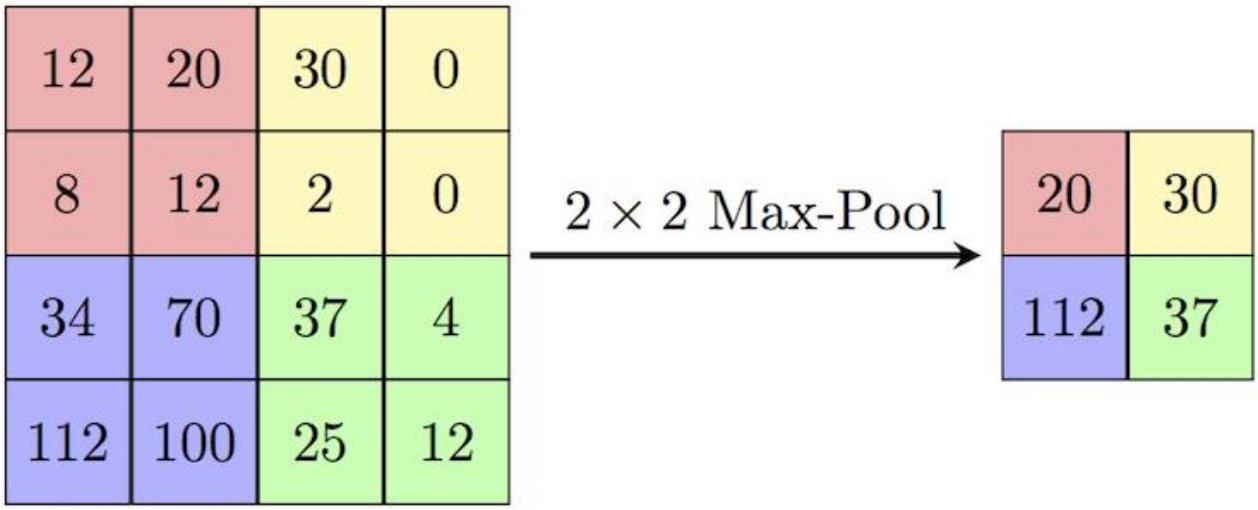

Рис. 1.3 – Візуалізація роботи максимізаційного агрегування

Даний процес можна проілюструвати на прикладі застосування максимізаційного агрегування з вікном 2x2 і кроком 2 до 16x16 вхідної карти ознак. Вибираючи максимальне значення у підобластях, можна отримати зменшену вихідну карту ознак, яка зберігає найбільш важливу інформацію. Таким чином, для вхідної карти ознак розміром 16x16 після застосування максимізаційного агрегування вихідна карта ознак матиме розмір 2x2.

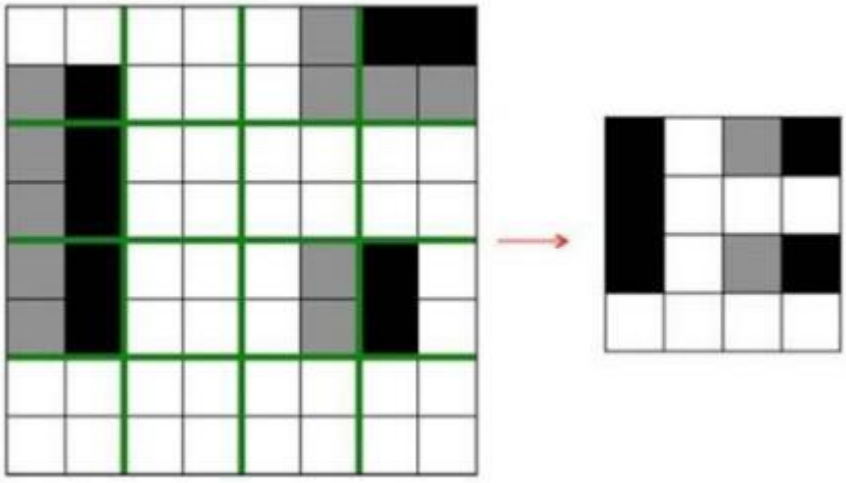

Рис. 1.4 – Приклад використання процедури максимізаційного агрегування

Гіперпараметри агрегувального шару, такі як розмір вікна і крок, впливають на ефективність зменшення розмірності та збереження інформації.

Вибір між видами об’єднання або іншими методами агрегації залежить від конкретного застосування і мети моделі. Наприклад, після максимізаційного агрегування з вікном 2x2 і кроком 2, розмір карти ознак зменшується вчетверо, що значно зменшує обчислювальну складність наступних шарів та робить модель більш стійкою до зміщень у вхідних даних.

Зменшення розмірності допомагає запобігти перенавчанню, зменшуючи кількість параметрів і зберігаючи лише найважливіші ознаки. Це особливо корисно у завданнях, таких як класифікація зображень, виявлення об’єктів та сегментація зображень. Використання агрегувальних шарів дозволяє моделі ефективніше обробляти дані та виявляти важливі ознаки на різних рівнях масштабування, що сприяє більш точному та ефективному розв’язанню складних задач.

1.1.3 Повноз’єднаний шар (Fully Connected Layer)

Повноз’єднаний шар є одним із основних компонентів нейронних мереж, який з’єднує кожен нейрон попереднього шару з кожним нейроном поточного шару. Він відіграє вирішальну роль при класифікації, регресії та інших завданнях, де потрібна складна нелінійна комбінація ознак. Кожен нейрон у повноз’єднаному шарі отримує вхідні дані від усіх нейронів попереднього шару. Кількість нейронів у шарі визначає його потужність і здатність моделі до навчання складних узорів. Кожен зв'язок між нейронами має вагу, яка налаштовується під час навчання моделі, і кожен нейрон має зсув, що додається до зваженої суми вхідних значень. Ваги і зсуви є параметрами, які оптимізуються під час навчання за допомогою алгоритму зворотного поширення помилки.

Після обчислення зваженої суми і додавання зсуву, результат проходить через активаційну функцію, яка додає нелінійність у модель. Вхідні дані до повноз’єднаного шару можуть надходити з попереднього шару нейронної мережі, зокрема зі згорткових або агрегувальних шарів. Кожен нейрон обчислює зважену суму своїх вхідних даних, де ваги і вхідні дані множаться і

додаються до зсуву, після чого результат проходить через активаційну функцію. Вихідні дані з кожного нейрону формують вихід повноз’єднаного шару, який стає вхідними даними для наступного шару.

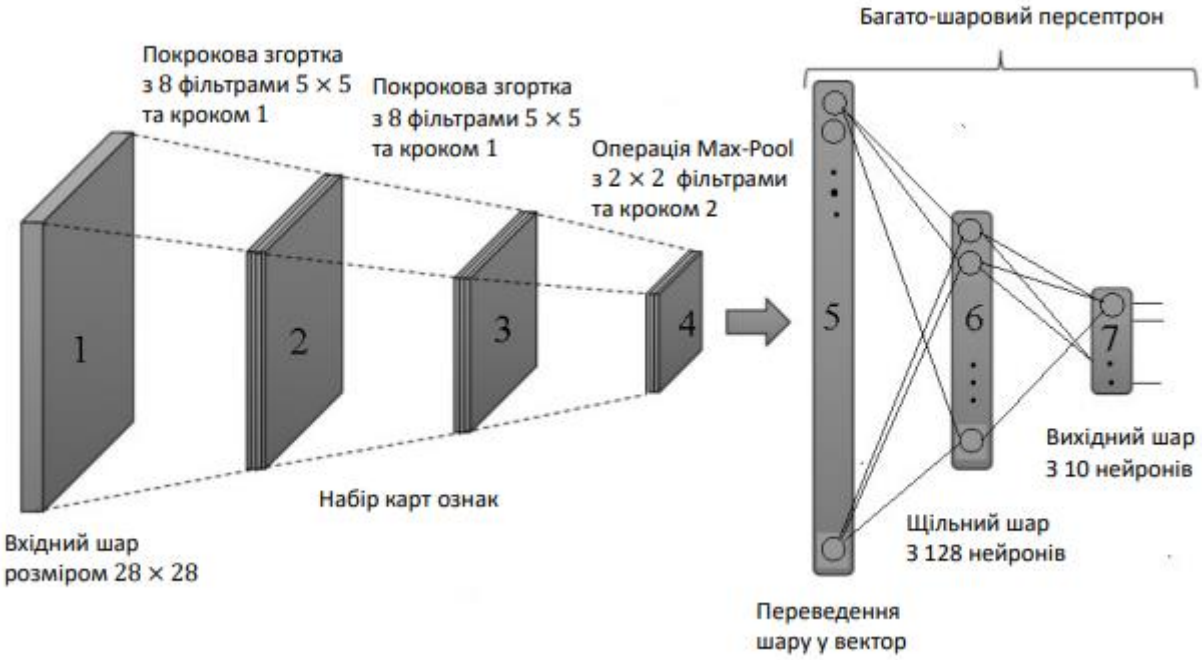

Рис. 1.5 – Приклад структури згорткової нейронної мережі

Кількість нейронів у повноз’єднаному шарі визначає розмірність вихідних даних шару. Збільшення кількості нейронів дозволяє моделі вивчати більш складні узори, але збільшує кількість параметрів і обчислювальну складність. Вибір активаційної функції впливає на здатність моделі вивчати нелінійні особливості, кожна з яких має свої переваги і недоліки в залежності від конкретного завдання. Методи регуляризації, такі як виключення, можуть бути використані для запобігання перенавчанню, випадково «вимикаючи» частину нейронів під час навчання, що сприяє генералізації моделі.

Отже, повноз’єднані шари є невід’ємною частиною нейронних мереж, оскільки забезпечують здатність моделі вивчати складні узори завдяки повній зв’язності між нейронами. Вони використовуються на завершальних етапах з метою класифікації витягнутих ознак з попередніх згорткових шарів. Подані шари використовуються для створення потужних моделей, що допомагають уникнути проблеми затухаючих градієнтів та сприяють кращому збереженню інформації. Повноз’єднані шари є ключовим компонентом нейронних мереж,

які забезпечують гнучкість та потужність при навчанні складних нелінійних узорів, і їх використання у комбінації з іншими типами шарів дозволяє створювати високоефективні моделі для широкого спектру задач у машинному навчанні.

1.1.4 Шар сплющення (Flatten Layer)

Шар сплющення у згортковій нейронній мережі відіграє важливу роль у підготовці даних для подальшої обробки повноз'єднаними шарами. Зазвичай, згорткові та агрегувальні шари створюють вихідні дані у вигляді багатовимірних тензорів. Ці тензори мають просторові розміри: висоту і ширину, а також глибину: кількість фільтрів або каналів. Для того, щоб передати ці дані на вхід повноз'єднаного шару, який очікує одномірний вектор, необхідно сплющити багатовимірний тензор у вектор

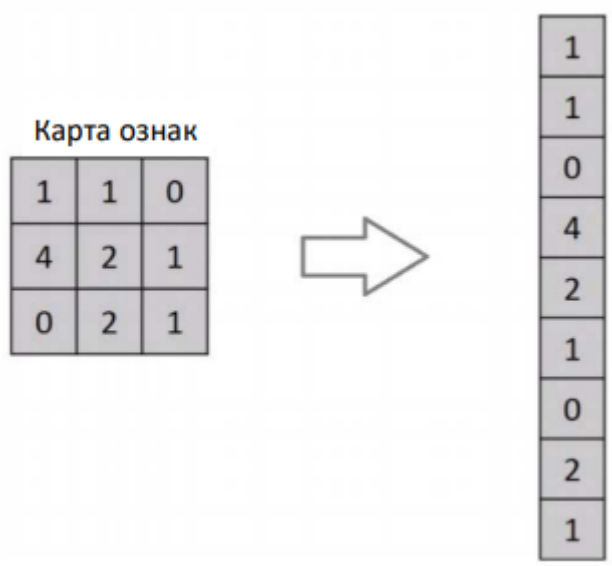

Рис. 1.6 – Візуалізація роботи сплющення

Даний процес реалізується за допомогою зміни форми тензора так, щоб усі його елементи були представлені у вигляді одного довгого вектора. Це забезпечує можливість підключення до шарів, які оперують з векторами фіксованої розмірності.

Крім того, шар сплющення дозволяє зберегти важливу інформацію про локальні узори та особливості, виявлені згортковими шарами, і передати цю інформацію до щільних шарів для подальшої класифікації або регресії. Це

забезпечує збереження складної просторової інформації у даних, яка може бути втрачена при інших методах перетворення, і значно покращує загальну ефективність мережі у завданнях обробки зображень та інших подібних проблем.

1.1.5 Шар нормалізації пакетів (Batch Normalization Layer)

Шар нормалізації пакетів є важливим компонентом сучасних нейронних мереж, який був введений для прискорення навчання та поліпшення стабільності моделі. Цей шар виконує нормалізацію вхідних даних у кожному міні-пакеті, що дозволяє зменшити внутрішнє зміщення та підвищити ефективність навчання. Вхідні дані до кожного шару нормалізуються на основі середнього значення та стандартного відхилення всього міні-пакету. Для кожного окремого виміру в пакеті даних обчислюються середнє значення і стандартне відхилення, після чого дані нормалізуються. Мале додатне значення, яке додається до стандартного відхилення, запобігає діленню на нуль. Після нормалізації дані масштабуються і зміщуються за допомогою параметрів, що навчаються разом з іншими параметрами моделі.

Нормалізація пакетів зменшує внутрішнє зміщення, що сприяє стабільнішому і швидшому навчанню, дозволяючи використовувати більші значення швидкості навчання і зменшуючи необхідність інших методів регуляризації, таких як виключення. Шар нормалізації пакетів може бути впроваджений у будь-якому місці в нейронній мережі, найчастіше після або перед активаційним шаром. Параметри масштабу та зміщення, які навчаються разом з іншими параметрами моделі, початково обираються як $\gamma=1$, $\beta=0$. Мале значення ϵ, яке додається до стандартного відхилення, зазвичай обирається як 10^{-5}. Момент визначає швидкість оновлення рухомих середніх та стандартних відхилень під час прогнозування, зазвичай обирається значення 0,9.

Під час навчання обчислюються середні та стандартні відхилення для кожного міні-пакету, які використовуються для нормалізації вхідних даних. Під час прогнозування використовуються заздалегідь обчислені середні та

стандартні відхилення, збережені під час навчання, що дозволяє моделі працювати стабільно з новими даними. Переваги пакетної нормалізації включають прискорення процесу навчання моделі, зменшення залежності від початкових значень ваг і зменшення необхідності у строгій регуляризації. Недоліки полягають у додаткових обчислювальних витратах під час навчання і необхідності обчислення статистик для кожного міні-пакету.

Нормалізацію пакетів використовують при класифікації зображень, обробці тексту та інших задачах глибинного навчання. Вона сприяє створенню більш швидких, стабільних і ефективних моделей, здатних вирішувати складні задачі у різних галузях. Цей шар є потужним інструментом, який допомагає підвищити ефективність та стабільність нейронних мереж.

1.1.6 Активаційний шар (Activation Layer)

Активаційний шар є ключовим компонентом у нейронних мережах, який додає нелінійність до моделі, дозволяючи їй вивчати складні узори та приймати гнучкі рішення. Без активаційних функцій нейронні мережі були б обмежені до лінійних моделей, що значно знижувало б їхню потужність. Активаційні функції додають нелінійність до мережі, що дозволяє моделі навчати складні та нерегулярні узори в даних. Кожен нейрон у мережі обчислює лінійну комбінацію вхідних значень, яка потім проходить через активаційну функцію.

Вибір активаційної функції залежить від конкретної задачі. Їх використовують у менш глибоких мережах або специфічних задачах, таких як бінарна класифікація чи моделювання часових рядів.

Даний шар можна додати до будь-якої нейронної мережі, наприклад, після повноз’єднаного шару, де нейрони обчислюють лінійну комбінацію вхідних даних, яка потім проходить через активаційну функцію, або після згорткового шару для додавання нелінійності. Вибір активаційної функції між шарами в глибоких мережах забезпечує навчання складних нелінійних узорів.

Таким чином, активаційний шар є вагомим компонентом нейронних мереж, який додає необхідну нелінійність і дозволяє моделям вчитися на

складних даних. Різноманітність активаційних функцій дає змогу вибирати оптимальну для конкретної задачі, що сприяє досягненню найкращих результатів у машинному навчанні.

1.1.7 Підсумок

Загалом, взаємодія вище описаних шарів забезпечує високу ефективність ЗНМ у навчанні на складних і різноманітних даних, дозволяючи досягати високих результатів у задачах розпізнавання образів, обробки природної мови та інших областях машинного навчання. Кожен з шарів виконує специфічну функцію, що в сукупності дозволяє створити потужну і гнучку модель для аналізу та інтерпретації даних.

1.2 Навчання «з нуля» (Zero-shot Learning)

Zero-shot Learning (ZSL) – це сучасний підхід у галузі машинного навчання, який дозволяє моделі класифікувати або розпізнавати об'єкти, які вона раніше не бачила під час навчання. На відміну від традиційних методів машинного навчання, які потребують значної кількості маркованих даних для кожного класу, ZSL використовує семантичну інформацію про нові класи для здійснення прогнозів.

1.2.1 Семантичні вектори

Семантичні вектори є одними з ключових елементів ZSL, оскільки вони забезпечують модель можливістю здійснювати класифікацію нових, досі невідомих класів. Вектори є математичним представлення класів багатовимірному просторі і містять інформацію про властивості або атрибути класів. Вони дозволяють моделі розуміти та класифікувати нові класи без прямого навчання на них.

Існують різні методи створення семантичних векторів, зокрема словникові та атрибутивні вектори. Серед словникових можна виділити ті вектори, які використовують нейронну мережу для навчання представлень слів

на основі їх контексту або ті, що використовують статистичну інформацію про частоту співставлення слів у тексті. У свою чергу, атрибутивні вектори відображають характеристики кожного класу і можуть бути визначені вручну або автоматично. Вони можуть бути як бінарними, так і числовими.

Джерелами даних для семантичних векторів служать великі текстові корпуси або новинні архіви, які містять різноманітні семантичні зв’язки між словами і поняттями. Онтології та семантичні мережі забезпечують зв’язки між поняттями на основі різних типів відношень. Метадані також можуть бути використані для визначення атрибутів класів, наприклад, інформація про тип об’єкта, колір, форму тощо.

Таким чином, семантичні вектори є ключовим елементом у навчанні «з нуля», забезпечуючи можливість класифікувати нові класи без прямого тренування на них. Правильне визначення, створення та використання семантичних векторів значно підвищує ефективність моделей ZSL, дозволяючи їм краще узагальнювати знання та застосовувати їх до нових завдань.

1.2.2 Навчання та проєкція

Навчання та проєкція є ключовими етапами у ZSL, які зосереджуються на створенні моделі, яка може ефективно вивчати зв’язок між вхідними даними та їхніми семантичними представленнями. Цей процес включає підготовку даних, вибір архітектури моделі, використання функцій втрат і методів регуляризації.

У контексті ZSL модель навчається на наборах даних, де доступні як вхідні дані, наприклад, зображення, так і семантичні вектори відповідних класів. Це дозволяє моделі вивчати зв’язки між вхідними даними та семантичними векторами. Модель використовує проєкційні методи для перенесення вхідних даних і семантичних векторів у спільний простір, де може порівнювати їх. Для цього застосовують різні архітектури проєкційних мереж, наприклад, CNN. Проєкційна модель навчається одночасно на обох видах даних, мінімізуючи функцію втрат для досягнення оптимальної проєкції в спільний простір.

Таким чином, навчання та проєкція у навчанні «з нуля» вимагають точного налаштування та оптимізації. Правильний вибір архітектури моделі, методів проєкції, функцій втрат та регуляризації значно впливає на продуктивність моделі.

1.2.3 Прогнозування результатів

Процес прогнозування у ZSL – це етап, на якому модель застосовується до нових, невідомих класів для здійснення класифікації. Під час нього модель використовує знання, здобуті на етапі навчання, для порівняння даних з семантичними векторами нових класів у спільному просторі.

Задля успішного прогнозу, вхідні дані, такі як зображення або текст, повинні бути нормалізовані відповідно до параметрів, використаних під час тренування моделі. Виділення ознак за допомогою ЗНМ для зображень дозволяє перетворити вхідні графічні дані у векторні представлення, готові для проєкції у спільний простір.

Наступним етапом є проєкція тестових даних у спільний простір. Вхідні вектори пропускаються через вже навчену модель, у якій порівнюються проєкції вхідних даних з семантичними векторами нових класів. Цей етап здійснюється за допомогою різних методів визначення подібності. Наприклад, косинусна відстань обчислює косинус кута між двома векторами, визначаючи їх схожість, що є корисним для нормалізованих векторів.

Завершальним етапом прогнозування є класифікація. Модель вибирає клас з найбільшою схожістю до вхідних даних. У випадках, коли необхідно отримати кілька найкращих прогнозів, модель може обирати класи за рівнем подібності.

Таким чином, процес прогнозування у ZSL має багатоступеневий підхід, що охоплює попередню обробку даних, проєкцію у спільний простір, порівняння з семантичними векторами та кінцеву класифікацію, забезпечуючи ефективне вирішення задач класифікації нових класів.

1.2.4 Підсумок

Підсумовуючи викладену інформацію, навчання «з нуля» є потужним інструментом для розв’язання задач, класифікації незнайомих класів. Завдяки своїй структурі, модель ZSL може демонструвати високу точність та надійність у задачах, де традиційних методів машинного навчання недостатньо. Правильне налаштування кожного з етапів процесу дозволяє максимально використати потенціал даного підходу для широкого спектру застосувань у реальному світі.

1.3 Вибір активаційної функції

Активаційні функції грають критичну роль у роботі нейронних мереж, а від її правильного вибору залежить здатність моделі до навчання та загальна ефективність. Вони використовуються в нейронних мережах для визначення виходу нейрона, який буде передано на наступний шар, а також додають нелінійність до моделі, що дозволяє нейронним мережам навчатися складним функціям та взаємозв'язкам у даних. Розгляну декілька основних активаційних функцій.

1.3.1 Випрямлена лінійна одиниця (ReLU)

ReLU (Rectified Linear Unit) є однією з найпоширеніших активаційних функцій у сучасних нейронних мережах.

Функція ReLU визначається як

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{1.1}$$

, де x – вхідне значення нейрона. Поширюється на проміжку $(-\infty; +\infty)$.

Похідна функції ReLU визначається як

$$f'(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{1.2}$$

, де x – вхідне значення нейрона.

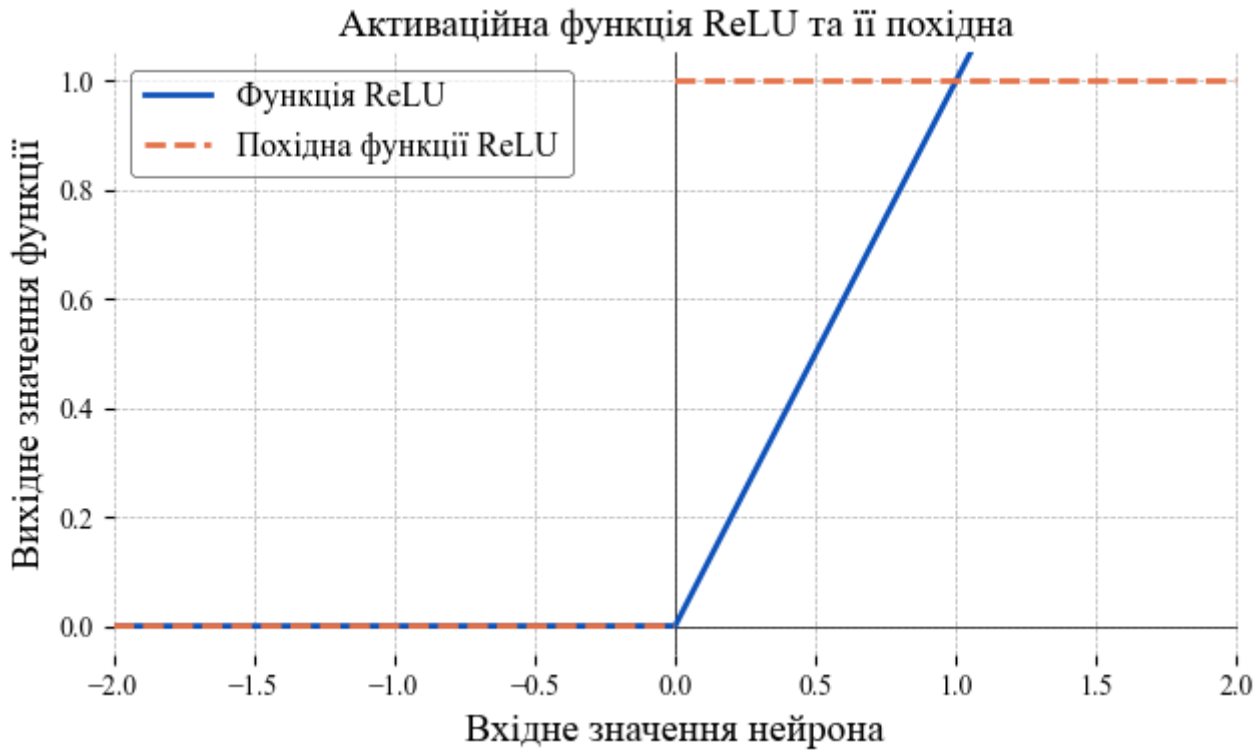

Рис. 1.7 – Активаційна функція ReLU та її похідна

Перевагами ReLU є простота обчислень та сприяння розрідженості виходів, що може покращити продуктивність мережі. Однак вона має і недоліки, такі як «вмирання нейронів», коли великі негативні значення вхідних даних можуть призвести до постійного нульового виходу нейронів. ReLU широко використовується в глибоких нейронних мережах для обробки зображень, таких як згорткові нейронні мережі (CNN), а також у загальних завданнях класифікації та регресії.

1.3.2 Нещільна випрямлена лінійна одиниця (Leaky ReLU)

Leaky ReLU є варіацією ReLU, який дозволяє невеликій частині вхідних значень проходити навіть у негативній області.

Функція Leaky ReLU визначається як

$$f(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{1.3}$$

, де x – вхідне значення нейрона, α – стале значення. Поширюється на проміжку $(-\infty; +\infty)$.

Похідна функції Leaky ReLU визначається як

$$f'(x) = \begin{cases} \alpha, & x < 0 \\ 1, & x \geq 0 \end{cases} \quad (1.4)$$

, де x – вхідне значення нейрона, α – стале значення.

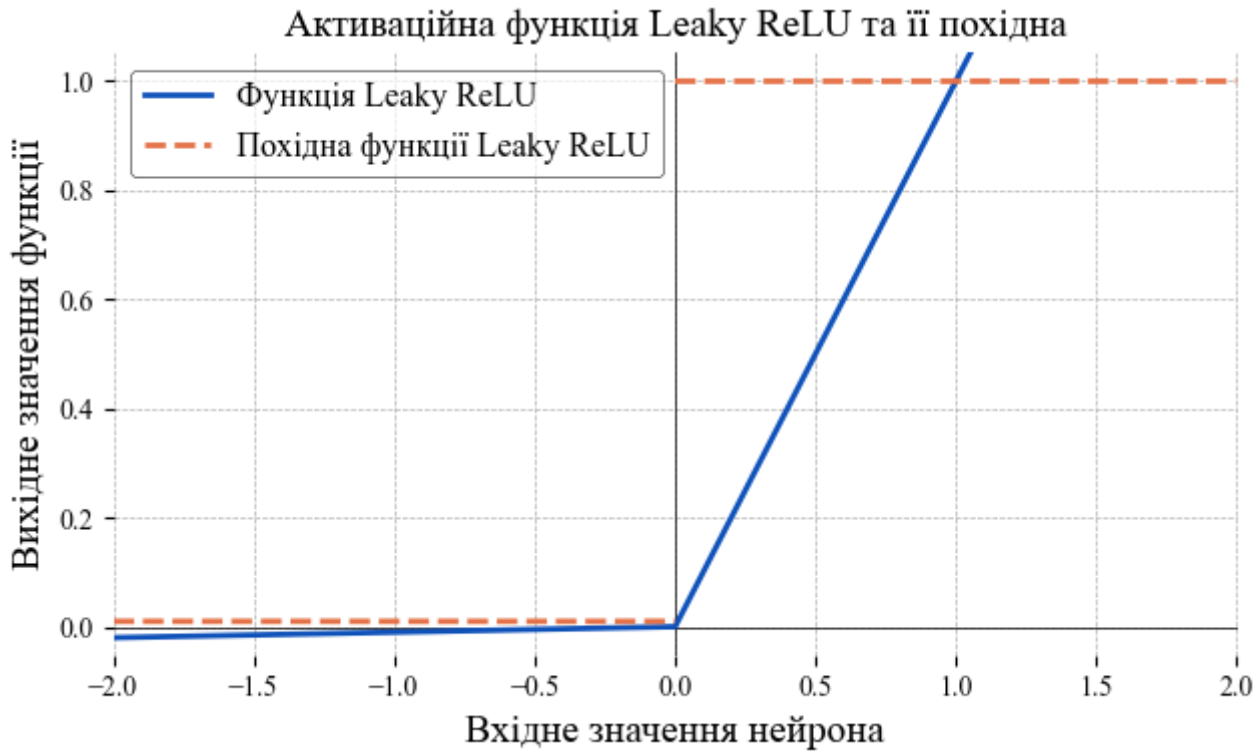

Рис. 1.8 – Активаційна функція Leaky ReLU та її похідна при $\alpha = 0.01$

Дана функція допомагає вирішити проблему «вмирання нейронів», зберігаючи при цьому більшість переваг ReLU. Недоліком Leaky ReLU може бути зниження точності при неправильному налаштуванні параметра витоку. Ця функція використовується у завданнях, де ReLU неефективна через "вмирання нейронів", та в мережах з великою кількістю шарів.

1.3.3 Експоненціальна лінійна одиниця (ELU)

ELU (Exponential Linear Unit) є ще однією модифікацією ReLU, яка надає вигоду від експоненціального зростання у негативній області.

Функція ELU визначається як

$$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases} \quad (1.5)$$

, де x – вхідне значення нейрона, α – стале значення. Поширюється на проміжку $(-\alpha; +\infty)$.

Похідна функції ELU визначається як

$$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & x < 0 \\ 1 & x \geq 0 \end{cases} \quad (1.6)$$

, де x – вхідне значення нейрона, α – стале значення.

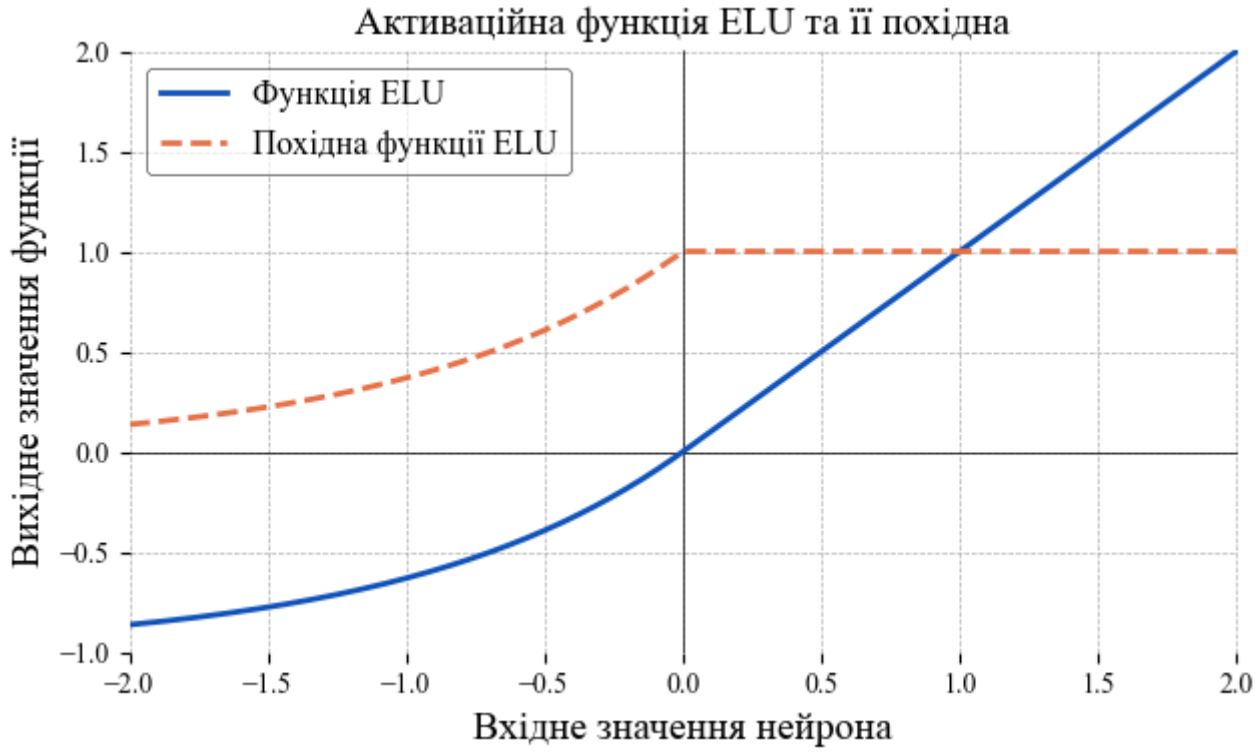

Рис. 1.9 – Активаційна функція ELU та її похідна при $\alpha = 1$

Ця функція сприяє швидшому навчанню та досягненню більшої продуктивності мережі, шляхом зменшення упередження зміщення середніх активацій до нуля. Також ELU має більшу обчислювальну складність порівняно з ReLU та потребує налаштування параметра α. ELU особливо корисна для глибоких нейронних мереж, де необхідна швидкість навчання та стабільність.

1.3.4 Масштабована експоненціальна лінійна одиниця (SELU)

SELU (Scaled Exponential Linear Unit) є варіацією ELU, яка автоматично масштабує вихідні значення, забезпечуючи самонормалізацію шарів нейронної мережі.

Функція SELU визначається як

$$f(\alpha, x) = \begin{cases} \lambda\alpha(e^x - 1) & x < 0 \\ \lambda x & x \geq 0 \end{cases} \quad (1.7)$$

, де x – вхідне значення нейрона, α – стале значення, λ – коефіцієнт масштабування. Поширюється на проміжку $(-\lambda\alpha; +\infty)$.

Похідна функції SELU визначається як

$$f'(\alpha, x) = \begin{cases} \lambda\alpha e^x & x < 0 \\ \lambda & x \geq 0 \end{cases} \quad (1.8)$$

, де x – вхідне значення нейрона, α – стале значення, λ – коефіцієнт масштабування.

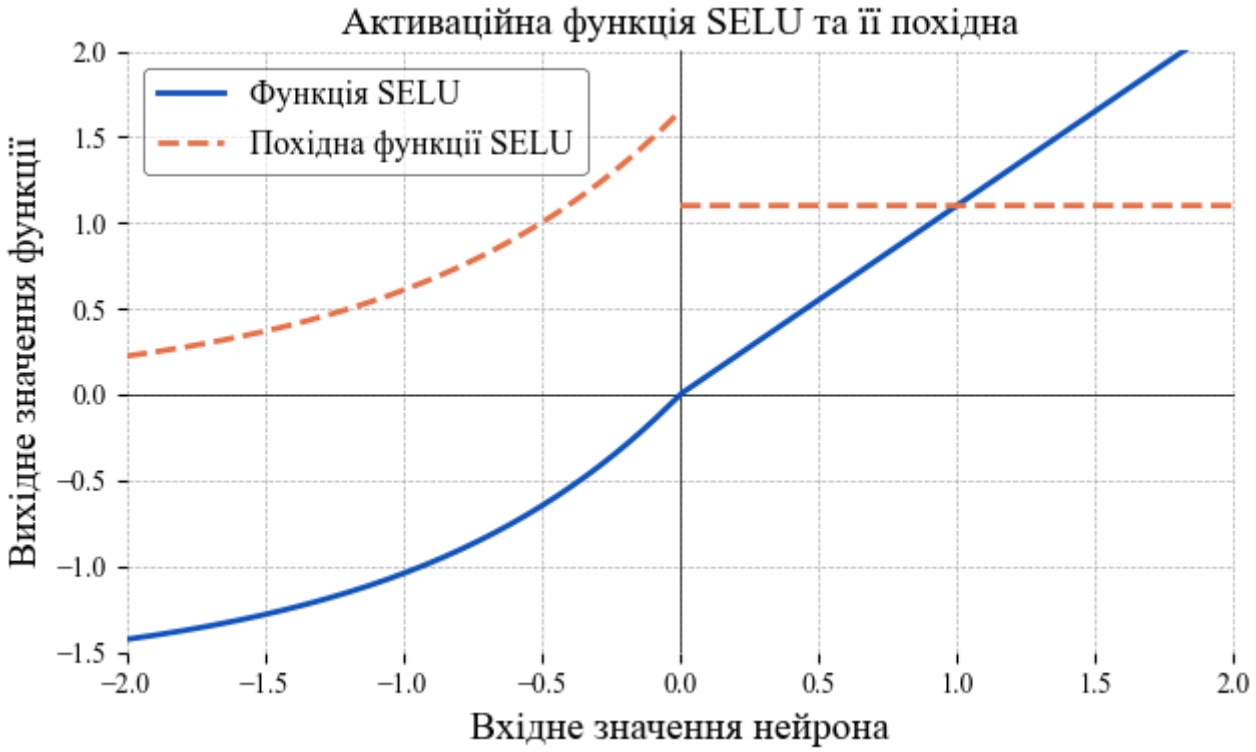

Рис. 1.10 – Активаційна функція SELU та її похідна при $\alpha = 1.5$ і $\lambda = 1.1$

Функція SELU сприяє стабільному і швидкому навчанню глибоких нейронних мереж. Проте вона потребує правильної ініціалізації та налаштування параметрів, і має більшу обчислювальну складність порівняно з ReLU і Leaky ReLU. Вона використовується у завданнях з високою вимогою до точності та продуктивності.

1.3.5 Нормована експоненціальна функція (Softmax)

Softmax є активаційною функцією, яка зазвичай використовується у вихідному шарі класифікаційних нейронних мереж. Вона перетворює вектор логітів у вектор ймовірностей, де сума всіх ймовірностей дорівнює одиниці.

Функція Softmax визначається як

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}, \quad i \in (1; J) \tag{1.9}$$

, де x – вхідне значення нейрона, i – номер поточної ітерації, J – розмірність вектора. Поширюється на проміжку $(0; 1)$.

Похідна функції Softmax визначається як

$$\frac{\partial f_i(\vec{x})}{\partial x_j} = f_i(\vec{x}) \left(\delta_{ij} - f_j(\vec{x}) \right) \tag{1.10}$$

, де x – вхідне значення нейрона, δ_{ij} – символ Кронекера, i, j – номери поточної ітерації.

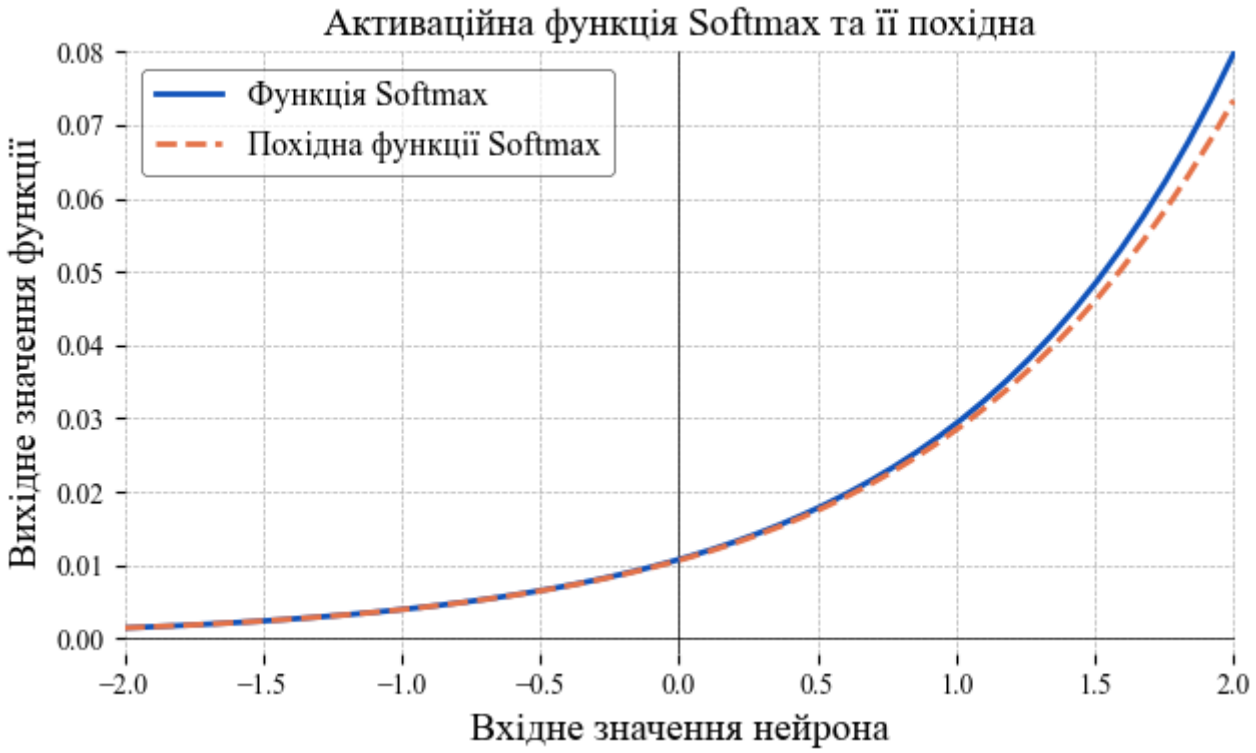

Рис. 1.11 – Активаційна функція Softmax та її похідна

Дана функція дозволяє інтерпретувати виходи моделі як ймовірності належності до певних класів, що є зручним для задач класифікації. Однак Softmax не підходить для використання в прихованих шарах через відсутність розрідженості. Вона використовується в класифікаційних задачах з багатьма класами, таких як розпізнавання образів та обробка природної мови.

1.3.6 Функція сигмоїди (Sigmoid)

Сигмоїда є однією з головних активаційних функцій, яка часто використовується в нейронних мережах для згладжування значень величини або визначення імовірності.

Функція сигмоїди визначається як

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.11}$$

, де x – вхідне значення нейрона. Поширюється на проміжку $(0; 1)$.

Похідна функції сигмоїди визначається як

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big) \tag{1.12}$$

, де x – вхідне значення нейрона.

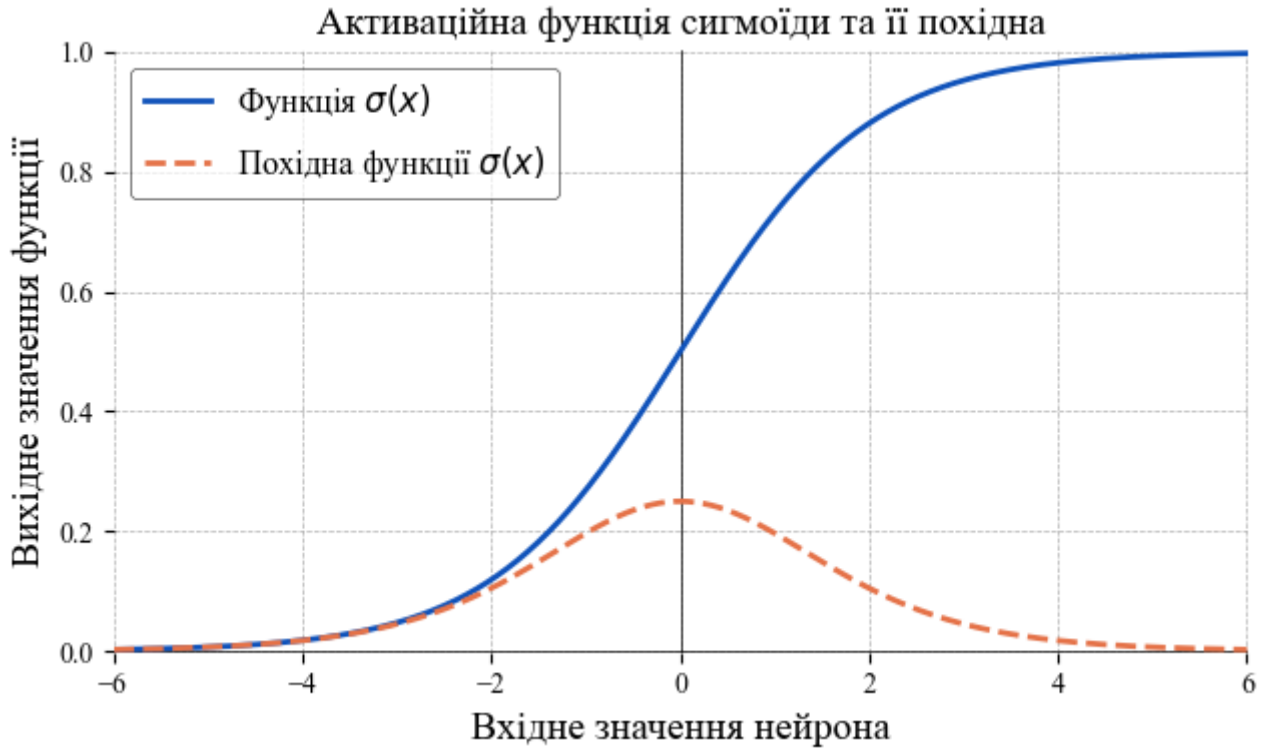

Рис. 1.12 – Активаційна функція сигмоїди та її похідна

Однією з головних переваг сигмоїди є її гладкість і можливість м'якого переходу між 0 і 1, що робить її корисною для задач, де важливо мати плавні градієнти, наприклад у бінарній класифікації. Однак ця функція має і недоліки, такі як проблема «згасаючих градієнтів», коли при великих або малих значеннях нейронів градієнт стає незначним, що ускладнює навчання глибоких нейронних мереж.

Отже, сигмоїда використовується у нейронних мережах із специфічною архітектурою та в задачах, де необхідно отримати вихідні значення у вигляді ймовірності.

1.3.7 Функція гіперболічного тангенса (Tanh)

Гіперболічний тангенс є фактично зміщеною та масштабованою версією сигмоїди. Тому він також успішно справляється із визначенням імовірностей, а також корисний при роботі із центрованими даними.

Функція Tanh визначається як

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{1.13}$$

, де x – вхідне значення нейрона. Поширюється на проміжку $(-1; 1)$.

Похідна функції Tanh визначається як

$$tanh'(x) = 1 - tanh^2(x) \tag{1.14}$$

, де x – вхідне значення нейрона.

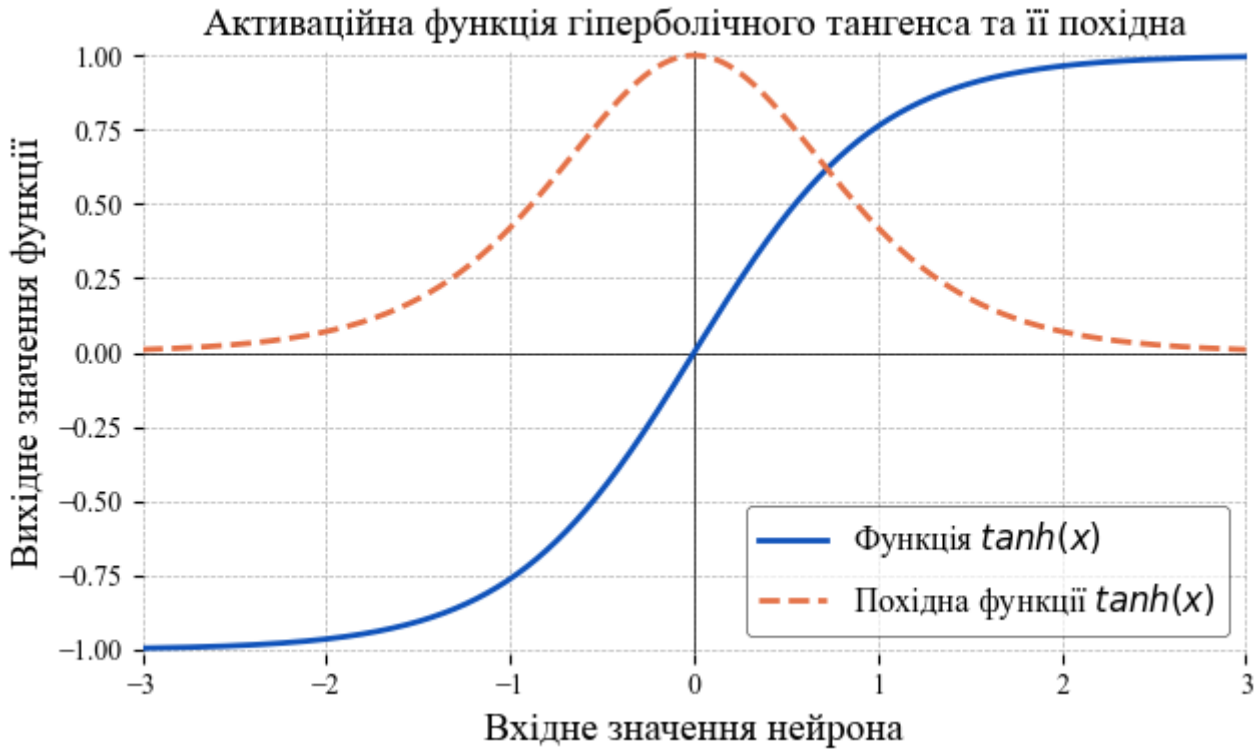

Рис. 1.13 – Активаційна функція гіперболічного тангенса та її похідна

Однією з головних переваг гіперболічного тангенса є його властивість центрованості навколо нуля, що може сприяти швидшому навчанню, оскільки

середнє значення активації ближче до нуля. Це допомагає зменшити зсув під час передавання даних між шарами нейронної мережі. Однак ця функція, як сигмоїда, має проблему «згасаючих градієнтів».

Функція гіперболічного тангенса є популярною у багатьох моделях нейронних мереж і використовується в у бінарній класифікації або у задачах, де необхідно отримати вихідні дані на проміжку $(-1; 1)$.

1.3.8 Підсумок

Таким чином, правильний вибір активаційної функції є ключовим аспектом у налаштуванні нейронної мережі, що безпосередньо впливає на її здатність до навчання і загальну продуктивність. Кожна активаційна функція має свої переваги, недоліки та області застосування. Вибір тієї чи іншої функції залежить від специфіки задачі, архітектури мережі та вимог до продуктивності.

1.4 Вибір оптимізатора нейронної мережі

Оптимізатори є важливою складовою навчання нейронних мереж. Вони відповідають за коригування ваг моделі з метою мінімізації функції втрат. Вибір правильного оптимізатора є критично важливим, адже від нього залежить ефективність, швидкість збіжності та стабільність моделі. Існує багато різних оптимізаторів, кожен з яких має свої особливості та підходить для різних типів задач. У цьому розділі розгляну декілька з них, їх переваги та недоліки.

1.4.1 Стохастичний градієнтний спуск (SGD)

SGD (Stochastic Gradient Descent) є базовим методом оптимізації для нейронних мереж. Він оновлює ваги моделі, використовуючи градієнт втрати для одного випадково зразка навчальних даних.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t) \tag{1.15}$$

, де θ_t – поточні ваги моделі на ітерації t, η – швидкість навчання (learning rate), а $\nabla_\theta L(\theta_t)$ – градієнт функції втрат L відносно параметрів θ на ітерації t.

Стохастичний градієнтний спуск (SGD) має кілька значних переваг, серед яких простота реалізації та низька обчислювальна вартість на кожній ітерації. Завдяки цим властивостям, метод легко застосовувати до різних задач, забезпечуючи швидке оновлення ваг на основі одного зразка даних. Це особливо корисно при роботі з великими наборами даних або в режимі реального часу.

Однак у цього оптимізатора є й недоліки. Основним з них є низька стабільність через використання одного зразка, що може призводити до значних коливань градієнтів і нестабільних оновлень ваг. Це може спричиняти застрягання у локальних мінімумах, що є проблемою при навчанні глибоких нейронних мереж. Також процес навчання може бути повільним, особливо на великих наборах даних.

Ключовим параметром SGD є швидкість навчання (η). Неправильне налаштування цього параметра може призвести до коливань або повільного навчання. Часто використовується метод зменшення швидкості навчання, що допомагає спочатку швидко знижувати функцію втрат, а потім обережніше наближатися до оптимуму. Коливання градієнтів можна зменшити за рахунок використання міні партій градієнтного спуску, який працює з невеликими групами зразків на кожній ітерації.

Загалом, стохастичний градієнтний спуск є ефективним методом оптимізації нейронних мереж, особливо для великих наборів даних, хоча і має певні недоліки, які можна частково компенсувати налаштуванням параметрів навчання та використанням міні партій.

1.4.2 Середньоквадратичне поширення (RMSprop)

RMSprop (Root Mean Square Propagation) є варіацією стохастичного градієнтного спуску (SGD), яка адаптивно налаштовує швидкість навчання для кожного параметра, використовуючи середньоквадратичне значення градієнта. Це дозволяє моделі ефективніше навчатися, особливо в умовах, коли градієнти мають різні масштаби. Формула RMSprop передбачає обчислення

експоненціально зваженого середньоквадратичного значення градієнтів та оновлення ваг з урахуванням цього значення, що допомагає уникати великих коливань під час навчання.

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho) g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \quad (1.16)$$

, де $E[g^2]_t$ – експоненціально зважене середньоквадратичне значення градієнта на ітерації t, ρ – фактор зважування, g_t – градієнт на ітерації t, θ_t – поточні ваги на ітерації t, η – швидкість навчання, а ϵ – мале значення для уникнення ділення на нуль.

Основні переваги RMSprop включають адаптивне налаштування швидкості навчання, що робить процес навчання стабільнішим і швидшим. Це особливо корисно для нейронних мереж, у яких проблема зникаючих або вибухаючих градієнтів є частою.

Проте, RMSprop має свої недоліки, зокрема залежність від правильного налаштування гіперпараметрів, таких як фактор зважування (ρ) і швидкість навчання (η). Зазвичай ρ вибирається близьким до 1, наприклад, 0.9 для забезпечення значного згладжування. Крім того, додаткові обчислення для підтримки середньоквадратичних значень градієнтів можуть збільшити загальну обчислювальну вартість.

Отже, RMSprop є ефективним методом оптимізації для нейронних мереж, особливо в задачах, де градієнти можуть бути малими або рідкими. Його здатність адаптивно регулювати швидкість навчання для кожного параметра забезпечує стабільність та швидкість збіжності. Однак, правильний вибір гіперпараметрів та додаткові обчислення вимагають більше ресурсів і уваги при налаштуванні.

1.4.3 Слідування за упорядкованим лідером (Ftrl)

Ftrl (Follow-The-Regularized-Leader) є оптимізатором, що поєднує градієнтний спуск і регуляризацію, ефективно працюючи з великими наборами даних та розрідженими функціями втрат. Метод базується на оновленні ваг за

допомогою градієнтів та регуляризаційних термінів, що допомагає знизити обчислювальні витрати та покращити інтерпретованість моделі.

$$\theta_{t+1} = \arg\min_{\theta}\left(\sum_{s=1}^{t} g_s\theta + \frac{1}{2}\sum_{s=1}^{t}\sigma_s\theta^2 + \lambda\|\theta\|_1\right) \quad (1.17)$$

, де θ_t – поточні ваги на ітерації t, g_s – градієнт на ітерації t, σ_s – адаптивний параметр регуляризації, λ – коефіцієнт L1-регуляризації.

Основні переваги Ftrl включають здатність працювати з великими наборами даних і моделями з великою кількістю параметрів, високу ефективність при використанні L1-регуляризації, що запобігає перенавчанню, та покращує продуктивність на розріджених функціях втрат. Цей метод добре підходить для задач із рідкими градієнтами.

Недоліками даного оптимізатора є складність налаштування гіперпараметрів, таких як адаптивний параметр регуляризації (σ) та коефіцієнт L1-регуляризації (λ), а також високу обчислювальну вартість через додаткові обчислення для регуляризації та оновлення ваг. Незважаючи на це, Ftrl може забезпечити ефективне навчання моделей за умови правильного налаштування параметрів.

Завдяки своїй здатності ефективно працювати з великими і розрідженими наборами даних, Ftrl є потужним інструментом для багатьох задач, хоча його налаштування та обчислювальна вартість вимагають додаткової уваги.

1.4.4 Оцінка адаптивного моменту (Adam)

Adam (Adaptive Moment Estimation) є одним з найпопулярніших оптимізаторів в глибокому навчанні. Він поєднує методи стохастичного градієнтного спуску (SGD) та обчислення моментів, дозволяючи адаптивно регулювати швидкість навчання для кожного параметра. Adam коригує кроки навчання відповідно до емпіричних моментів першого і другого порядку, що допомагає моделі швидко і стабільно збігатися.

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
\theta_{t+1} &= \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned}
\tag{1.18}
$$

, де m_t – оцінка першого моменту (середнє значення градієнтів), v_t – оцінка другого моменту (середнє значення квадратів градієнтів), β_1, β_2 – коефіцієнти згладжування для моментів, $\hat{m}_t, \hat{v}_t$ – скориговані оцінки моментів, θ_t – поточні ваги на ітерації t, η – швидкість навчання, а ϵ – мале значення для уникнення ділення на нуль.

Основною перевагою Adam є його здатність адаптивно налаштовувати швидкість навчання для кожного параметра, що забезпечує ефективне навчання та обробку шумних градієнтів. Це сприяє уникненню коливань і забезпечує більш гладку збіжність. Завдяки цим властивостям, оптимізатор демонструє швидшу збіжність порівняно з іншими, що робить його особливо корисним для великих і складних нейронних мереж.

Однак, оцінка адаптивного моменту має і деякі недоліки. Через необхідність збереження додаткових моментів першого та другого порядку, вона може вимагати значної кількості пам'яті, особливо для великих моделей. Крім того, Adam може стикатися з проблемами при обробці дуже великих градієнтів, що може призводити до труднощів зі збіжністю. Ефективність цього оптимізатора також сильно залежить від правильного налаштування гіперпараметрів, таких як коефіцієнти згладжування моментів та швидкість навчання.

Таким чином, Adam є потужним і гнучким оптимізатором, який добре підходить для різноманітних задач у глибокому навчанні, хоча його використання потребує належного налаштування та ресурсів.

1.4.5 Оцінка адаптивного моменту з максимумом (Adamax)

Adamax є варіацією алгоритму Adam, яка використовує безперервне нормалізоване значення градієнтів замість другого моменту, що покращує стабільність навчання при роботі з великими градієнтами.

$$\begin{gathered} m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \\ u_t = \max(\beta_2 u_{t-1}; |g_t|) \\ \theta_{t+1} = \theta_t - \eta \frac{m_t}{u_t} \end{gathered} \tag{1.19}$$

, де m_t – оцінка першого моменту, u_t – максимум абсолютних градієнтів, β_1, β_2 – коефіцієнти згладжування, θ_t – поточні ваги на ітерації t, η – швидкість навчання.

Основною перевагою Adamax є його здатність адаптивно регулювати швидкість навчання для кожного параметра, що забезпечує ефективне і стабільне навчання. Використання максимуму абсолютних значень градієнтів підвищує стабільність, особливо у випадках з вибухом градієнтів, що робить оптимізатор менш чутливим до вибору гіперпараметрів, таких як β_2. Завдяки оновленню першого моменту і максимуму абсолютних значень градієнтів на кожні ітерації, Adamax забезпечує більш плавне та стабільне зближення у порівнянні з іншими оптимізаторами. Ваги оновлюються з урахуванням цих двох компонентів, що дозволяє оптимізатору ефективно працювати навіть у складних і великих нейронних мережах.

Недоліками є високі вимоги до пам'яті через необхідність збереження додаткових моментів, та більша обчислювальна вартість через обчислення максимуму абсолютних градієнтів.

Отже, Adamax є потужним і стабільним оптимізатором, який добре підходить для широкого спектра задач, особливо при роботі з великими градієнтами. Його здатність ефективно адаптувати швидкість навчання для кожного параметра підвищує продуктивність та стабільність моделі, хоча високі вимоги до пам'яті та обчислювальних ресурсів можуть бути викликами при використанні цього оптимізатора.

1.4.6 Метод адаптивної швидкості навчання (Adadelta)

Adadelta – це адаптивний оптимізатор, який комбінує ідеї методів SGD і RMSprop, використовуючи експоненціальне згладжування градієнтів для адаптивного регулювання швидкості навчання. Формули методу дозволяють враховувати історію градієнтів і змін ваг, забезпечуючи адаптивне коригування ваг без потреби у фіксованій швидкості навчання.

$$\begin{gathered} E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho) g_t^2 \\ \Delta\theta_t = -\frac{\sqrt{E[\Delta\theta^2]_{t-1} + \epsilon}}{\sqrt{E[\Delta g^2]_t + \epsilon}} g_t \\ E[\Delta\theta^2]_t = \rho E[\Delta\theta^2]_{t-1} + (1-\rho)(\Delta\theta_t)^2 \\ \theta_{t+1} = \theta_t + \Delta\theta_t \end{gathered} \tag{1.20}$$

, де $E[g^2]_t$ – експоненціально зважене середньоквадратичне значення градієнта на ітерації t, ρ – коефіцієнт згладжування, g – градієнт на ітерації t, θ_t – поточні ваги на ітерації t, ϵ – мале значення для уникнення ділення на нуль, а $E[\Delta\theta^2]_t$ – експоненціально зважене середньоквадратичне значення змін ваг на ітерації t.

Переваги Adadelta включають автоматичне налаштування швидкості навчання, що робить його зручним для автоматизованих систем, та стабільність при великих градієнтах завдяки експоненціальному згладжуванню. Він також зменшує необхідність у глобальних гіперпараметрах, спрощуючи процес налаштування моделі.

Однак, даний оптимізатор може бути менш ефективним для дуже великих наборів даних та вимагати додаткових обчислювальних ресурсів через збереження експоненціально зважених середніх значень градієнтів і змін ваг.

Таким чином, Adadelta є ефективним і стабільним оптимізатором, що забезпечує адаптивне і стабільне навчання без необхідності у ручному налаштуванні гіперпараметрів. Він підходить для задач з різноманітними градієнтами та допомагає уникнути нестабільності при великих градієнтах. Проте, його ефективність може знижуватися на дуже великих наборах даних, де потрібні більш специфічні налаштування.

1.4.7 Адаптивний градієнтний алгоритм (Adagrad)

Adagrad (Adaptive Gradient Algorithm) є оптимізатором, який адаптивно регулює швидкість навчання для кожного параметра, враховуючи його історію градієнтів. Це робить оптимізатор ефективним для задач із рідкими та нестаціонарними градієнтами. Основні формули включають оновлення суми квадратів градієнтів та оновлення ваг. Зміни ваг проводяться із використанням зворотної величини квадратного кореня суми квадратів градієнтів, що означає, що параметри з високими градієнтами мають менші зміни, а параметри з низькими градієнтами – більші. Це дозволяє враховувати всю історію градієнтів, роблячи процес навчання більш адаптивним і стабільним.

$$G_t = G_{t-1} + g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t} + \epsilon} g_t \tag{1.21}$$

, де G_t – сума квадратів градієнтів на ітерації t, g_t – градієнт на ітерації t, θ_t – поточні ваги на ітерації t, η – швидкість навчання, а ϵ – мале значення для уникнення ділення на нуль.

Адаптивний градієнтний алгоритм має кілька переваг, включаючи автоматичне налаштування швидкості навчання і ефективність для рідких градієнтів.

Однак, накопичення градієнтів може призвести до значного зменшення швидкості навчання з часом, що може зупинити навчання перед досягненням оптимуму. Також, зберігання суми квадратів градієнтів для кожного параметра вимагає додаткової пам'яті, що може бути проблематичним для моделей з великою кількістю параметрів.

Отже, Adagrad є ефективним адаптивним оптимізатором для задач з рідкими та нестаціонарними градієнтами. Його здатність автоматично налаштовувати швидкість навчання на основі історії градієнтів допомагає уникнути перенавчання та забезпечує стабільне навчання. Проте, зниження швидкості навчання з часом через накопичення градієнтів може бути викликом, який варто враховувати при використанні цього оптимізатора.

1.4.8 Підсумок

Вибір оптимізатора є важливим кроком у процесі навчання нейронних мереж. Кожен оптимізатор має свої унікальні переваги та недоліки, які варто враховувати при виборі оптимального варіанту для конкретної задачі. Наприклад, SGD є простим та швидким, але може бути нестабільним. RMSprop та Adadelta адаптивно регулюють швидкість навчання, що підвищує стабільність. Adam та його варіанти, наприклад Adamax, ефективні завдяки комбінуванню моментів, але можуть потребувати більше пам'яті. Adagrad добре працює з рідкими градієнтами, але його швидкість навчання може зменшуватися з часом.

Експериментування з параметрами кожного оптимізатора для конкретної задачі є ключем до забезпечення оптимальної швидкості та якості навчання моделі. Врахування характеристик даних, специфіки задачі та ресурсних обмежень допоможе зробити правильний вибір і досягти кращих результатів у навчанні нейронної мережі.

1.5 Опис метрик оцінки моделі

Метрики використовуються для оцінки продуктивності моделі та допомагають зрозуміти, наскільки добре модель виконує своє завдання, забезпечуючи об'єктивні критерії для порівняння різних моделей та їх налаштувань. Вибір відповідної метрики є критично важливим, оскільки різні метрики можуть по-різному відображати продуктивність моделі залежно від характеру даних та конкретного завдання. Важливо розуміти, як кожна метрика працює, її переваги та недоліки, щоб робити обґрунтовані рішення під час розробки та оцінки моделей машинного навчання.

1.5.1 Функція втрат (Loss)

Функція втрат є однією з найважливіших метрик для навчання нейронних мереж. Вона вимірює наскільки добре модель передбачає цільові значення.

Категорична перехресна ентропія (Categorical Cross-entropy) є однією з найбільш поширених функцій втрат, яка використовується для задач багатокласової класифікації. Вона вимірює різницю між розподілом імовірностей, що передбачені моделлю, і фактичним розподілом.

$$L = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{C} y_{ij}\log(p_{ij}) \tag{1.22}$$

, де N – кількість зразків у наборі даних, y_{ij} – бінарний індикатор для зразка j і класу i, а p_{ij} – передбачена імовірність для зразка j і класу i.

Категорична перехресна ентропія враховує імовірності, що передбачені моделлю, і стимулює модель надавати більш точні імовірності для кожного класу, а також має властивості, які забезпечують стабільні градієнти під час процесу оптимізації, що допомагає швидкому та ефективному навчанню моделі.

Для задач бінарної класифікації використовується бінарна перехресна ентропія (Binary Cross-entropy). Вона є спрощеною версією категоричної перехресної ентропії і використовується тоді, коли вихід моделі може приймати лише два значення, наприклад, 0 або 1.

$$L = -\frac{1}{N}\sum_{j=1}^{N}\big(y_j\log(p_j) + (1-y_j)\log(1-p_j)\big) \tag{1.23}$$

, де N – кількість зразків у наборі даних, y_j –фактичне значення для зразка j, а p_j – передбачена імовірність для зразка j.

Однак, для використання функції, передбачення моделі повинні бути нормалізовані до імовірностей, що зазвичай досягається за допомогою активаційної функції Softmax на вихідному шарі нейронної мережі. Для бінарної класифікації використовується активаційна функція сигмоїди, яка перетворює вихідне значення у необхідний діапазон $(0; 1)$.

Як і багато інших функцій втрат, категорична та бінарна перехресна ентропії можуть бути чутливими до дисбалансу класів у наборі даних. Якщо один клас значно переважає інші, модель може навчитися віддавати перевагу цьому класу, що призведе до поганої продуктивності для менш представлених

класів. Для покращення роботи, у випадках з дисбалансом класів, можна використовувати додаткові методи: збільшення вагових коефіцієнтів для менш представлених і відповідно їх зменшення для більш представлених класів, або вирівнювання кількості зразків кожного класу.

Таким чином, функція втрат є потужним інструментом для задач багатокласової класифікації, забезпечуючи ефективне навчання моделей та чутливість до точності передбачуваних імовірностей. Проте, для найкращих результатів, особливо у випадках з дисбалансом класів, її слід використовувати разом з додатковими методами або модифікаціями.

1.5.2 Точність (Accuracy)

Точність є загальною метрикою для оцінки класифікаційних моделей. Вона визначає наскільки правильно модель класифікує зразки як позитивні, так і негативні.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.24}$$

, де TP (True Positives) – кількість істинно позитивних випадків, тобто кількість правильно передбачених позитивних зразків, TN (True Negatives) – кількість істинно негативних випадків, тобто кількість правильно передбачених негативних зразків, FP (False Positives) – кількість хибно позитивних випадків, тобто кількість негативних зразків, які були неправильно класифіковані як позитивні, а FN (False Negatives) – кількість хибно негативних випадків, тобто кількість позитивних зразків, які були неправильно класифіковані як негативні.

Точність є простою та зрозумілою метрикою, яка надає загальну оцінку продуктивності класифікаційної моделі і яка легко обчислюється. Проте, вона може бути оманливою при роботі з наборами даних, де певні класи значно перевершують інші за кількістю прикладів. Наприклад, якщо 95% зразків належать до одного класу, модель, яка завжди передбачає цей клас, матиме високу точність, хоча й буде неефективною для менш представленого класу. А також, точність не враховує відмінності між помилками класифікації: хибно позитивними та хибно негативними результатами. Щоб отримати більш повну

картину продуктивності моделі, точність часто використовується разом з іншими метриками, такими як влучність, повнота і F_1-міра.

Таким чином, хоч точність і є корисною метрикою, важливо доповнювати її іншими метриками для повної оцінки продуктивності класифікаційної моделі, особливо в контексті незбалансованих наборів даних.

1.5.3 Влучність (Precision)

Влучність є основною метрикою для оцінки класифікаційних моделей. Вона вимірює наскільки добре модель класифікує позитивні зразки, визначаючи відсоток істинних серед усіх позитивних передбачень.

$$Precision = \frac{TP}{TP + FP} \tag{1.25}$$

, де TP – кількість істинно позитивних випадків, а FP – кількість хибно позитивних випадків.

Високий показник влучності є критично важливим у випадках, де хибні позитивні результати можуть бути дорогими або шкідливими, наприклад, у медицині при діагностиці рідкісних захворювань.

Втім, дана метрика не враховує хибні негативні результати, що може бути проблемою у випадках, коли важливо не пропустити жодного позитивного випадку, наприклад, у скринінгу раку.

Таким чином, влучність є корисною метрикою для оцінки продуктивності класифікаційної моделі, особливо у випадках, де критично важливо зменшити кількість хибних позитивних результатів. Однак, для повної оцінки моделі, влучність часто використовують у парі з іншою метрикою – повнотою.

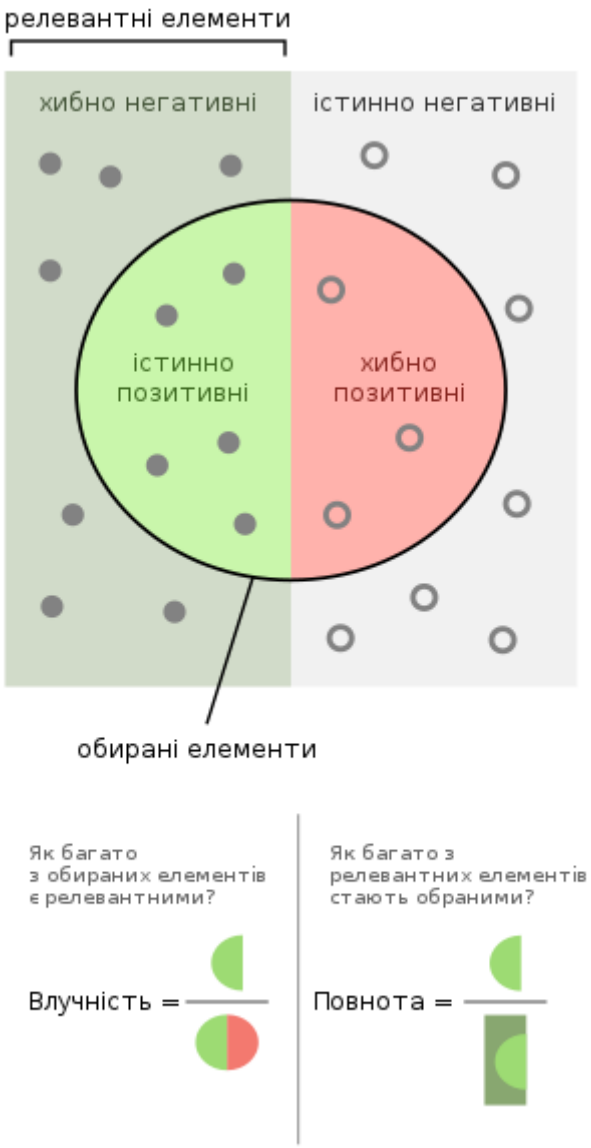

Рис. 1.14 – Візуалізація влучності та повноти

1.5.4 Повнота (Recall)

Повнота, як і влучність, також є основною метрикою для оцінки класифікаційних моделей. Вона визначає наскільки добре модель виявляє всі позитивні зразки, обчислюючи відсоток істинних позитивних передбачень серед усіх позитивних зразків.

$$Recall = \frac{TP}{TP + FN} \tag{1.26}$$

, де TP – кількість істинно позитивних випадків, а FN – кількість хибно негативних випадків.

Високий показник повноти є критично важливим у випадках, де пропускання позитивних випадків може мати серйозні наслідки, наприклад, у медичних діагнозах, де важливо виявити всі випадки захворювання.

Однак, дана метрика не враховує хибні позитивні результати, що може бути проблемою у випадках, коли важливо мати їх низьку кількість, наприклад, у фільтрах спаму, де важливо не блокувати легітимні повідомлення.

Таким чином, повнота є важливою метрикою для оцінки продуктивності класифікаційної моделі, особливо у випадках, де критично важливо мінімізувати кількість хибних негативних результатів.

1.5.5 F_1-міра (F_1-score)

F_1-міра є гармонійним середнім між влучністю та повнотою і використовується для забезпечення збалансованої оцінки продуктивності моделі.

$$F_1 = 2\frac{Precision \cdot Recall}{Precision + Recall} \tag{1.27}$$

, де $Precision$ – це влучність, а $Recall$ – повнота.

F_1-міра поєднує влучність і повноту в одну метрику, забезпечуючи збалансовану оцінку продуктивності моделі. Це корисно, коли важливо мати як високу влучність, так і високу повноту. Високе значення F_1-міри свідчить про те, що модель добре балансує між влучністю і повнотою, зменшуючи як хибні позитивні, так і хибні негативні результати. Це особливо важливо у випадках, де обидва типи помилок можуть мати значні наслідки.

Проте, вона може приховувати важливі аспекти продуктивності моделі, такі як абсолютна кількість позитивних чи негативних зразків, наприклад, дві моделі з однаковою F_1-мірою можуть мати дуже різну кількість хибних позитивних та хибних негативних результатів.

Таким чином, F_1-міра є важливою метрикою для оцінки продуктивності класифікаційної моделі, особливо у випадках, де існує дисбаланс між кількістю позитивних і негативних класів. Вона забезпечує збалансовану оцінку

поєднуючи влучність і повноту, але, як і для будь-якої іншої метрики, для повної оцінки моделі її слід використовувати разом з іншими метриками.

1.5.6 Крива робочої характеристики приймача (ROC Curve)

ROC (Receiver Operating Characteristic) крива є графічним представленням продуктивності класифікаційної моделі, показуючи співвідношення між істинним позитивним показником (TPR) та хибним позитивним показником (FPR) при різних порогових значеннях.

TPR (True Positive Rate) визначається як

$$TPR = \frac{TP}{TP + FN} \tag{1.28}$$

, де TP – кількість істинно позитивних випадків, а FN – кількість хибно негативних випадків.

FPR (False Positive Rate) визначається як

$$FPR = \frac{FP}{FP + TN} \tag{1.29}$$

, де FP – кількість хибно позитивних випадків, а TN – кількість істинно негативних випадків.

Крива ROC дозволяє візуально оцінити здатність моделі розрізняти позитивні та негативні зразки. Вона будується за допомогою графіків $TPR(FPR)$ для різних порогових значень. Наближеність до верхнього лівого кута графіка свідчить про вищу ефективність моделі.

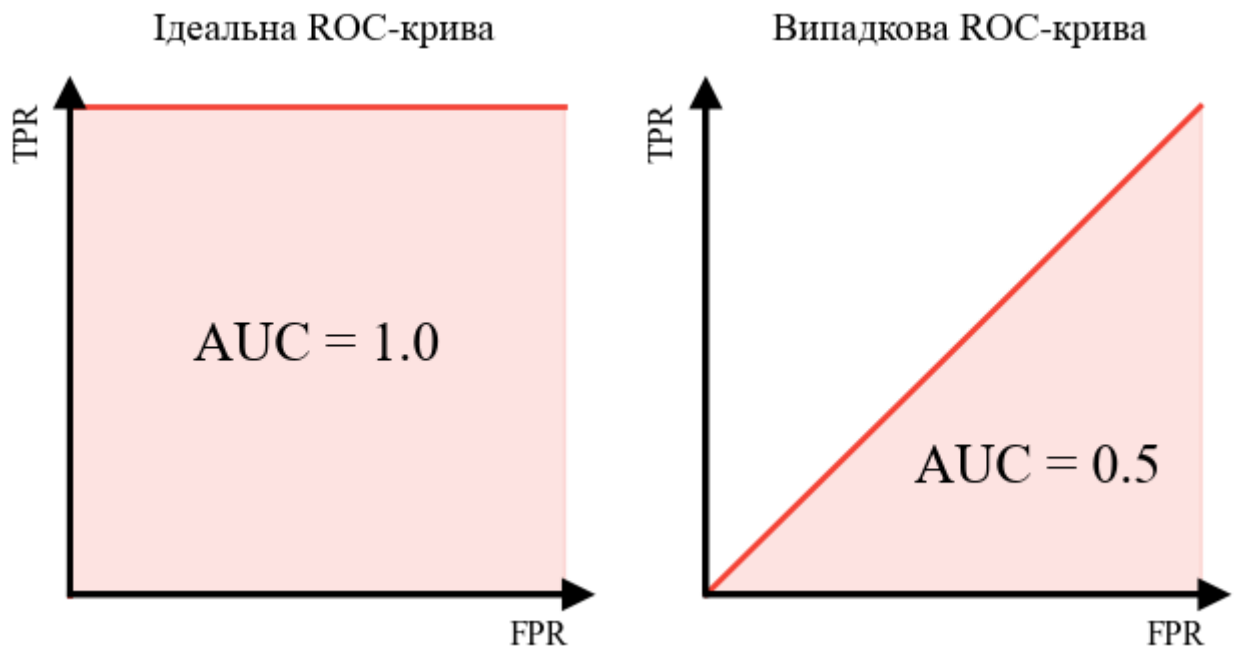

Рис. 1.15 – Приклад ідеальної і випадкової ROC-кривої

Ідеальна модель матиме точку в верхньому лівому куті ($TPR = 1$, $FPR = 0$), що означає 100% точність у визначенні позитивних зразків та відсутність хибних позитивних результатів. Випадкова модель відображається діагональною лінією від $(0; 0)$ до $(1; 1)$, що означає, що модель не розрізняє позитивні та негативні зразки краще, ніж випадкове вгадування. Загальна продуктивність моделі також може бути оцінена за допомогою показника AUC (Area Under Curve).

1.5.7 Площа під кривою робочої характеристики приймача (AUC)

AUC (Area Under the Curve) вимірює якість класифікаційної моделі, обчислюючи площу під кривою робочої характеристики приймача (ROC).

Площа під кривою визначається як

$$AUC = \int_0^1 TPR(FPR)d(FPR) \tag{1.30}$$

, де TPR – істинний позитивний показник, FPR – хибний позитивний показник.

AUC забезпечує всебічну оцінку продуктивності моделі, враховуючи всі можливі порогові значення для класифікації. Вона є стійкою до незбалансованих наборів даних, оскільки оцінює продуктивність моделі

незалежно від розподілу класів. ROC крива та AUC надають можливість візуальної оцінки продуктивності, що дозволяє легко порівнювати різні моделі.

Не зважаючи на це, у деяких специфічних випадках, таких як задачі з дуже високими витратами на хибні позитивні або хибні негативні результати, AUC може не надати достатньо точної інформації про продуктивність моделі.

Таким чином, AUC є потужною метрикою для оцінки класифікаційних моделей, забезпечуючи всебічну оцінку їхньої здатності розрізняти позитивні та негативні класи. Вона особливо корисна у випадках з незбалансованими наборами даних.

1.5.8 Підсумок

Вибір метрики є важливим етапом оцінки продуктивності моделі. Різні метрики підходять для різних задач і типів даних. Знання переваг та недоліків кожної метрики дозволяє більш точно оцінювати моделі та оптимізувати їхню ефективність для конкретних завдань. Використання комплексного підходу до оцінки, який включає кілька метрик, забезпечує більш повну картину продуктивності моделі і допомагає виявити сильні та слабкі сторони підходів, що використовуються. Це дозволяє приймати більш обґрунтовані рішення при виборі та налаштуванні моделей машинного навчання для досягнення найкращих результатів.

РОЗДІЛ 2 МАШИННЕ ВТІЛЕННЯ ЗАДАЧІ

У цьому розділі описано комп’ютерне втілення задачі, яке охоплює кілька ключових етапів: вибір набору даних, підготовка нейронних моделей та їх навчання. Буде детально розглянуто огляд і попередню обробку даних, а також їх використання в згорткових нейронних мережах та системах навчання «з нуля».

2.1 Вибір набору даних

Для повноцінного втілення поставленої задачі класифікації зображень із мінімізацією зусиль на додаткове навчання, мені потрібен набір даних, котрий містить велику кількість різноманітних зображень різних класів, а також список атрибутів, які цим класам властиві. Під ці вимоги потрапляє набір «Тварини з атрибутами 2» («AwA2»), який є вдосконаленим і розширеним варіантом своєї однойменної першої версії.

У порівнянні з попередником, «AwA2» включає значно більше зображень та атрибутів: 37322 зображень формату JPEG, розділені на 50 видів тварин, а кожна тварина описується за допомогою 85 атрибутів. Для кожного зображення збережена інформація про його першоджерело та ліцензія на використання та поширення. Це робить його особливо корисним для застосування при навчанні «з нуля».

У структурі даних передбачена організація зображень у папки відповідно до класів тварин. До кожного класу додається значення його атрибутів у вигляді матриці двійкових або плаваючих значень, а перелік усіх класів разом із запропонованим розподілом класів на навчальні та тестові надаються у окремих текстових файлах. Це спрощує процес завантаження та обробки даних, роблячи AwA2 зручним у використанні. Завдяки цьому можна аналізувати та моделювати взаємозв’язки між атрибутами, вивчати ефективність різних алгоритмів класифікації та здійснювати перенесення знань для класифікації нових класів об'єктів.

Рис. 2.1 – Приклади зображень із набору даних

Однак, незважаючи на численні переваги, такі як великий обсяг даних та висока роздільна здатність зображень, AwA2 має деякі обмеження. Зокрема, можлива неоднорідність якості зображень та нечіткість деяких атрибутів для певних класів тварин, що може вплинути на точність моделей.

Загалом, набір даних AwA2 є потужним інструментом для досліджень у галузі машинного навчання. Його багатий набір атрибутів та значна кількість зображень роблять його ідеальним для експериментів з класифікації зображень та вивчення атрибутів.

2.2 Огляд та попередня обробка даних у програмному середовищі

Перед початком роботи я зчитую дані про класи: навчальні та тестові, про їх зв’язок із атрибутами, а також переглядаю список усіх ліцензій та залишаю тільки ті зображення, які доступні для вільного використання та розповсюдження.

Для обробки зображень я використав кілька корисних функцій, таких як створення каталогів для зберігання шляхів до змінених зображень, перевірка кольорової моделі зображення, зміна розміру зображень тощо.

Результатом обробки є створення словника, який асоціює класи із шляхами до відповідних зображень, виключаючи зображення з авторськими правами та невалідні зображення. Створений словник завантажується у робочу папку і потім використовується за потреби.

2.3 Підготовка даних до використання у CNN

Для обробки зображень згортковою нейронною мережею створимо словник класів зображень розміром 128 пікселів на 128 пікселів. При використанні цієї функції автоматично створюється папка із систематизованими зображеннями відповідного розміру. Словники використовуються з метою ефективного використання пам’яті, оскільки вони займають набагато менше простору, аніж, наприклад, числові масиви або матриці.

Зважаючи на нерівномірний розподіл зображень по класам (див. рис. 2.2), окремою функцією я збільшую об’єм вибірки завдяки незначній видозміні 40 попередньо обраних класів, які будуть приймати участь у майбутньому навчанні моделі CNN. Зображення по класам трансформовані рівномірно шляхом віддзеркалення, зміни яскравості, контрастності, кута нахилу, а також зміщення по осям.

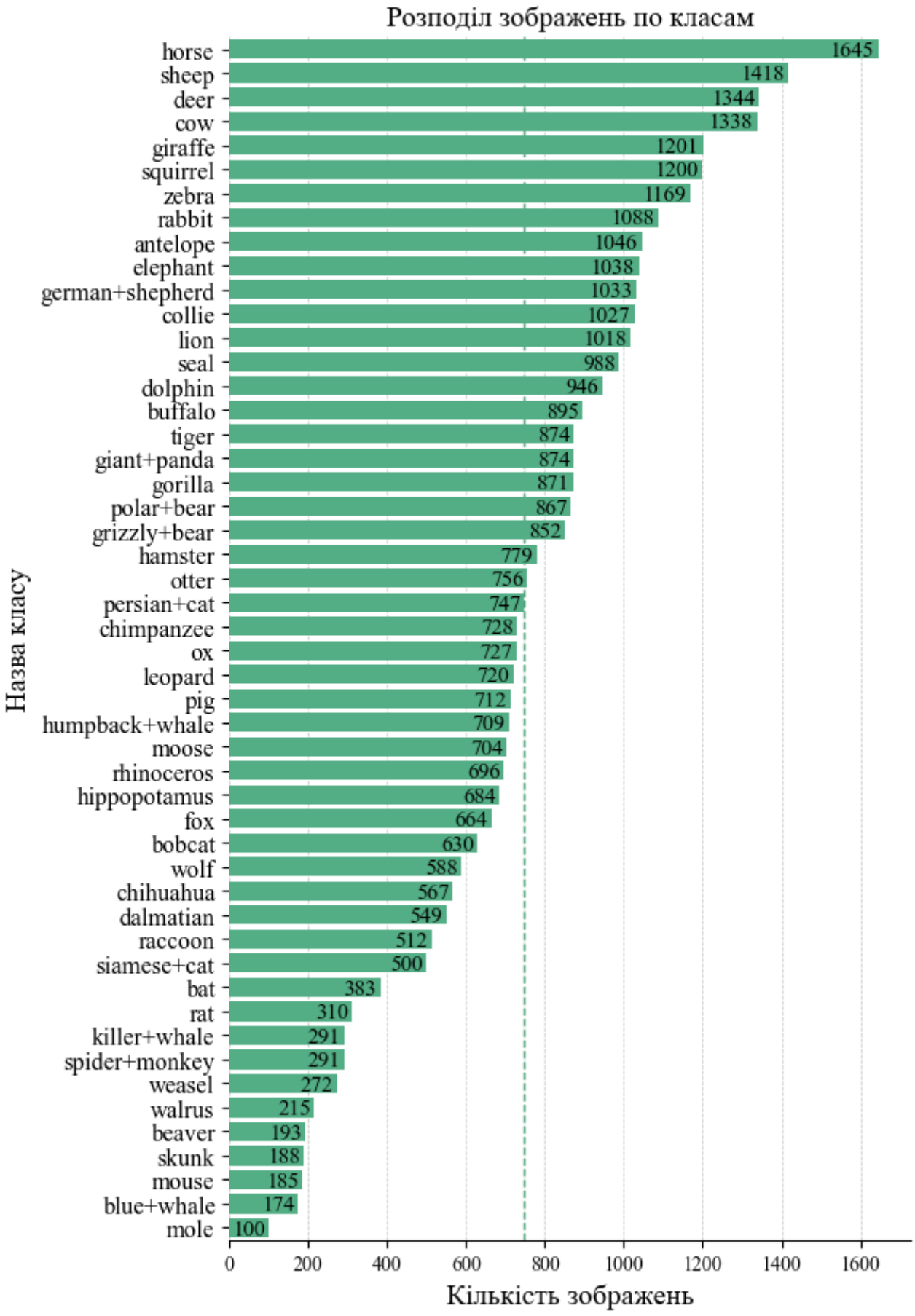

Рис. 2.2 – Розподіл зображень по класам

Наступним етапом є розподіл даних на бачені та небачені, який відбувається за поданими у наборі даних списками відповідних класів. Таким чином я отримую 40 бачених та 10 небачених класів. Однак, для навчання ЗНМ я, у випадковому порядку, групую зображення бачених класів у навчальну, валідаційну та тестову вибірки у відношенні $70\% - 10\% - 20\%$ відповідно.

Для перевірки та візуальної оцінки я виводжу графік розподілу зображень між навчальним, тестовим та валідаційним наборами. Діаграма показує кількість зображень у кожному класі для кожного з наборів з додаванням лінійних анотацій для полегшення інтерпретації даних.

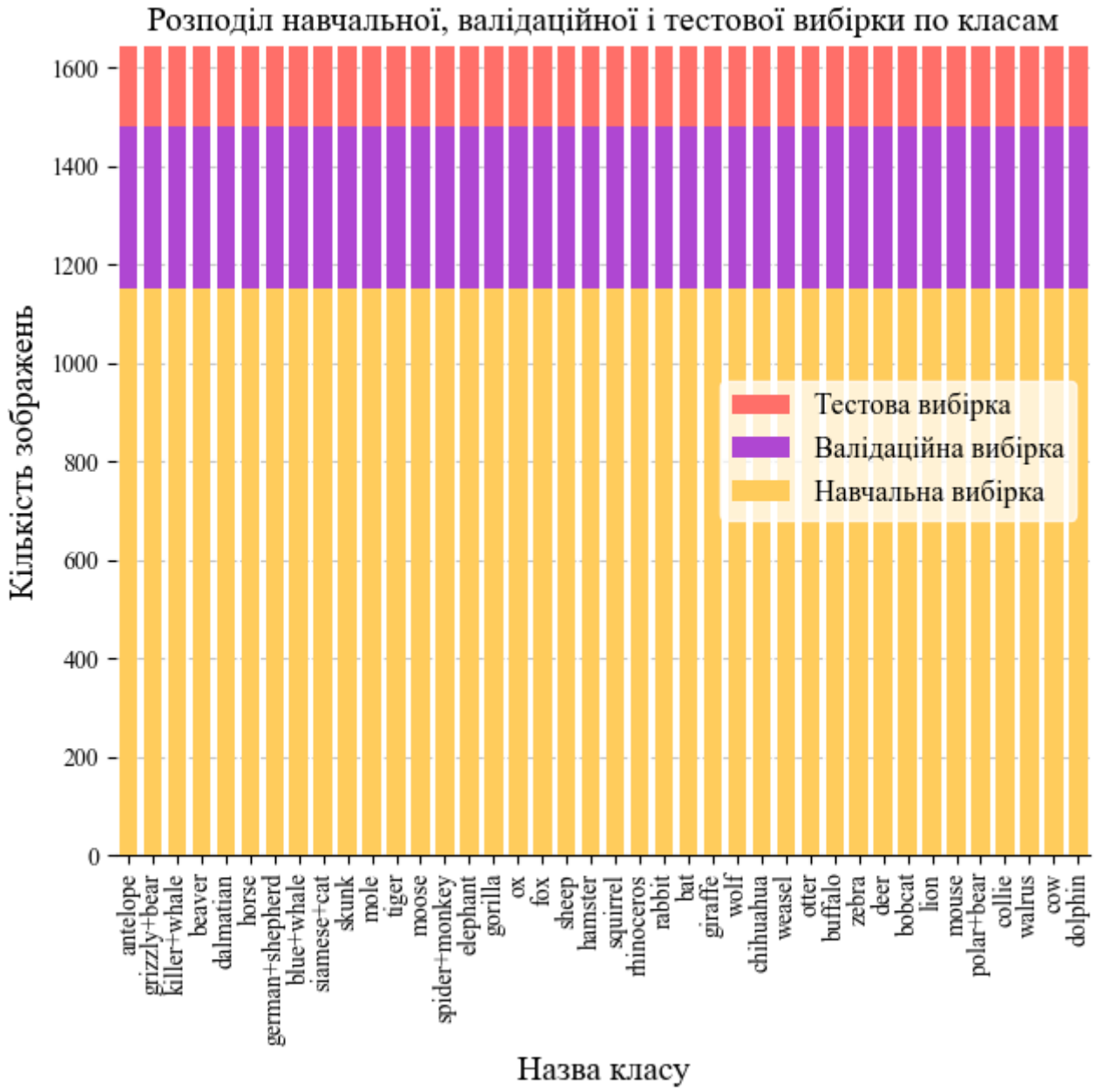

Рис. 2.3 – Розподіл навчальної, валідаційної і тестової вибірки по класам

2.4 Машинне втілення згорткової нейронної мережі

Як відомо, ЗНМ із великою кількістю прихованих шарів вимагає багато обчислювальних операцій із великими за розміром даними. Це означає, що цей процес вимагає значної кількості ресурсів, у тому числі і часу. Тому з метою пришвидшення роботи нейронної моделі я використовую обчислювальні потужності графічного процесора, однак за потреби переходжу до звичного застосування центрального процесора.

2.4.1 Створення генератора даних

Для глибоких нейронних мереж, як правило, застосовуються генератори даних. Їх основною функцією є створення пакетів зображень, які використовуються для навчання та оцінювання моделей. Це допомагає контрольовано подавати дані на модель, тим самим не перевантажуючи пам’ять обчислювального пристрою.

Окрім розподілу на пакети, клас генератора містить інші допоміжні, однак не менш важливі, методи. Наприклад, саме всередині нього відбувається пряме завантаження зображень із локальної папки, тасування даних, one-hot кодування номеру класу.

Для отримання результатів роботи моделі, генератор також містить методи передбачення для всіх зображень, які обчислюють точність моделі по класам на основі передбачених і фактичних даних з використанням матриці плутанини. Також є можливість видобутку ознак зображень на основі обчислених ваг моделі.

Для задання генераторів даних створюються три екземпляри: для навчального, валідаційного та тестового наборів даних. Використовується однаковий розмір пакету, який складається з 16 зображень.

2.4.2 Ініціалізація моделі CNN

При створенні нейронної моделі, спочатку я задаю їй архітектуру з п’ятьма згортковими шарами, нормалізацією пакетів, максимізаційними агрегувальними шарами, шаром сплющення та двома повноз’єднаними шарами.

Нова модель створюється з використанням активаційних функцій SELU та Softmax на виході, оптимізатора Adagrad із швидкістю навчання 0,001 та метрик: точності, AUC, влучності, повноти та F_1-міри. Використовуються два критерії для ранньої зупинки тренування на основі втрат на навчальному та валідаційному наборах. Після тренування модель та історія тренування зберігаються.

Цей код забезпечує гнучке створення, тренування та оцінку моделей згорткових нейронних мереж, використовуючи добре організований генератор даних та чітко визначену архітектуру моделі.

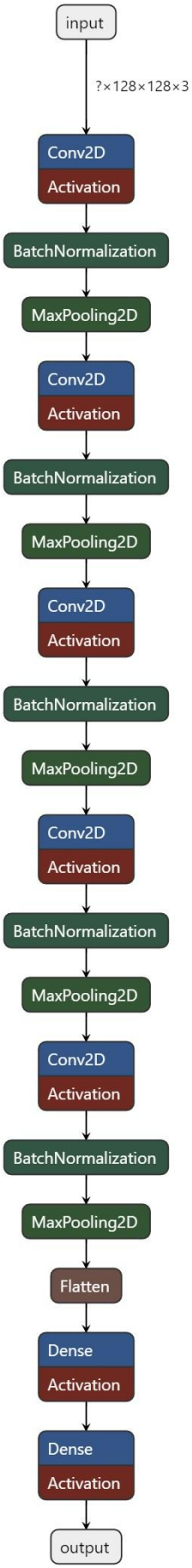

Рис. 2.4 – Архітектура моделі CNN

2.4.3 Виведення метрик оцінки моделі CNN

З метою візуальної інтерпретації метрик навчальної та валідаційної вибірки, я використовую спеціальну функцію, яка будує графік залежності метрики від епохи навчання. Для покращення сприйняття, лінії графіків для кожної вибірки мають різний колір: зелений – для навчальної, а червоний – для валідаційної. Значення метрики на найкращій епосі виділяється окремою точкою і додається в легенду.

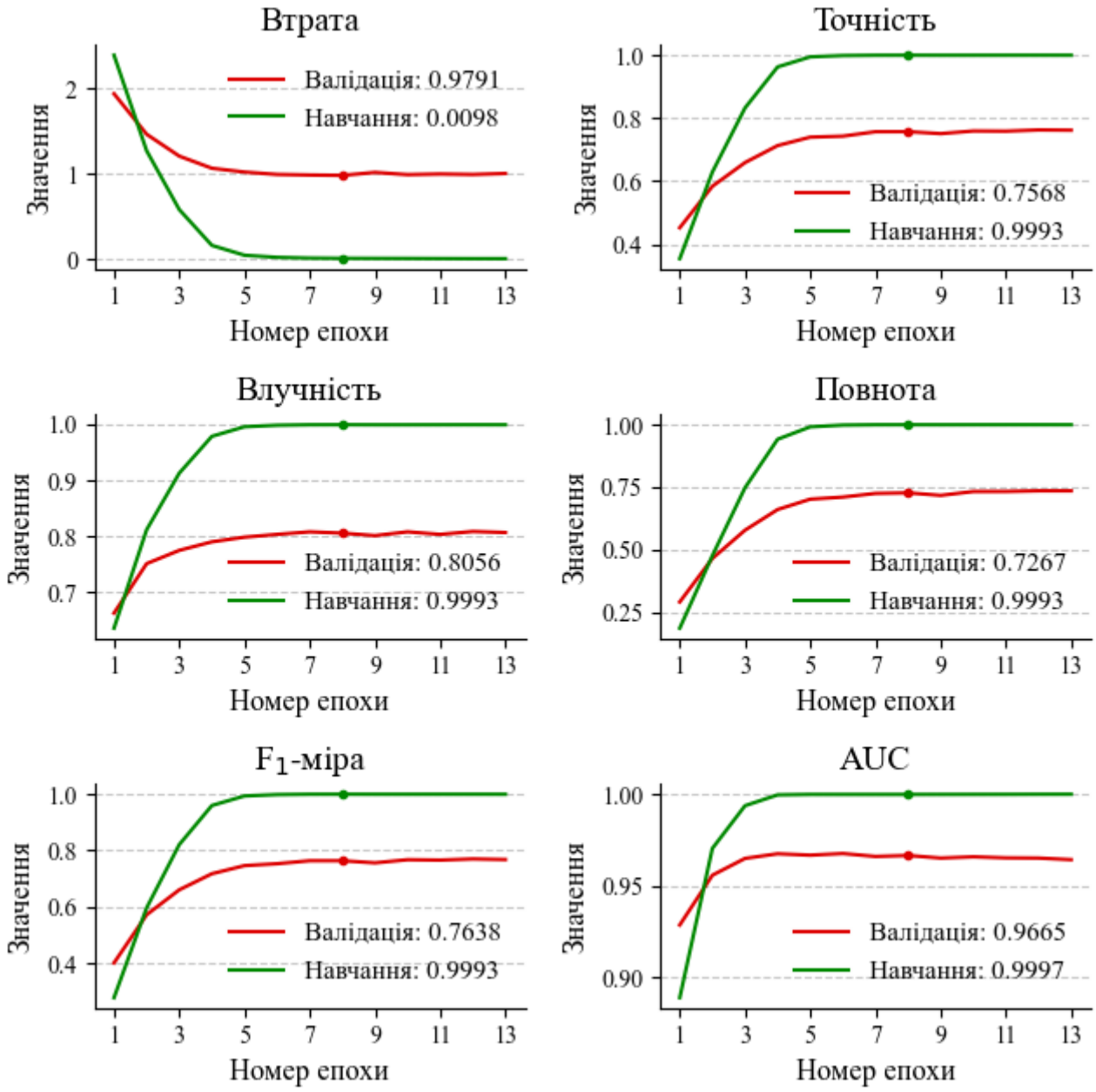

Рис. 2.5 – Значення метрик оцінки моделі CNN на кожній епосі

На графіку втрат видно, що значення на тренувальних даних дуже швидко знижуються до майже нуля, тоді як втрати на валідаційних даних

знижуються значно повільніше і стабілізуються на значно вищому рівні. Це може свідчити про перенавчання, коли модель добре підходить під тренувальні дані, але погано узагальнює на нові дані. Точність на тренувальних даних дуже висока, майже 100%, тоді як точність на валідаційних даних досягає лише приблизно 75%, що є ще одним показником перенавчання.

Показник AUC на тренувальних даних також дуже високий, майже 1.0, а на валідаційних даних трохи нижче, але все ще дуже високий, приблизно 0.97. Це свідчить про те, що модель дуже добре відрізняє класи на тренувальних даних, але трохи менш ефективно на валідаційних. Влучність на тренувальних даних знову ж таки дуже висока, а на валідаційних значно нижча, що свідчить про потенційне перенавчання.

Повнота на тренувальних даних майже 100%, тоді як на валідаційних даних близько 73%. Це також показує перенавчання. F_1-міра на тренувальних даних майже 100%, тоді як на валідаційних даних близько 76%. F_1-міра враховує як точність, так і повноту, і знову ж таки показує, що модель перенавчена.

Загалом, модель показує ознаки перенавчання, оскільки на тренувальних даних метрики значно кращі, ніж на валідаційних. Це може бути результатом занадто складної моделі або недостатньої кількості тренувальних даних. Щоб покращити узагальнення моделі на нові дані, можна спробувати застосувати регуляризацію, збільшити обсяг тренувальних даних, спрощувати модель тощо. Ці кроки можуть допомогти знизити перенавчання та покращити продуктивність моделі на валідаційних даних.

2.4.4 Підсумок

Отже, генератори даних допомагають ефективно керувати обсягом даних, використовуваних для навчання та оцінки моделей, забезпечуючи при цьому стабільну роботу без перевантаження пам'яті обчислювального пристрою. Створена модель ЗНМ має чітко визначену архітектуру та використовує ефективні активаційні функції та оптимізатори. Крім того, спеціальна функція

виводить метрики навчання та валідації у вигляді графіків для зручного аналізу результатів. Це означає, що модель для отримання ознак із зображень готова і можна переходити до наступного етапу – навчання «з нуля».

2.5 Підготовка даних до використання у ZSL

Початковий етап підготовки даних включає завантаження матриці відношень між класами і атрибутами з метою вивчення їхніх відношень за допомогою графіків. Це дозволяє краще зрозуміти структуру та взаємодію між класами зображень і їхніми атрибутами.

2.5.1 Візуалізація відношень класів зображень до атрибутів

Перший графік показує кількість атрибутів, які властиві кожному класу. Дані сортуються в порядку спадання, і для кожного класу будується стовпчик, що відображає кількість атрибутів (див. рис. 2.6).

Нерівномірність розподілу атрибутів по класах вказує на те, що деякі класи мають значно більше атрибутів, ніж інші, наприклад, горностай, шимпанзе і горила, що може свідчити про більш деталізований опис цих класів або про наявність більшої кількості інформації. Класи з меншою кількістю атрибутів, такі як синій кит та буйвол, можуть мати менш точне представлення, що може впливати на точність класифікації.

Другий графік показує кількість класів для кожного атрибута. Подібно до першого графіка, дані сортуються в порядку спадання, і будується стовпчик для кожного атрибута (див. рис. 2.7).

Нерівномірність кількості класів на атрибут свідчить про те, що деякі атрибути, такі як швидкий, має хвіст, гнучкий та активний, дуже поширені і зустрічаються у багатьох класах, що робить їх менш унікальними для конкретних класів. Інші атрибути, такі як живе в океані, має бивні і має плавці, зустрічаються дуже рідко, що робить їх більш унікальними, але можливо менш корисними для узагальнення.

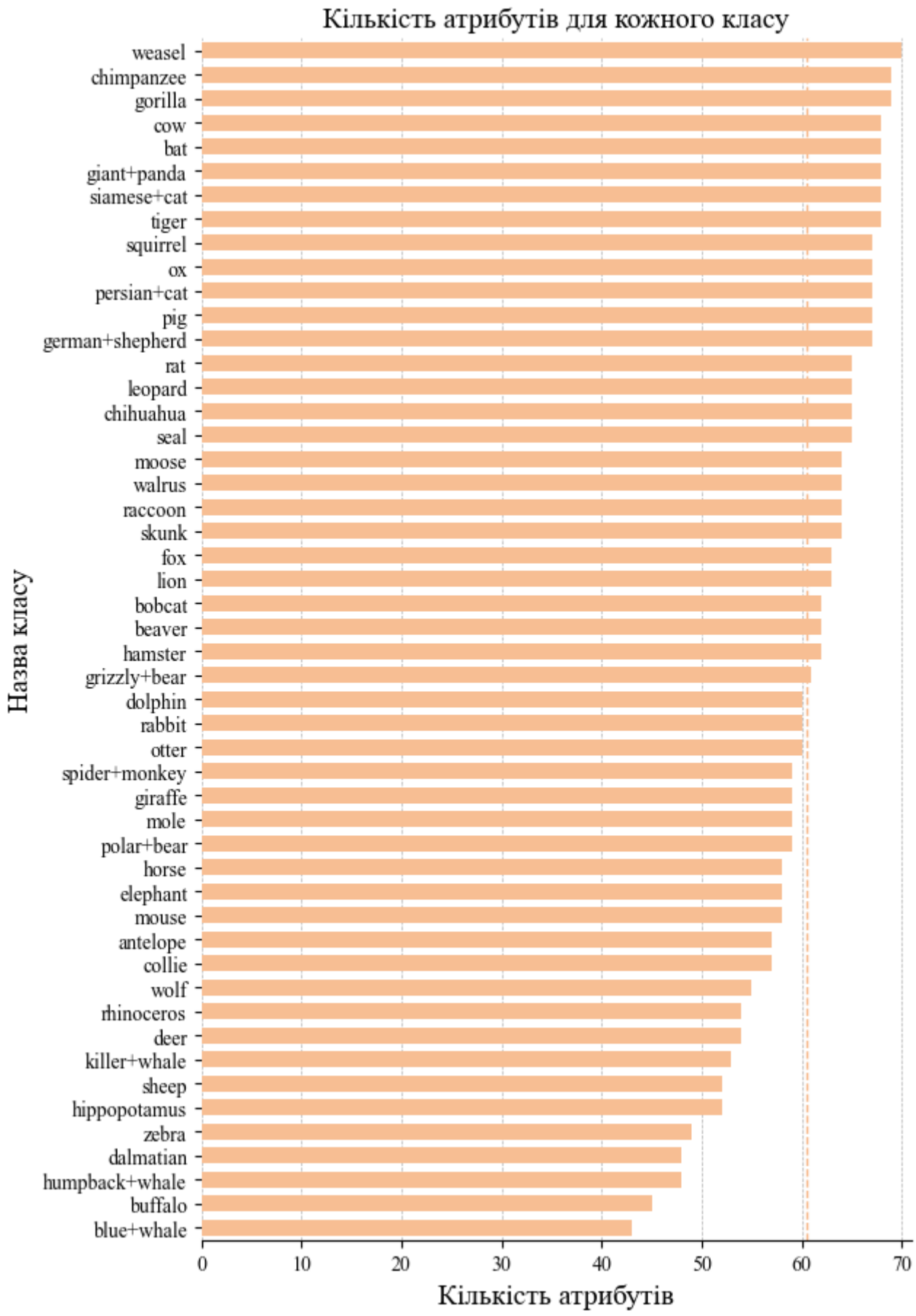

Рис. 2.6 – Кількість атрибутів для кожного класу

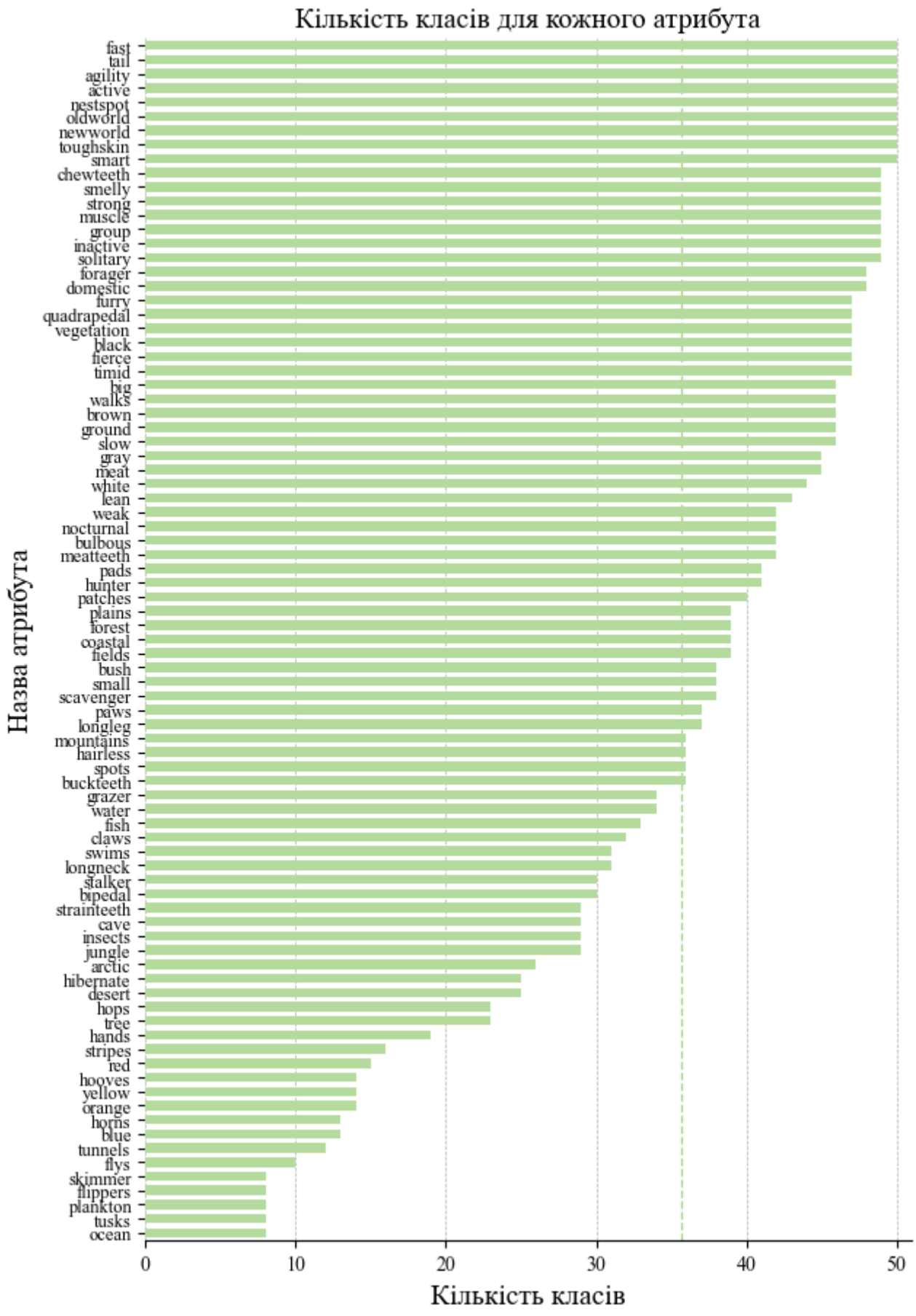

Рис. 2.7 – Кількість класів для кожного атрибута

Для покращення якості моделі слід розширити кількість атрибутів для класів з малою їх кількістю і забезпечити рівномірний розподіл атрибутів по класах. Також варто зосередитися на унікальних атрибутах для покращення розрізнення між класами та переглянути і, можливо, змінити або додати нові атрибути для більш точного опису класів. Ці кроки можуть допомогти поліпшити якість моделі і зменшити проблему перенавчання.

2.5.2 Витягування ознак зображень

Наступним кроком є перемішування та розподіл матриці на векторні подання класів та атрибутів. Ці вектори використовуються для аналізу бачених та небачених даних.

Потім, із використанням попередньо навченої моделі CNN, визначаються генератори даних для навчального та валідаційного наборів ознак зображень. Ознаки небачених зображень також отримуються на цьому етапі, однак при навчанні моделі не використовуються, а сприймаються як тестова вибірка.

2.6 Машинне втілення навчання «з нуля»

Наведений опис програми демонструє, як створити та оцінити ZSL-модель класифікації з використанням згорткової нейронної мережі. допомагає автоматично, яка допомагає виділяти значущі ознаки зображень для їх класифікації. У поєднанні з принципами ZSL, модель може класифікувати зображення в категорії, для яких не було прямих прикладів під час навчання, за допомогою семантичних векторів.

2.6.1 Створення генератора даних

Аналогічно до ЗНМ, ZSL також отримує велику перевагу від використання генераторів даних. У даному випадку він допомагає підготувати дані для навчання та тестування моделей ZSL.

Клас генератору даних для нової моделі хоч і має багато спільного із попереднім, однак має і свої вагомі відмінності. Наприклад, був доданий метод нормалізації ознак у межах від мінімального до максимального значення.

Для використання на тестовій вибірці існує метод, який генерує передбачення для всіх пакетів даних, використовуючи нейронну модель, та обробляє можливі помилки, пов’язані з використанням ресурсів системи. Також була успішно втілена класифікація передбачених результатів завдяки обчисленню косинусної подібності між передбаченими та фактичними векторами атрибутів та визначення точності класифікації для кожного класу.

Як і у попередньому випадку, для задання генераторів даних створюються три екземпляри: для навчального, валідаційного та небаченого наборів даних. Використовується однаковий розмір пакету, який складається з 32 зображень.

2.6.2 Ініціалізація моделі ZSL

На цьому етапі я описую архітектуру нової моделі, яка використовується для класифікації об’єктів, які не були представлені під час навчання. Вона має вхідний шар, два повноз’єднані шари з активаційною функцією SELU, і вихідний шар з активаційною функцією сигмоїди.

Параметри моделі, такі як розмір пакету 16, швидкість навчання 0,001, кількість епох 100, функція втрат бінарна крос-ентропія та метрики оцінки, визначаються перед компіляцією моделі. Також встановлюються критерії ранньої зупинки за втратами під час навчання або валідації. Після завершення навчання модель та історія навчання зберігаються у відповідні файли.

Описаний алгоритм забезпечує гнучку можливість створення, тренування та оцінки ZSL-моделей, використовуючи добре організований генератор даних та чітко визначену архітектуру моделі.

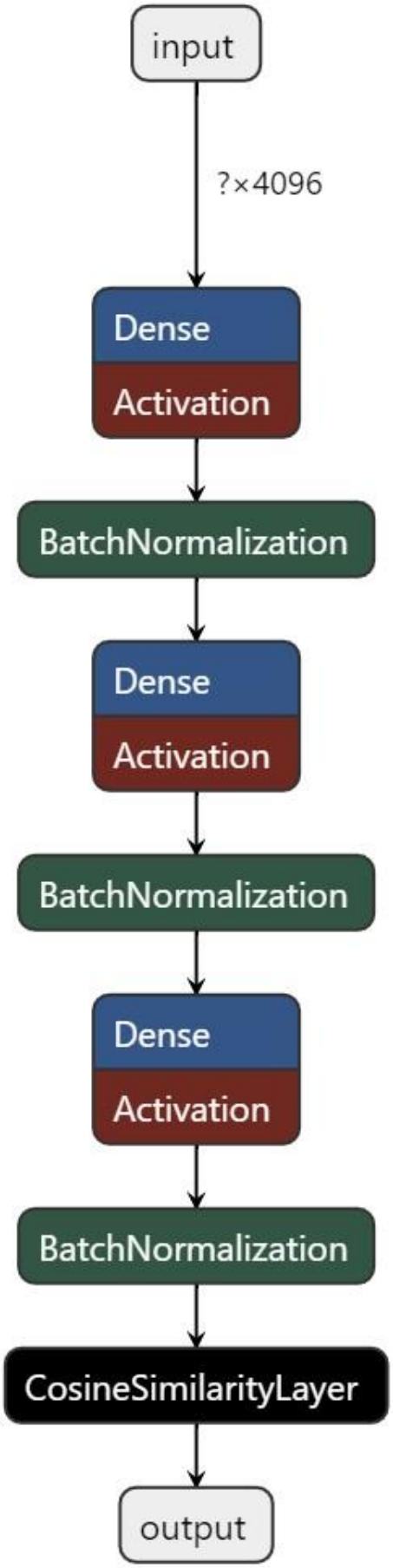

Рис. 2.8 – Архітектура моделі ZSL

2.6.3 Виведення метрик оцінки моделі ZSL

Виведення метрик моделі ZSL відбувається за допомогою тієї ж функції, що і для моделі CNN. Тому лінії графіків для кожної вибірки мають відповідні кольори: зелений – для навчальної, а червоний – для валідаційної. А значення метрики на останній епосі також додається в легенду.

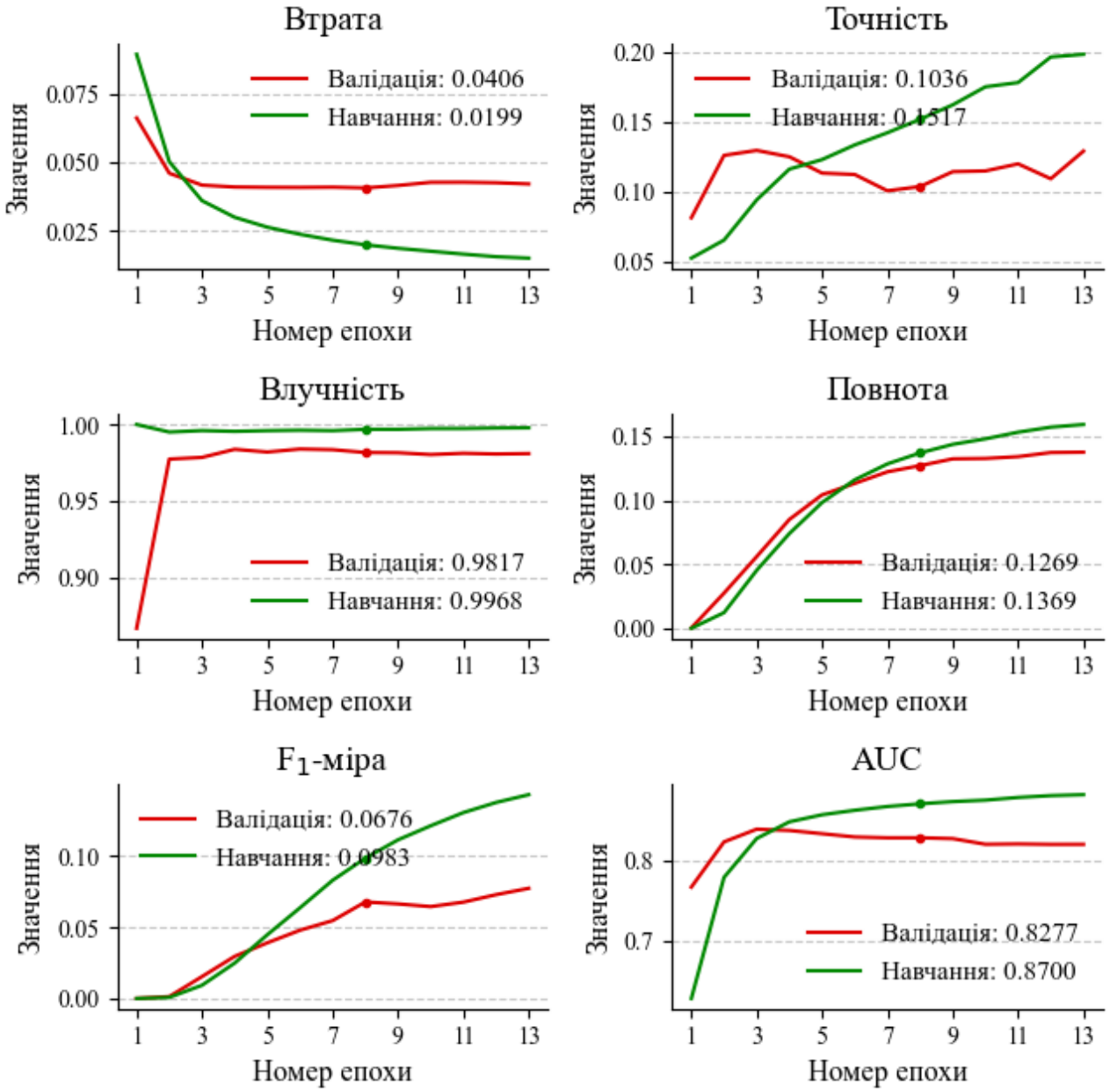

Рис. 2.9 – Значення метрик оцінки моделі ZSL на кожній епосі

Аналіз графіків показує, що втрата зменшується з кожною епохою для навчальних і валідаційних даних, але стабілізується на вищому рівні для валідації 0.0406 порівняно з навчальними даними 0.0199, що вказує на перенавчання моделі. Точність також зростає з кожною епохою, досягаючи

0.1517 на навчанні та 0.1036 на валідації. Влучність залишається майже незмінною і високою: 0.9968 на навчанні та 0.9817 на валідації, що свідчить про добру здатність моделі передбачати позитивні класи. Повнота та F1-міра також зростають, але з кращими показниками на навчальних даних, досягаючи 0.1369 та 0.0983 відповідно, в порівнянні з 0.1269 та 0.0676 на валідації. AUC також зростає, досягаючи 0.8700 на навчанні та 0.8277 на валідації.

Загалом, ознаки перенавчання моделі проявляються у значно кращих показниках на навчальних даних порівняно з валідаційними. Висока влучність свідчить про добру здатність передбачати позитивні приклади, але низькі значення повноти та F_1-міри на валідації вказують на можливість пропуску деяких позитивних прикладів. Для покращення узагальнюючої здатності моделі рекомендується розглянути методи регуляризації, зменшити складність моделі або зібрати більше даних для валідації. Це допоможе зменшити різницю між показниками на навчанні та валідації, підвищуючи загальну продуктивність моделі на нових даних.

2.6.4 Підсумок

Таким чином, описані програма і модель ZSL демонструють значний потенціал у класифікації об’єктів без прямих прикладів. Використання генераторів даних і чіткої архітектури моделі допомагає у підготовці та оцінці моделей. Однак, аналіз метрик виявляє необхідність у подальшому вдосконаленні для підвищення продуктивності та стабільності моделей. Готовність до проведення огляду та аналізу результатів обох моделей підкреслює важливість подальшого розвитку для поліпшення їх ефективності.

РОЗДІЛ 3 АНАЛІЗ РЕЗУЛЬТАТІВ ЕКСПЕРИМЕНТУ

У цьому розділі представлено детальний аналіз результатів експерименту, зокрема оцінку ефективності моделей глибокого навчання на основі нейронних мереж. Розглянуто точність класифікації, використовуючи різні візуалізації для полегшення розуміння результатів. Особливу увагу буде приділено порівнянню моделей CNN та ZSL, а також візуальному аналізу правильно та неправильно класифікованих зображень.

3.1 Аналіз результатів моделі CNN

Для оцінки точності моделі на кожному класі я також використовую генератор даних. Це допомагає зручно і ефективно зрозуміти, як добре модель виконує свої завдання для різних категорій.

Спочатку я створюю стовпчасту діаграму, де кожен стовпчик відповідає точності для певного класу. На графіку також відображається середня точність у вигляді пунктирної лінії. Значення точності додаються як анотації над стовпчиками, що дозволяє легко бачити точність кожного класу.

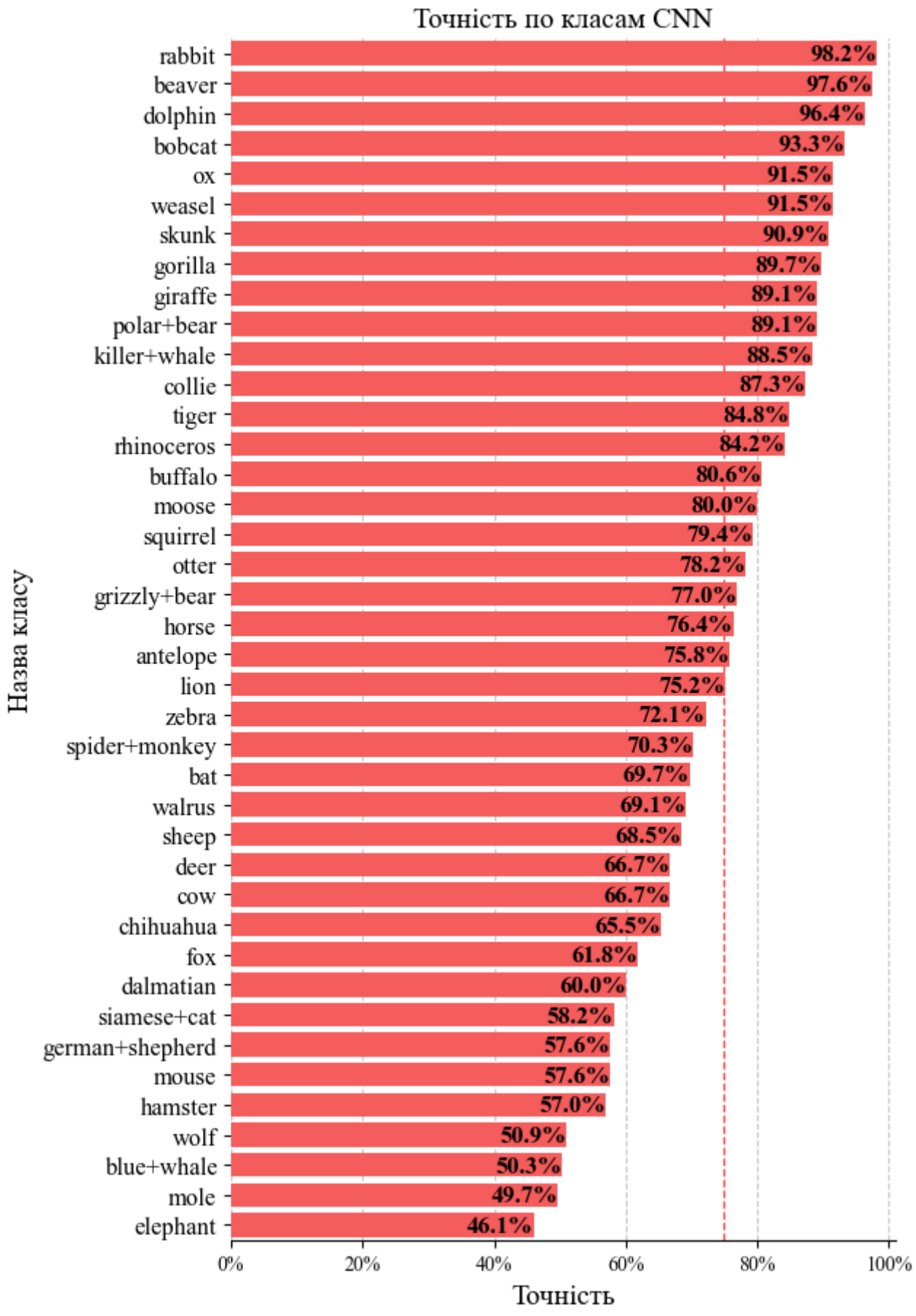

Рис. 3.1 – Точність по класам CNN

В цілому, результати показують непогану середню точність – 74.8%. Модель досягає найвищої точності (вище 90%) для наступних видів: кріль, бобер, дельфін, рись, віл, горностай, скунс. Це свідчить про те, що модель добре розпізнає ці класи, можливо, через чітко виражені особливості, які притаманні цим об'єктам.

Ще 15 класів мають прийнятну, більшу за середнє значення, точність, але все ще можуть потребувати додаткового покращення для досягнення вищих показників.

Найнижчу точність (нижче 60%) модель демонструє для класів, таких як сіамська кішка, німецька вівчарка, миша, хом’як, вовк, синій кит, кріт, слон. Це може свідчити про те, що модель плутає ці класи з іншими або має недостатньо даних для їх тренування.

Можливі причини низької точності для деяких класів включають нестачу даних, коли деякі класи представлені меншою кількістю прикладів у тренувальному наборі даних, схожість між класами, яка ускладнює їх розпізнавання, наприклад, вовк і німецька вівчарка, складність класифікації через високу варіативність зображень або менш виражені особливості, а також перенавчання моделі на певні класи, що знижує її здатність узагальнювати.

Для покращення продуктивності моделі варто звернути увагу на баланс даних, тонке налаштування моделі з використанням різних архітектур або налаштування гіперпараметрів, а також застосування методів регуляризації для запобігання перенавчанню. Загалом, модель демонструє хорошу точність для деяких класів, але має значні проблеми з іншими, що вимагає подальших заходів для покращення загальної продуктивності.

Потім я виводжу на графік випадкові зображення з класів, точності яких знаходяться у певному діапазоні. кожен стовпець відповідає класу, а кожен рядок – окремому зображенню з цього класу. Над кожним стовпцем відображається назва класу та його точність. Це дозволяє візуально оцінити якість зображень для класів, які беруть участь у навчанні моделі.

Рис. 3.2 – Приклади зображень із найкраще визначених класів

Рис. 3.3 – Приклади зображень із найгірше визначених класів

3.2 Аналіз результатів моделі ZSL

Наступний крок – побудова графіків точності класифікації зображень, які не приймали участь у навчанні. Для досягнення цієї мети я також використовую стовпчасту діаграму із підписами. На відміну він CNN, я використовую для аналізу результатів декілька варіантів. Наприклад, окрім звичної діаграми, заснованої на найкращому передбаченні, я ще будую графік, який показує точність для декількох найкращих передбачень.

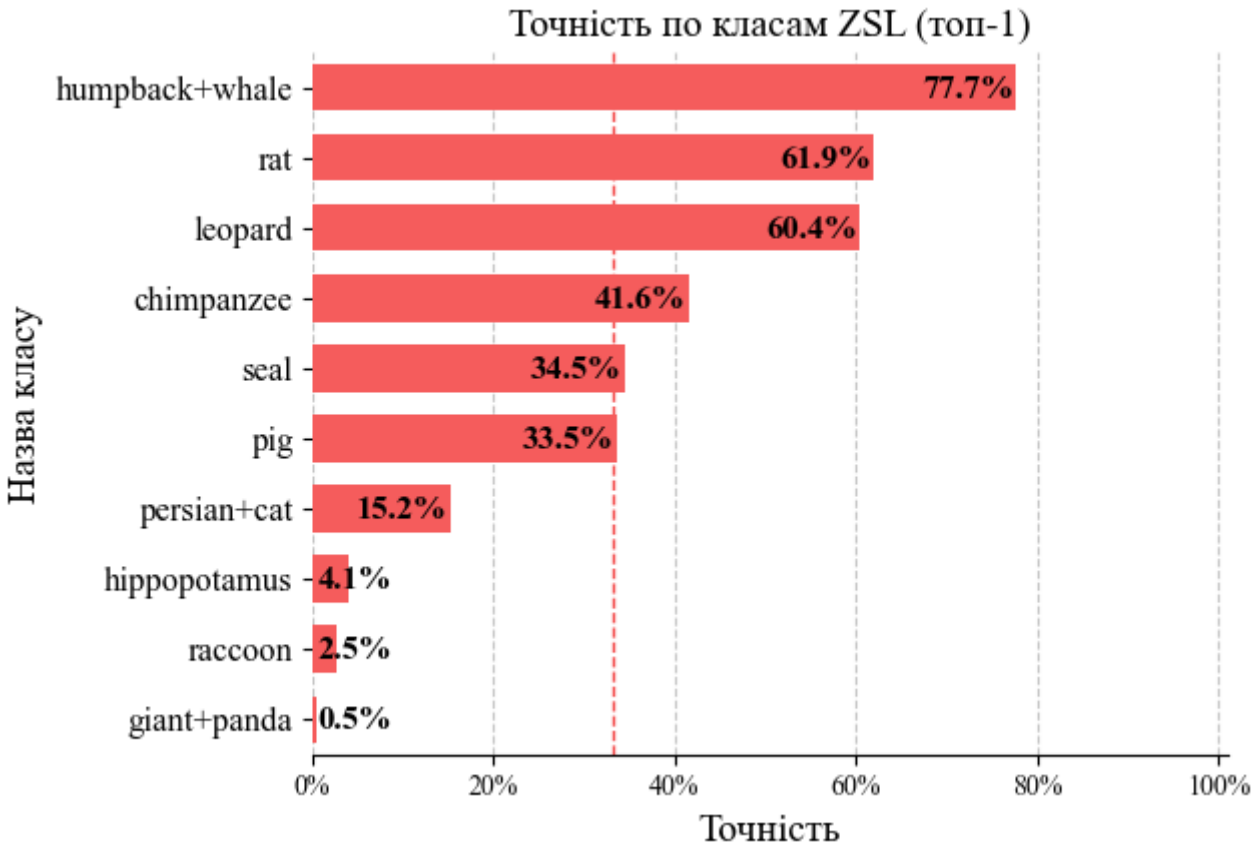

Рис. 3.4 – Точність по класам ZSL (топ-1)

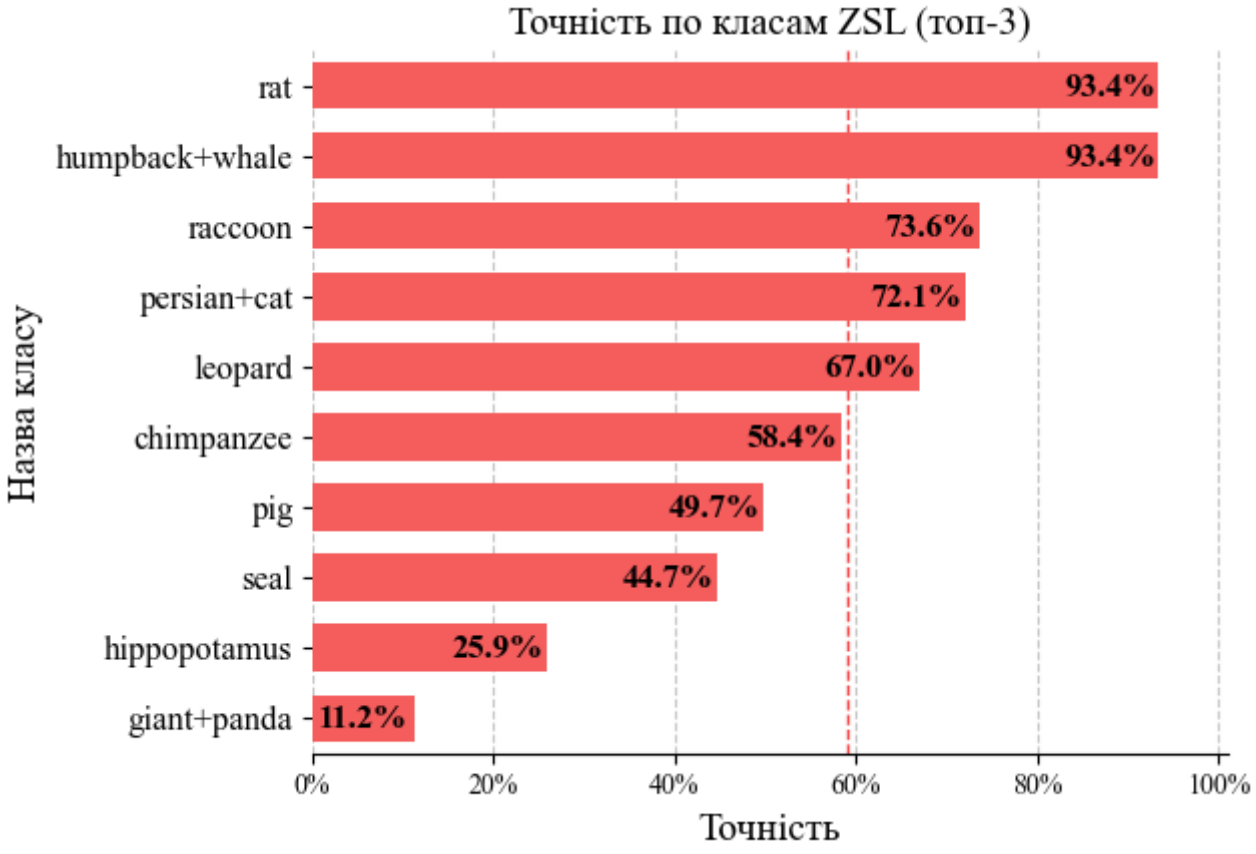

Рис. 3.5 – Точність по класам ZSL (топ-3)

На цих двох зображеннях представлені результати точності класифікації моделі ZSL для різних класів. Перший графік (див. рис. 3.4) показує точність моделі для кожного класу при використанні топ-1 прогноз, тобто коли правильний клас є першим передбаченням моделі. Для класу горбатий кит точність складає 77.7%, що є найвищим серед усіх класів. Клас пацюк показує друге за точністю передбачення з результатом 61.9%, а клас леопард – третій за точністю зі значенням 60.4%. А ось для видів бегемот, єнот, і велика панда точність дуже низька, складаючи 4.1%, 2.5%, і 0.5% відповідно.

Другий графік (див. рис. 3.5) демонструє точність моделі для кожного класу при використанні топ-3 прогнозів, тобто коли правильний клас є серед трьох перших передбачень моделі. Класи пацюк та горбатий кит знову мають найвищу точність – 93.4%. Для єнота точність значно підвищується до 73.6% порівняно з топ-1. Клас персидський кіт також показує значне покращення до 72.1%. Тоді як для виду бегемот точність підвищується до 25.9%, що все ще є низьким показником. Клас велика панда має точність 11.2%, що є покращенням, але все ще дуже низьким показником.

Загалом модель демонструє високу точність для трьох класів, що вказує на хорошу здатність моделі справлятися з цими класами. Однак інші класи мають дуже низьку точність, особливо в топ-1 прогнозах, що може свідчити про схожість цих класів з іншими класами. Точність для всіх класів значно покращується при переході від топ-1 до топ-3 прогнозів, що свідчить про те, що правильний клас часто є серед трьох перших передбачень моделі. Значна різниця в точності між класами вказує на те, що модель має проблеми з класифікацією деяких класів, що може бути покращено шляхом додаткового навчання моделі на більш збалансованих даних. Ці висновки допоможуть визначити, які аспекти моделі потребують покращення, та розробити стратегії для подальшого вдосконалення моделі ZSL.

Аналогічно до попереднього етапу, я також виводжу випадкові зображення із набору даних на екран.

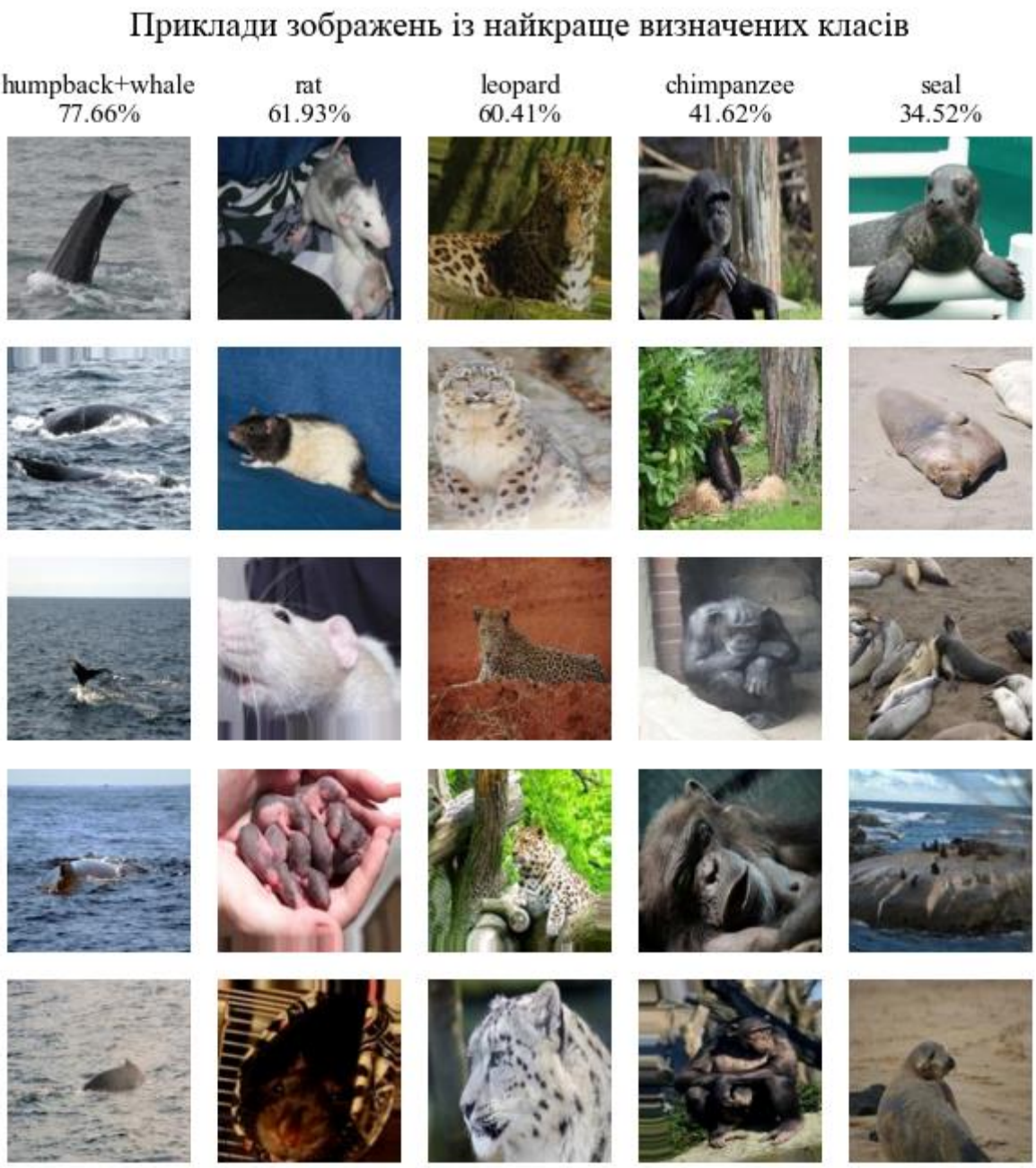

Рис. 3.6 – Приклади зображень із найкраще визначених класів

ВИСНОВОК

Розглянутий підхід до створення та оцінки навчання моделі CNN з функціональністю ZSL демонструє важливість інтеграції сучасних методів глибокого навчання для розв’язання складних задач класифікації. Використання генератора даних, нормалізація ознак та правильно налаштована архітектура CNN дозволяють досягти високих результатів у розпізнаванні зображень. Завдяки ZSL, модель може ефективно класифікувати нові, невідомі класи, використовуючи семантичні описи, що значно розширює її застосування в реальних умовах.

Оцінка моделі показала, що вона здатна до узагальнення знань на нові дані, що підтверджується результатами класифікації як відомих, так і невідомих класів. Візуалізація точностей для різних класів надає цінну інформацію для подальшого вдосконалення моделі. Такий підхід демонструє, як потужні інструменти глибокого навчання можуть бути використані для створення адаптивних і ефективних систем класифікації, що мають широке практичне значення у багатьох сферах, включаючи комп’ютерний зір, розпізнавання образів та інші області штучного інтелекту.

Таким чином, інтеграція CNN та ZSL у межах однієї моделі відкриває нові можливості для обробки та класифікації зображень, забезпечуючи високу точність і гнучкість у роботі з новими, невідомими класами. Цей підхід може служити основою для подальших досліджень і розвитку в галузях глибокого навчання та штучного інтелекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Zero-Shot learning – a comprehensive evaluation of the good, the bad and the ugly / Y. Xian et al. IEEE transactions on pattern analysis and machine intelligence. 2019. Vol. 41, no.9. P. 2251–2265. URL: https://doi.org/10.1109/tpami.2018.2857768 (date of access: 09.06.2024).

2. Харченко В. О. Моделювання нейронних мереж [Електронний ресурс] : навч. посіб. Суми : СумДУ, 2024. 263 с.

3. Lampert C. H., Nickisch H., Harmeling S. Learning to detect unseen object classes by between-class attribute transfer. 2009 IEEE conference on computer vision and pattern recognition (CVPR), Miami, FL, USA, 20–25 June 2009. 2009. URL: https://doi.org/10.1109/cvprw.2009.5206594 (date of access: 09.06.2024).

4. Lampert C. H., Nickisch H., Harmeling S. Attribute-Based classification for zero-shot visual object categorization. IEEE transactions on pattern analysis and machine intelligence. 2014. Vol. 36, no. 3. P. 453–465. URL: https://doi.org/10.1109/tpami.2013.140 (date of access: 09.06.2024).

5. M. Abadi, A. Agarwal, et al. TensorFlow: large-scale machine learning on heterogeneous distributed systems. URL: https://doi.org/10.48550/arXiv.1603.04467 (date of access: 09.06.2024).

6. Chollet F. Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek. MITP Verlags GmbH, 2018.

7. Skansi S. Convolutional neural networks. Undergraduate topics in computer science. Cham, 2018. P. 121–133. URL: https://doi.org/10.1007/978-3-319-73004-2_6 (date of access: 09.06.2024).

8. What are convolutional neural networks? | IBM. IBM - United States. URL: https://www.ibm.com/topics/convolutional-neural-networks (date of access: 09.06.2024).

9. Saha S. A comprehensive guide to convolutional neural networks – the ELI5 way. Medium. URL: https://towardsdatascience.com/a-comprehensive-guide-to-

convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (date of access: 09.06.2024).

10. Doshi K. Batch Norm Explained Visually–How it works, and why neural networks need it. Medium. URL: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739 (date of access: 09.06.2024).

11. Activation functions in neural networks [12 types & use cases]. V7 - Build Trustworthy AI at Scale | Automate Labeling. URL: https://www.v7labs.com/blog/neural-networks-activation-functions (date of access: 08.06.2024).

12. Chaudhary M. Activation functions: sigmoid, tanh, relu, leaky relu, softmax. Medium. URL: https://medium.com/@cmukesh8688/activation-functions-sigmoid-tanh-relu-leaky-relu-softmax-50d3778dcea5 (date of access: 08.06.2024).

13. Doshi S. Various optimization algorithms for training neural network. Medium. URL: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6 (date of access: 08.06.2024).

14. A comprehensive guide on optimizers in deep learning. Analytics Vidhya. URL: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/ (date of access: 08.06.2024).

15. Mishra A. Metrics to evaluate your machine learning algorithm. Medium. URL: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234 (date of access: 08.06.2024).

16. Roeder L. Netron: visualizer for neural network, deep learning and machine learning models. URL: https://www.lutzroeder.com/ai (date of access: 09.06.2024).

17. Wikimedia Commons. URL: https://upload.wikimedia.org/wikipedia/commons/thumb/0/03/Precisionrecall_uk.svg/330px-Precisionrecall_uk.svg.png (дата звернення: 09.06.2024).

18. Welcome – Computer Science Wiki. URL: https://computersciencewiki.org/images/8/8a/MaxpoolSample2.png (дата звернення: 09.06.2024).

ДОДАТОК

Лістинг програмного коду

```
!python -m ipykernel install --user --name tf --display-name "Python (tf)"
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import seaborn as sns
import random
import pickle
import time
import itertools
import csv
import os
import gc
import sys
import cv2
import warnings
from pathlib import Path
from collections import Counter
from PIL import Image
from matplotlib import rcParams
from matplotlib.ticker import FuncFormatter
from matplotlib.font_manager import FontProperties
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.linear_model import Ridge
from sklearn.preprocessing import LabelBinarizer
from scipy.spatial.distance import cdist
from scipy.ndimage import rotate, shift
from concurrent.futures import ThreadPoolExecutor
from keras import backend as K
from keras import Sequential, mixed_precision
from keras.models import load_model, Model
```

```
from keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dropout, Dense, BatchNormalization, Layer, LeakyReLU, Concatenate, Reshape, RepeatVector, GlobalAveragePooling2D, Lambda
from keras.optimizers import Adam, SGD, RMSprop, Adadelta, Adagrad, Adamax, Ftrl
from keras.metrics import Precision, Recall, AUC
from keras.initializers import Constant
from keras.activations import gelu
from keras.callbacks import EarlyStopping, Callback, ModelCheckpoint, ReduceLROnPlateau, LearningRateScheduler
from keras.utils import Sequence, to_categorical, load_img, img_to_array
from keras.regularizers import l2
rcParams['font.family'] = 'serif'
rcParams['font.serif'] = 'Times New Roman'
warnings.filterwarnings('ignore')
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    tf.config.experimental.set_memory_growth(gpus[0], True)
directory_path = Path("Animals_with_Attributes2")
directory_names = []
file_names = []
for item in directory_path.iterdir():
    if item.is_file():
        print("File:", item.name)
        file_names.append(item.name)
    elif item.is_dir():
        print("Directory:", item.name)
        directory_names.append(item.name)
def read_txt_file(file_name):
    file_path = directory_path / file_name
    try:
        with file_path.open(mode='r') as file:
            file_txt = file.read()
    except FileNotFoundError:
        print(f"File {file_path} not found or path is incorrect.")
        return []
    lines = file_txt.split('\n')
    file_data = []
```

```
    for line in lines:
        line = line.strip()
        if line:
            if '\t' in line:
                elements = line.split('\t')
                if len(elements) >= 2:
                    file_data.append(elements[1])
            else:
                file_data.append(line)
    return file_data
def read_and_print_data(file_name, data_name):
    data = read_txt_file(file_name)
    print(f"Number of {data_name} in '{file_name}': {len(data)}")
    print(f"{data_name}:", data)
    return data
classes = read_and_print_data("classes.txt", "classes")
testclasses = read_and_print_data("testclasses.txt", "testclasses")
trainclasses = read_and_print_data("trainclasses.txt", "trainclasses")
predicates = read_and_print_data("predicates.txt", "predicates")
testclasses = [cls_name for cls_name in classes if cls_name not in trainclasses]
def read_predicate_matrix(file_name):
    file_path = directory_path / file_name
    try:
        with open(file_path, 'r') as file:
            file_txt = file.read()
            lines = file_txt.strip().split('\n')
            matrix = np.array([list(map(float, line.split())) for line in lines])
            return matrix
    except FileNotFoundError:
        print(f"File {file_path} not found or path is incorrect.")
        return None
predicate_matrix_binary = read_predicate_matrix("predicate-matrix-binary.txt")
predicate_matrix_continuous = read_predicate_matrix("predicate-matrix-continuous.txt")
if predicate_matrix_binary is not None:
    print("Shape of predicate_matrix_binary:", predicate_matrix_binary.shape)
if predicate_matrix_continuous is not None:
    print("Shape of predicate_matrix_continuous:", predicate_matrix_continuous.shape)
```

```
def create_folder(size=(0, 0)):
    if size[0] == size[1]:
        folder_name = f"x{size[0]}"
    else:
        folder_name = f"{size[0]}x{size[1]}"
    folder_path = directory_path / 'cnn_images_data' / folder_name
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)
    return folder_path
def is_rgb(image_path):
    try:
        img = Image.open(image_path)
        return img.mode == 'RGB'
    except:
        return False
def modify_path(path, size):
    if not size == (0, 0):
        parts = str(path).split(os.sep)
        if size[0] == size[1]:
            added_index = "_x" + str(size[1])
        else:
            added_index = "_" + str(size[0]) + "x" + str(size[1])
        if len(parts) == 1:
            parts[0] = parts[0].split('_')[0] + added_index
        elif len(parts) >= 2:
            parts[1] = parts[1].split('_')[0] + added_index
            if len(parts) >= 3:
                parts[2] = parts[2].split('_')[0] + added_index
        return os.path.join(*parts)
    else:
        return path
def add_info_to_strings(filename, size):
    if not size == (0, 0):
        if size[0] == size[1]:
            new_filename = f"{filename.split('.')[0]}_x{size[0]}.{filename.split('.')[-1]}"
        else:
            new_filename = f"{filename.split('.')[0]}_m{size[0]}x{size[1]}.{filename.split('.')[-1]}"
```

```
        return new_filename
    else:
        return filename
def resize_images(dictionary_to_modify, size):
    created_count = 0
    existed_count = 0
    for class_name, image_paths in dictionary_to_modify.items():
        for image_path in image_paths:
            if not os.path.exists(image_path):
                print(f"File not found: {image_path}")
                continue
            modified_path = modify_path(image_path, size)
            if modified_path != image_path:
                if os.path.exists(modified_path):
                    existed_count += 1
                    continue
                image = Image.open(image_path)
                resized_image = image.resize(size)
                os.makedirs(os.path.dirname(modified_path), exist_ok=True)
                resized_image.save(modified_path)
                created_count += 1
    print(f"{created_count} images were created.")
    print(f"{existed_count} images already existed.")
def save_data(data, file_path):
    with open(file_path, "wb") as file:
        pickle.dump(data, file)
def load_data(file_path):
    with open(file_path, "rb") as file:
        return pickle.load(file)
licenses_data_folder = Path(directory_path / "licenses_data")
if not os.path.exists(licenses_data_folder):
    os.makedirs(licenses_data_folder)
dictionary_file_path = licenses_data_folder / "licenses_data_dictionary.pkl"
if dictionary_file_path.exists():
    licenses_data_dictionary = load_data(dictionary_file_path)
    unique_licenses = licenses_data_dictionary["unique_licenses"]
    files_with_empty_license = licenses_data_dictionary["files_with_empty_license"]
```

```
    copyrighted_files = licenses_data_dictionary["copyrighted_files"]
    free_to_use_files = licenses_data_dictionary["free_to_use_files"]
    print(f"Dictionary {dictionary_file_path} is loaded successfully.")
else:
    licenses_path = directory_path / "licenses"
    licenses_array = np.array([(os.path.relpath(root, licenses_path), len(files))
                               for root, _, files in os.walk(licenses_path) if root != str(licenses_path)],
                              dtype=object)
    unique_licenses = set()
    files_with_empty_license = []
    all_txt_files = True
    non_txt_files = []
    total_checked_files = 0
    copyrighted_files = []
    free_to_use_files = []
    for directory, count in licenses_array:
        licenses_directory_path = licenses_path / directory
        files = os.listdir(licenses_directory_path)
        total_checked_files += count
        for file in files:
            if not file.endswith('.txt'):
                all_txt_files = False
                non_txt_files.append(file)
            else:
                filename = os.path.splitext(file)[0]
                with open(licenses_directory_path / file, 'r', encoding='utf-8') as f:
                    for line in f:
                        if line.startswith("License      :"):
                            license_value = line.split(":")[1].strip()
                            if "(c)" in license_value:
                                copyrighted_files.append(filename)
                            elif not license_value or license_value.isspace():
                                files_with_empty_license.append(filename)
                            else:
                                license_value_cut = license_value.split(" (")[0]
                                unique_licenses.add(license_value_cut)
                                free_to_use_files.append(filename)
```

```
    licenses_data_dictionary = {
        "unique_licenses": unique_licenses,
        "files_with_empty_license": files_with_empty_license,
        "copyrighted_files": copyrighted_files,
        "free_to_use_files": free_to_use_files
    }
    save_data(licenses_data_dictionary, dictionary_file_path)
    print(f"Dictionary {dictionary_file_path} is saved successfully.")
    if all_txt_files:
        print("All", total_checked_files, "files in directories are .txt files.")
    else:
        print("Not all", total_checked_files, "files in directories are .txt files. Non .txt files:")
        for file in non_txt_files:
            print(file)
print("\nUtilized License Types:")
print(unique_licenses)
if files_with_empty_license:
    print("\nFiles with blank or whitespace-only License entries:")
    for file in files_with_empty_license:
        print(file)
else:
    print("\nNo files were found with blank or whitespace-only License entries.")
print("\nNumber of copyrighted files:", len(copyrighted_files))
print("Number of free-to-use files:", len(free_to_use_files))
images_name = "JPEGImages"
images_path = directory_path / images_name
images_directories_list = [item for item in os.listdir(images_path) if os.path.isdir(os.path.join(images_path, item))]
print("Number of directories:", len(images_directories_list))
print("List of directories:", images_directories_list)
def count_image_types(path):
    images_type_counter = Counter(
        os.path.splitext(file_name)[1].lower()
        for root, _, files in os.walk(path)
        for file_name in files
    )
    print(f"Unique image file types in {path}:")
    for file_type, count in images_type_counter.items():
```

```
        print(f"{file_type}: {count}")
count_image_types(images_path)
def create_class_images_dictionary(target_path, target_size=(0, 0)):
    target_path = Path(modify_path(target_path, target_size))
    print(target_path)
    data_folder_path = create_folder(target_size)
    dictionary_file_path = data_folder_path / add_info_to_strings("class_images_dictionary.pkl", target_size)
    if dictionary_file_path.exists():
        with open(dictionary_file_path, "rb") as file:
            loaded_dictionary = pickle.load(file)
        print(f"Dictionary {dictionary_file_path} is loaded successfully.")
        return loaded_dictionary
    print(f"Dictionary {dictionary_file_path} is not found. Creating new dictionary...")
    if not target_path.exists():
        resize_images(create_class_images_dictionary(target_path, (0, 0)), target_size)
    dictionary = {}
    excluded_copyrighted = 0
    excluded_non_rgb = 0
    for target_class in os.listdir(target_path):
        target_class_path = os.path.join(target_path, target_class)
        if os.path.isdir(target_class_path):
            for file_name in os.listdir(target_class_path):
                if file_name.endswith(('.jpg', '.jpeg', '.png')):
                    file_path = os.path.join(target_class_path, file_name)
                    if os.path.splitext(file_name)[0] in copyrighted_files:
                        excluded_copyrighted += 1
                        continue
                    if not is_rgb(file_path):
                        excluded_non_rgb += 1
                        continue
                    if target_class not in dictionary:
                        dictionary[target_class] = []
                    dictionary[target_class].append(file_path)
    total_images = sum(len(images) for images in dictionary.values())
    print("Number of copyrighted images excluded:", excluded_copyrighted)
    print("Number of non-RGB images excluded:", excluded_non_rgb)
    print("Number of images included:", total_images)
```

```
    dict_classes = [add_info_to_strings(c + '.', target_size).split('.')[0] for c in classes]
    sorted_dictionary = {k: dictionary[k] for k in dict_classes if k in dictionary}
    with open(dictionary_file_path, "wb") as file:
        pickle.dump(sorted_dictionary, file)
    print(f"Dictionary {dictionary_file_path} is saved successfully.")
    return sorted_dictionary
class_images_dictionary = create_class_images_dictionary(images_path)
def plot_image_distribution(class_images_dictionary):
    class_names = list(class_images_dictionary.keys())
    image_quantity = [len(paths) for paths in class_images_dictionary.values()]
    class_names, image_quantity = zip(*sorted(zip(class_names, image_quantity), key=lambda x: x[1], reverse=True))
    class_names = list(class_names)[::-1]
    image_quantity = list(image_quantity)[::-1]
    fig, ax = plt.subplots(figsize=(6.5, 9.25))
    fig.patch.set_facecolor('#ffffff')
    ax.set_facecolor('#ffffff')
    bars = ax.barh(class_names, image_quantity, color='#53ae85', zorder=3)
    for bar in bars:
        width = bar.get_width()
        if width >= 1000:
            ax.annotate(f'{width}',
                        xy=(width, bar.get_y() + bar.get_height() / 2),
                        xytext=(-24, -0.25),
                        textcoords="offset points",
                        ha='left', va='center', fontsize=11, color='black')
        elif width >= 100:
            ax.annotate(f'{width}',
                    xy=(width, bar.get_y() + bar.get_height() / 2),
                    xytext=(-18, -0.25),
                    textcoords="offset points",
                    ha='left', va='center', fontsize=11, color='black')
    mean_accuracy = np.mean(image_quantity)
    ax.axvline(mean_accuracy, color='#4a9c77', linestyle='--', linewidth=1)
    ax.set_xlabel('Кількість зображень', fontsize=14)
    ax.set_ylabel('Назва класу', fontsize=14)
    ax.set_title('Розподіл зображень по класам', fontsize=14)
    ax.set_ylim([-0.5, len(classes) - 0.5])
```

```
    ax.tick_params(axis='y', labelsize=12)
    ax.grid(axis='x', linestyle='--', linewidth=0.5, alpha=0.7, zorder=0)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    plt.tight_layout()
    plt.show()
plot_image_distribution(class_images_dictionary)
def show_random_images_from_classes_by_accuracy(images_dict, class_accuracies, accuracy_range,
num_images_per_class=5, word=''):
    class_names = list(images_dict.keys())
    min_accuracy = min(accuracy_range[0], np.max(class_accuracies))
    max_accuracy = max(accuracy_range[1], np.min(class_accuracies))
    classes_in_range = [(class_name, accuracy) for class_name, accuracy in zip(class_names, class_accuracies) if
accuracy >= min_accuracy and accuracy <= max_accuracy]
    sorted_classes = sorted(classes_in_range, key=lambda x: x[1], reverse=True)
    sorted_class_names = [item[0] for item in sorted_classes]
    if word == 'найкраще':
        sorted_classes = sorted_classes[:5]
        sorted_class_names = sorted_class_names[:5]
    elif word == 'найгірше':
        sorted_classes = sorted_classes[-5:]
        sorted_class_names = sorted_class_names[-5:]
    num_classes = len(sorted_classes)
    fig, axes = plt.subplots(num_images_per_class, num_classes, figsize=(6, 6.75))
    fig.suptitle(f'Приклади зображень із {word} визначених класів', fontsize=14)
    if num_images_per_class == 1:
        axes = axes[np.newaxis, :]
    if num_classes == 1:
        axes = axes[:, np.newaxis]
    for i, (sorted_class_names, class_accuracy) in enumerate(sorted_classes):
        class_images = images_dict[sorted_class_names]
        random_indices = random.sample(range(len(class_images)), num_images_per_class)
        axes[0, i].set_title(f'{sorted_class_names}\n{class_accuracy:.2f}%', fontsize=11)
        for j, idx in enumerate(random_indices):
            image_path = class_images[idx]
            image = cv2.imread(image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
            axes[j, i].imshow(image)
            axes[j, i].axis('off')
    plt.tight_layout()
    plt.show()
def show_random_image_from_classes(class_images_dictionary, num_classes=20):
  num_classes = min(num_classes, len(class_images_dictionary))
  random_classes = random.sample(list(class_images_dictionary.keys()), num_classes)
  fig, axes = plt.subplots(nrows=5, ncols=4, figsize=(6.5, 6.75))
  fig.suptitle(f'Приклади зображень із набору даних', fontsize=14)
  fig.patch.set_facecolor('#ffffff')
  for ax, class_name in zip(axes.flatten(), random_classes):
    image_paths = class_images_dictionary[class_name]
    random_image_path = random.choice(image_paths)
    with Image.open(random_image_path) as image:
      ax.imshow(image)
      ax.set_title(class_name.split('_')[0], fontsize=12)
      ax.axis('off')
  for ax in axes.flatten():
    if not ax.images:
      fig.delaxes(ax)
  plt.tight_layout()
  plt.show()
show_random_image_from_classes(class_images_dictionary)
class ImageAugmentor:
    def __init__(self, output_dir, target_size, num_workers=12):
        self.output_dir = output_dir
        self.num_workers = num_workers
        self.target_size = target_size
        os.makedirs(self.output_dir, exist_ok=True)
    def augment_and_save(self, data_dict):
        class_counts = {class_name: len(image_paths) for class_name, image_paths in data_dict.items()}
        max_images = max(class_counts.values())
        with ThreadPoolExecutor(max_workers=self.num_workers) as executor:
            futures = []
            for class_name, image_paths in data_dict.items():
                augment_count = max_images - class_counts[class_name]
                for i in range(augment_count):
```

```
                random_image_path = random.choice(image_paths)
                futures.append(executor.submit(self._process_image, class_name, random_image_path, i))
        for future in futures:
            future.result()
    def _process_image(self, class_name, image_path, index, ):
        img = cv2.imread(image_path)
        if img is not None:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            augmented_img = self.augment_image(img)
            class_output_dir = os.path.join(self.output_dir, modify_path(class_name, self.target_size))
            os.makedirs(class_output_dir, exist_ok=True)
            output_path = os.path.join(class_output_dir, f'aug_{index}.jpg')
            augmented_img_bgr = cv2.cvtColor(augmented_img, cv2.COLOR_RGB2BGR)
            cv2.imwrite(output_path, augmented_img_bgr)
        else:
            print(f"Warning: Image at path '{image_path}' could not be loaded.")
    def augment_image(self, image):
        augmentations = [
            self.random_flip,
            self.random_brightness,
            self.random_contrast,
            self.random_rotation,
            self.random_translation
        ]
        augmented_image = image
        for augment in augmentations:
            augmented_image = augment(augmented_image)
        return augmented_image
    def random_flip(self, image):
        if random.choice([True, False]):
            return np.flip(image, axis=1)
        return image
    def random_brightness(self, image):
        alpha = random.uniform(0.75, 1.25)
        return np.clip(image.astype(np.float32) * alpha, 0, 255).astype(np.uint8)
    def random_contrast(self, image):
        alpha = random.uniform(0.75, 1.25)
```

```
        return np.clip((image.astype(np.float32) - 127.5) * alpha + 127.5, 0, 255).astype(np.uint8)
    def random_rotation(self, image):
        angle = random.uniform(-15, 15)
        rotated_image = rotate(image, angle, reshape=False, mode='nearest')
        return np.clip(rotated_image, 0, 255).astype(np.uint8)
    def random_translation(self, image):
        height, width = image.shape[:2]
        x_translation = int(random.uniform(-0.1, 0.1) * width)
        y_translation = int(random.uniform(-0.1, 0.1) * height)
        translated_image = shift(image, (x_translation, y_translation, 0), mode='nearest')
        return np.clip(translated_image, 0, 255).astype(np.uint8)
def count_images_per_directory(output_dir):
    counts = {}
    for root, dirs, files in os.walk(output_dir):
        if files:
            directory_name = os.path.basename(root)
            counts[directory_name] = len(files)
    return counts
def delete_images_starting_with(output_dir, prefix):
    deleted_count = 0
    for root, dirs, files in os.walk(output_dir):
        for file in files:
            if file.startswith(prefix):
                file_path = os.path.join(root, file)
                os.remove(file_path)
                deleted_count += 1
    return deleted_count
target_size = (128, 128)
target_dictionary = create_class_images_dictionary(images_path, target_size)
data_folder_path = create_folder(target_size)
seen_dict= {}
unseen_dict = {}
for class_name, image_paths in target_dictionary.items():
    class_name = class_name.split('_')[0]
    if class_name in trainclasses:
        seen_dict[class_name] = image_paths
    elif class_name in testclasses:
```

```
        unseen_dict[class_name] = image_paths
def size_dict(dictionary):
    key_quantity = len(dictionary)
    value_quantity = sum(len(values) for values in dictionary.values())
    return key_quantity, value_quantity
def print_dict(dictionary):
    num_classes, total_images = size_dict(dictionary)
    details = ", ".join([f"{key}: {len(values)}" for key, values in dictionary.items()])
    print(f"Classes: {num_classes}, Images: {total_images}, Details: {details}")
print_dict(target_dictionary)
print_dict(seen_dict)
print_dict(unseen_dict)
def train_test_val_split(seen_dict, train_ratio=0.7, val_ratio=0.2, test_ratio=0.1):
    train_dict = {}
    val_dict = {}
    test_dict = {}
    for class_name, image_paths in seen_dict.items():
        random.shuffle(image_paths)
        total_images = len(image_paths)
        train_size = int(total_images * train_ratio)
        val_size = int(total_images * val_ratio)
        test_size = total_images - train_size - val_size

        train_images = image_paths[:train_size]
        val_images = image_paths[train_size:train_size + val_size]
        test_images = image_paths[train_size + val_size:]

        train_dict[class_name] = train_images
        val_dict[class_name] = val_images
        test_dict[class_name] = test_images
    return train_dict, val_dict, test_dict
train_dict, val_dict, test_dict = train_test_val_split(seen_dict)
def reduce_data(data_dict, reduction_ratio=0.2):
    reduced_dict = {}
    for class_name, image_paths in data_dict.items():
        reduced_size = int(len(image_paths) * reduction_ratio)
        reduced_images = random.sample(image_paths, reduced_size)
```

```
        reduced_dict[class_name] = reduced_images
    return reduced_dict
print_dict(train_dict)
print_dict(val_dict)
print_dict(test_dict)
def plot_data_distribution(train_dict, test_dict, val_dict):
    def extract_labels_and_counts(data_dict):
        categories = list(data_dict.keys())
        counts = [len(data_dict[category]) for category in categories]
        return np.array(categories), np.array(counts)
    train_categories, train_counts = extract_labels_and_counts(train_dict)
    test_categories, test_counts = extract_labels_and_counts(test_dict)
    val_categories, val_counts = extract_labels_and_counts(val_dict)
    val_categories, val_counts = zip(*sorted(zip(val_categories, val_counts), key=lambda x: x[1], reverse=True))
    sorted_train_counts = [train_counts[np.where(train_categories == category)[0][0]] if category in train_categories else 0 for category in val_categories]
    sorted_test_counts = [test_counts[np.where(test_categories == category)[0][0]] if category in test_categories else 0 for category in val_categories]
    bar_width = 0.75
    fig, ax = plt.subplots(figsize=(6.5, 6.5))
    fig.patch.set_facecolor('#ffffff')
    ax.set_facecolor('#ffffff')
    ax.bar(np.arange(len(val_categories)), sorted_test_counts, width=bar_width, color='#ff6f69', label='Тестова вибірка', bottom=np.array(sorted_train_counts) + np.array(val_counts), zorder=3)
    ax.bar(np.arange(len(val_categories)), val_counts, width=bar_width, color='#af47d2', label='Валідаційна вибірка', bottom=sorted_train_counts, zorder=3)
    ax.bar(np.arange(len(val_categories)), sorted_train_counts, width=bar_width, color='#ffcc5c', label='Навчальна вибірка', zorder=3)
    legend = ax.legend(loc='right', fontsize=12, frameon=True)
    legend.get_frame().set_facecolor('#ffffff')
    legend.get_frame().set_edgecolor('#ffffff')
    legend.get_frame().set_alpha(0.75)
    ax.set_xlabel('Назва класу', fontsize=14)
    ax.set_ylabel('Кількість зображень', fontsize=14)
    ax.set_title('Розподіл навчальної, валідаційної і тестової вибірки по класам', fontsize=14)
    ax.set_xlim([-0.5, len(val_categories) - 0.5])
    ax.set_ylim([0, 1645 + 1])
    ax.grid(axis='y', linestyle='--', alpha=0.7, zorder=0)
    ax.spines['top'].set_visible(False)
```

```
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    ax.set_xticks(range(len(val_categories)))
    ax.set_xticklabels([label for label in val_categories], rotation=90, fontsize=10)
    plt.tight_layout()
    plt.show()
plot_data_distribution(train_dict, test_dict, val_dict)
def f1_score(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1 = 2 * (precision * recall) / (precision + recall + K.epsilon())
    return f1
class DataGeneratorCNN(Sequence):
    def __init__(self, data_dict, batch_size, do_shuffle=True, num_workers=12):
        self.data_dict = data_dict
        self.batch_size = batch_size
        self.do_shuffle = do_shuffle
        self.num_workers = num_workers
        self._flatten_data()
        self.on_epoch_end()
    def __len__(self):
        return int(np.ceil(len(self.image_paths) / self.batch_size))
    def __getitem__(self, index):
        batch_indices = self.indices[index * self.batch_size:(index + 1) * self.batch_size]
        batch_paths = [self.image_paths[i] for i in batch_indices]
        batch_labels = [self.labels[i] for i in batch_indices]
        X = self._load_images(batch_paths)
        y = np.array(batch_labels)
        return X, y
    def on_epoch_end(self):
        if self.do_shuffle:
            np.random.shuffle(self.indices)
    def _flatten_data(self):
        self.image_paths = [path for paths in self.data_dict.values() for path in paths]
```

```
        self.labels = [label for label, paths in self.data_dict.items() for _ in paths]
        self.indices = np.arange(len(self.image_paths))
        lb = LabelBinarizer()
        self.labels = lb.fit_transform(self.labels)
    def _load_images(self, batch_paths):
        with ThreadPoolExecutor(max_workers=self.num_workers) as executor:
            images = list(executor.map(self._load_image, batch_paths))
        if images:
            return np.array(images)
        else:
            print("Warning: No images were loaded successfully. Returning an empty array.")
            return np.array([])
    def _load_image(self, path):
        img = cv2.imread(path)
        if img is not None:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = img / 255.0
        else:
            print(f"Warning: Image at path '{path}' could not be loaded. Removing it from the batch.")
            return None
        return img
    def generate_predictions(self, model):
        predictions = []
        with ThreadPoolExecutor(max_workers=self.num_workers) as executor:
            futures = [executor.submit(self._predict_batch, model, i) for i in range(len(self))]
            for future in futures:
                batch_predictions = future.result()
                predictions.extend(batch_predictions)
        return np.argmax(predictions, axis=1)
    def _predict_batch(self, model, index):
        images, _ = self[index]
        try:
            if tf.config.list_physical_devices('GPU'):
                with tf.device('/GPU:0'):
                    batch_predictions = model.predict(images, verbose=0)
            else:
                with tf.device('/CPU:0'):
```

```
                batch_predictions = model.predict(images, verbose=0)
        except tf.errors.ResourceExhaustedError:
            print(f"Error: Resource Exhausted on batch {index}. Falling back to CPU.")
            with tf.device('/CPU:0'):
                batch_predictions = model.predict(images, verbose=0)
        return batch_predictions
    def evaluate_model(self, model):
        predictions = self.generate_predictions(model)
        true_labels = np.argmax(self.labels, axis=1)
        conf_matrix = confusion_matrix(true_labels, predictions)
        class_accuracies = conf_matrix.diagonal() / conf_matrix.sum(axis=1) * 100
        return class_accuracies
    def extract_features(self, model):
        features = []
        with ThreadPoolExecutor(max_workers=self.num_workers) as executor:
            futures = [executor.submit(self._extract_batch_features, model, i) for i in range(len(self))]
            for future in futures:
                batch_features = future.result()
                features.extend(batch_features)
        return np.array(features)
    def _extract_batch_features(self, model, index):
        images, _ = self[index]
        try:
            if tf.config.list_physical_devices('GPU'):
                with tf.device('/GPU:0'):
                    batch_features = model.predict(images, verbose=0)
            else:
                with tf.device('/CPU:0'):
                    batch_features = model.predict(images, verbose=0)
        except tf.errors.ResourceExhaustedError:
            print(f"Error: Resource Exhausted on batch {index}. Falling back to CPU.")
            with tf.device('/CPU:0'):
                batch_features = model.predict(images, verbose=0)
        return batch_features
tf.config.experimental.get_memory_info('GPU:0')
batch_size = 12
train_generator = DataGeneratorCNN(train_dict, batch_size)
```

```
val_generator = DataGeneratorCNN(val_dict, batch_size)
test_generator = DataGeneratorCNN(test_dict, batch_size, do_shuffle=False)
def create_cnn_model(input_shape, output_shape, activation):
    model = Sequential([
        Input(shape=input_shape, name='input'),
        Conv2D(64, (3, 3), activation=activation, name='conv_1'),
        BatchNormalization(name='batchnorm_1'),
        MaxPooling2D((2, 2), strides=2, name='maxpool_1'),
        Conv2D(128, (3, 3), activation=activation, name='conv_2'),
        BatchNormalization(name='batchnorm_2'),
        MaxPooling2D((2, 2), strides=2, name='maxpool_2'),
        Conv2D(256, (3, 3), activation=activation, name='conv_3'),
        BatchNormalization(name='batchnorm_3'),
        MaxPooling2D((2, 2), strides=2, name='maxpool_3'),
        Conv2D(512, (3, 3), activation=activation, name='conv_4'),
        BatchNormalization(name='batchnorm_4'),
        MaxPooling2D((2, 2), strides=2, name='maxpool_4'),
        Conv2D(1024, (3, 3), activation=activation, name='conv_5'),
        BatchNormalization(name='batchnorm_5'),
        MaxPooling2D((2, 2), strides=2, name='maxpool_5'),
        Flatten(name='flatten'),
        Dense(4096, activation=activation, name='dense_1'),
        Dense(output_shape, activation='softmax', name='output')
    ], name='cnn_model')
    return model
K.clear_session()
trained_cnn_path = data_folder_path / add_info_to_strings('CNN_Model.keras', target_size)
history_cnn_path = data_folder_path / add_info_to_strings('CNN_Model_History.pkl', target_size)
if all(os.path.exists(path) for path in (trained_cnn_path, history_cnn_path)):
    cnn_model = load_model(trained_cnn_path, custom_objects={'f1_score': f1_score})
    print(f"Trained CNN model loaded successfully from {trained_cnn_path}.")
    history_cnn_dict = load_data(history_cnn_path)
    print(f"Training history dictionary loaded successfully from {history_cnn_path}.")
else:
    print("Trained model or history not found. Training a new model...")
    batch_size = 48
    learning_rate = 0.01
```

```
    epochs = 100
    activation = 'selu'
    loss = 'categorical_crossentropy'
    metrics = ['accuracy', AUC(name='auc'), Precision(name='precision'), Recall(name='recall'), f1_score]
    optimizer = Adagrad(lr=learning_rate)
    kernel_regularizer = l2(0.001)
    input_shape = (*target_size, 3)
    num_classes = len(seen_dict)
    print("GPU available, enabling GPU acceleration...") if gpus else print("No GPU available, using CPU...")
    cnn_model = create_cnn_model(input_shape, num_classes, activation)
    cnn_model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
    early_stopping_loss = EarlyStopping(monitor='loss', patience=5, restore_best_weights=True, min_delta=0,
mode='min', baseline=None, verbose=0)
    early_stopping_val_loss = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, min_delta=0,
mode='min', baseline=None, verbose=0)
    history_cnn = cnn_model.fit(train_generator, epochs=epochs, callbacks=[early_stopping_loss,
early_stopping_val_loss], validation_data=val_generator, verbose=1)
    if early_stopping_loss.stopped_epoch > 0:
        print(f"\nTraining stopped at epoch {early_stopping_loss.stopped_epoch + 1} due to training loss criteria.")
    elif early_stopping_val_loss.stopped_epoch > 0:
        print(f"\nTraining stopped at epoch {early_stopping_val_loss.stopped_epoch + 1} due to validation loss criteria.")
    else:
        print("\nTraining stopped due to completing all epochs.")
    cnn_model.save(trained_cnn_path)
    print(f"Trained CNN model saved successfully at {trained_cnn_path}.")
    history_cnn_dict = history_cnn.history
    save_data(history_cnn.history, history_cnn_path)
    print(f"Training history dictionary saved successfully at {history_cnn_path}.")
def plot_metrics(history_dict):
    best_epoch = history_dict['val_loss'].index(min(history_dict['val_loss']))
    fig, axs = plt.subplots(3, 2, figsize=(6.5, 6.5))
    fig.patch.set_facecolor('#ffffff')
    for row in axs:
        for ax in row:
            ax.set_facecolor('#ffffff')
    metrics_list = ['loss', 'accuracy', 'precision', 'recall', 'f1_score', 'auc']
    metrics_list_ua = ['Втрата', 'Точність', 'Влучність', 'Повнота', 'F$_1$-міра', 'AUC']
    for i, metric in enumerate(metrics_list):
```

```
        row_index = i // 2
        col_index = i % 2
        train_metric = history_dict[metric]
        val_metric = history_dict['val_' + metric]
        axs[row_index, col_index].plot(range(1, len(val_metric) + 1), val_metric, color='#dd0000', linewidth=1.5)
        axs[row_index, col_index].plot(range(1, len(train_metric) + 1), train_metric, color='#008800', linewidth=1.5)
        axs[row_index, col_index].set_xlabel('Номер епохи', fontsize=12)
        axs[row_index, col_index].set_ylabel("Значення", fontsize=12)
        axs[row_index, col_index].set_title(f'{metrics_list_ua[i]}', fontsize=14)
        train_best_value = train_metric[best_epoch]
        val_best_value = val_metric[best_epoch]
        axs[row_index, col_index].legend([f'Валідація: {val_best_value:.4f}', f'Навчання: {train_best_value:.4f}'],
loc='best', fontsize=11, frameon=False)
        axs[row_index, col_index].plot(best_epoch + 1, val_best_value, 'o', color='#dd0000', markersize=3, zorder=5)
        axs[row_index, col_index].plot(best_epoch + 1, train_best_value, 'o', color='#008800', markersize=3, zorder=5)
        axs[row_index, col_index].grid(axis='y', linestyle='--', alpha=0.7)
        axs[row_index, col_index].spines['top'].set_visible(False)
        axs[row_index, col_index].spines['right'].set_visible(False)
        axs[row_index, col_index].set_xticks(range(1, len(val_metric) + 1, 2))
    plt.tight_layout()
    plt.show()
plot_metrics(history_cnn_dict)
seen_class_accuracies = test_generator.evaluate_model(cnn_model)
def plot_seen_class_accuracies(class_accuracies, dictionary):
    sorted_indices = np.argsort(class_accuracies)[::-1]
    sorted_class_accuracies = [class_accuracies[i] for i in sorted_indices]
    sorted_x_labels = [list(dictionary.keys())[i] for i in sorted_indices]
    fig, ax = plt.subplots(figsize=(12, 6))
    fig.patch.set_facecolor('#f0f0f0')
    ax.set_facecolor('#f0f0f0')
    classes = np.arange(len(sorted_class_accuracies))
    bars = ax.bar(classes, sorted_class_accuracies, color='#f44336', zorder=3)
    mean_accuracy = np.mean(class_accuracies)
    ax.axhline(mean_accuracy, color='#f44336', linestyle='--', linewidth=2, label=f'Mean Accuracy:
{mean_accuracy:.1f}%')
    for bar in bars:
        height = bar.get_height()
        if height >= 19:
```

```
            ax.annotate(f'{height:.1f}%',
                        xy=(bar.get_x() + bar.get_width() / 2, height),
                        xytext=(1.25, -45),
                        textcoords="offset points",
                        ha='center', va='bottom', fontsize=12, rotation=90, fontweight='bold', color='black')
        else:
            ax.annotate(f'{height:.1f}%',
                        xy=(bar.get_x() + bar.get_width() / 2, 0),
                        xytext=(1.25, 4),
                        textcoords="offset points",
                        ha='center', va='bottom', fontsize=12, rotation=90, fontweight='bold', color='black')
    ax.set_xlabel('Classes', fontsize=16, fontweight='bold')
    ax.set_ylabel('Accuracy (%)', fontsize=16, fontweight='bold')
    ax.set_title('Accuracies by Class', fontsize=16, fontweight='bold')
    ax.set_xticks(classes)
    ax.set_xticklabels(sorted_x_labels, rotation=90, fontsize=11)
    ax.set_ylim([0, 100])
    ax.grid(axis='y', linestyle='--', alpha=0.7, zorder=0)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.legend(loc='best', frameon=False, prop=FontProperties(weight='bold', size=14))
    plt.tight_layout(rect=[0, 0, 1, 0.99])
    plt.show()
def plot_seen_class_accuracies(class_accuracies, dictionary, word=''):
    sorted_indices = np.argsort(class_accuracies)
    sorted_class_accuracies = [class_accuracies[i] for i in sorted_indices]
    sorted_x_labels = [list(dictionary.keys())[i] for i in sorted_indices]
    fig, ax = plt.subplots(figsize=(6.5, 9.25))
    sorted_indices = list(sorted_indices)
    sorted_class_accuracies = list(sorted_class_accuracies)
    fig.patch.set_facecolor('#ffffff')
    ax.set_facecolor('#ffffff')
    classes = np.arange(len(sorted_class_accuracies))
    bars = ax.barh(classes, sorted_class_accuracies, color='#f55c5c', zorder=3)
    mean_accuracy = np.mean(class_accuracies)
    print(mean_accuracy)
    ax.axvline(mean_accuracy, color='#f44a4a', linestyle='--', linewidth=1)
```

```
    for bar in bars:
        width = bar.get_width()
        if width >= 11:
            ax.annotate(f'{width:.1f}%',
                        xy=(width, bar.get_y() + bar.get_height() / 2),
                        xytext=(-33, -0.75),
                        textcoords="offset points",
                        ha='left', va='center', fontsize=12, fontweight='bold', color='black')
        else:
            ax.annotate(f'{width:.1f}%',
                    xy=(0, bar.get_y() + bar.get_height() / 2),
                    xytext=(2, -0.75),
                    textcoords="offset points",
                    ha='left', va='center', fontsize=12, fontweight='bold', color='black')
    ax.set_ylabel('Назва класу', fontsize=14)
    ax.set_xlabel('Точність', fontsize=14)
    ax.set_title(f'Точність по класам {word}', fontsize=14)
    ax.set_ylim([-0.5, len(classes) - 0.5])
    ax.set_xlim([0, 101])
    ax.set_yticks(classes)
    ax.set_yticklabels(sorted_x_labels, fontsize=12)
    ax.grid(axis='x', linestyle='--', alpha=0.7, zorder=0)
    formatter = FuncFormatter(lambda x, pos: f'{int(x)}%')
    ax.xaxis.set_major_formatter(formatter)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    plt.tight_layout()
    plt.show()
plot_seen_class_accuracies(seen_class_accuracies, seen_dict, 'CNN')
def show_random_images_from_classes_by_accuracy(images_dict, class_accuracies, accuracy_range,
num_images_per_class=5, word=''):
    class_names = list(images_dict.keys())
    min_accuracy = min(accuracy_range[0], np.max(class_accuracies))
    max_accuracy = max(accuracy_range[1], np.min(class_accuracies))
    classes_in_range = [(class_name, accuracy) for class_name, accuracy in zip(class_names, class_accuracies) if
accuracy >= min_accuracy and accuracy <= max_accuracy]
    sorted_classes = sorted(classes_in_range, key=lambda x: x[1], reverse=True)
```

```
    sorted_class_names = [item[0] for item in sorted_classes]
    if word == 'найкраще':
        sorted_classes = sorted_classes[:5]
        sorted_class_names = sorted_class_names[:5]
    elif word == 'найгірше':
        sorted_classes = sorted_classes[-5:]
        sorted_class_names = sorted_class_names[-5:]
    num_classes = len(sorted_classes)
    fig, axes = plt.subplots(num_images_per_class, num_classes, figsize=(6, 6.75))
    fig.suptitle(f'Приклади зображень із {word} визначених класів', fontsize=14)
    if num_images_per_class == 1:
        axes = axes[np.newaxis, :]
    if num_classes == 1:
        axes = axes[:, np.newaxis]
    for i, (sorted_class_names, class_accuracy) in enumerate(sorted_classes):
        class_images = images_dict[sorted_class_names]
        random_indices = random.sample(range(len(class_images)), num_images_per_class)
        axes[0, i].set_title(f'{sorted_class_names}\n{class_accuracy:.2f}%', fontsize=11)
        for j, idx in enumerate(random_indices):
            image_path = class_images[idx]
            image = cv2.imread(image_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            axes[j, i].imshow(image)
            axes[j, i].axis('off')
    plt.tight_layout()
    plt.show()
show_random_images_from_classes_by_accuracy(test_dict, seen_class_accuracies, accuracy_range=(0, 60),
word='найгірше')
show_random_images_from_classes_by_accuracy(test_dict, seen_class_accuracies, accuracy_range=(90, 100),
word='найкраще')
predicate_matrix_continuous = read_predicate_matrix("predicate-matrix-continuous.txt")
predicate_matrix_binary = read_predicate_matrix("predicate-matrix-binary.txt")
def replace_negatives_with_zero(matrix):
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            if matrix[i][j] < 0:
                matrix[i][j] = 0
replace_negatives_with_zero(predicate_matrix_continuous)
```

```
replace_negatives_with_zero(predicate_matrix_binary)
original_classes_labels = {index: label for index, label in enumerate(classes)}
original_predicates_labels = {index: label for index, label in enumerate(predicates)}
class_to_index = {v: k for k, v in original_classes_labels.items()}
seen_indices = [class_to_index[cls] for cls in trainclasses]
unseen_indices = [class_to_index[cls] for cls in testclasses]
seen_matrix = predicate_matrix_continuous[seen_indices, :]
unseen_matrix = predicate_matrix_continuous[unseen_indices, :]
print("Seen matrix shape:", seen_matrix.shape)
print("Unseen matrix shape:", unseen_matrix.shape)
def plot_class_attribute_count(matrix):
    if not isinstance(matrix, pd.DataFrame):
        matrix = pd.DataFrame(matrix)
    class_attributes = (matrix != 0).sum(axis=1)
    sorted_indices = np.argsort(class_attributes)[::-1]
    sorted_class_attributes = [class_attributes[i] for i in sorted_indices]
    sorted_x_labels = [original_classes_labels[label] for label in sorted_indices]
    sorted_class_attributes.reverse()
    sorted_x_labels.reverse()
    fig, ax = plt.subplots(figsize=(6.5, 9.25))
    fig.patch.set_facecolor('#ffffff')
    ax.set_facecolor('#ffffff')
    classes = np.arange(len(sorted_class_attributes))
    ax.barh(classes, sorted_class_attributes, color='#f7be93', height=0.667, zorder=3)
    mean_accuracy = np.mean(sorted_class_attributes)
    ax.axvline(mean_accuracy, color='#f7b787', linestyle='--', linewidth=1)
    ax.set_ylim([-0.5, len(classes) - 0.5])
    ax.set_ylabel('Назва класу', fontsize=14)
    ax.set_xlabel('Кількість атрибутів', fontsize=14)
    ax.set_title('Кількість атрибутів для кожного класу', fontsize=14)
    ax.set_yticks(range(len(classes)))
    ax.set_yticklabels(sorted_x_labels, fontsize=10)
    ax.set_xlim([0, max(class_attributes) + 1])
    ax.grid(axis='x', linestyle='--', linewidth=0.5, zorder=0)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
```

```
    plt.tight_layout()
    plt.show()
def plot_attribute_class_count(matrix):
    if not isinstance(matrix, pd.DataFrame):
        matrix = pd.DataFrame(matrix)
    attribute_classes = (matrix != 0).sum(axis=0)
    sorted_indices = np.argsort(attribute_classes)[::-1]
    sorted_attribute_classes = [attribute_classes[i] for i in sorted_indices]
    sorted_attribute_names = [original_predicates_labels[index] for index in sorted_indices]
    sorted_attribute_classes.reverse()
    sorted_attribute_names.reverse()
    fig, ax = plt.subplots(figsize=(6.5, 9.25))
    fig.patch.set_facecolor('#ffffff')
    ax.set_facecolor('#ffffff')
    attributes = np.arange(len(sorted_attribute_classes))
    ax.barh(attributes, sorted_attribute_classes, color='#b4db9d', height=0.667, zorder=3)
    mean_accuracy = np.mean(sorted_attribute_classes)
    ax.axvline(mean_accuracy, color='#acd793', linestyle='--', linewidth=1)
    ax.set_ylim([-0.5, len(attributes) - 0.5])
    ax.set_ylabel('Назва атрибута', fontsize=14)
    ax.set_xlabel('Кількість класів', fontsize=14)
    ax.set_title('Кількість класів для кожного атрибута', fontsize=14)
    ax.set_yticks(range(len(attributes)))
    ax.set_yticklabels(sorted_attribute_names, fontsize=9)
    ax.set_xlim([0, max(attribute_classes) + 1])
    ax.grid(axis='x', linestyle='--', linewidth=0.5, zorder=0)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    plt.tight_layout()
    plt.show()
plot_attribute_class_count(predicate_matrix_continuous)
def plot_attribute_image_count(matrix):
    attribute_images = []
    for predicate_index, predicate_column in enumerate(matrix.T):
        classes_with_predicate = [index for index, predicate_value in enumerate(predicate_column) if predicate_value]
        class_names_with_predicate = [original_classes_labels[index] for index in classes_with_predicate]
```

```
        total_images_for_predicate = sum(len(class_images_dictionary[class_name]) for class_name in
class_names_with_predicate)
        attribute_images.append(total_images_for_predicate)
    sorted_indices = np.argsort(attribute_images)[::-1]
    sorted_attribute_images = [attribute_images[i] for i in sorted_indices]
    sorted_attribute_names = [original_predicates_labels[index] for index in sorted_indices]
    fig, ax = plt.subplots(figsize=(12, 4))
    fig.patch.set_facecolor('#f0f0f0')
    ax.set_facecolor('#f0f0f0')
    attributes = np.arange(len(sorted_attribute_images))
    bars = ax.bar(attributes, sorted_attribute_images, color='#00bb88', zorder=3)
    for bar in bars:
        height = bar.get_height()
        if height >= 10000:
            ax.annotate(f'{height}',
                        xy=(bar.get_x() + bar.get_width() / 2, height),
                        xytext=(0.5, -32),
                        textcoords="offset points",
                        ha='center', va='bottom', fontsize=8, rotation=90, fontweight='bold', color='black')
        else:
            ax.annotate(f'{height}',
                        xy=(bar.get_x() + bar.get_width() / 2, height),
                        xytext=(1, -27),
                        textcoords="offset points",
                        ha='center', va='bottom', fontsize=8, rotation=90, fontweight='bold', color='black')
    ax.set_xlabel('Attributes', fontsize=16, fontweight='bold')
    ax.set_ylabel('Number of Images', fontsize=16, fontweight='bold')
    ax.set_title('Number of Images per Attribute', fontsize=16, fontweight='bold')
    ax.set_xticks(range(len(attributes)))
    ax.set_xticklabels(sorted_attribute_names, rotation=90, fontsize=8)
    ax.set_ylim([0, max(attribute_images) + 1])
    ax.grid(axis='y', linestyle='--', alpha=0.7, zorder=0)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    plt.tight_layout(rect=[0, 0, 1, 0.99])
    plt.show()
plot_attribute_image_count(predicate_matrix_continuous)
def min_max_normalize(arr):
```

```
    min_val = np.min(arr)
    max_val = np.max(arr)
    normalized_arr = (arr - min_val) / (max_val - min_val)
    return normalized_arr
train_indices = [class_to_index[cls] for cls in trainclasses]
test_indices = [class_to_index[cls] for cls in testclasses]
new_order = train_indices + test_indices
predicate_matrix = predicate_matrix_continuous[new_order, :]
predicates_vectors = np.array([predicate_matrix[:, j] for j in range(predicate_matrix.shape[1])])
classes_vectors = np.array([predicate_matrix[i, :] for i in range(predicate_matrix.shape[0])])
mean_val = np.mean(classes_vectors)
std_val = np.std(classes_vectors)
print(mean_val, std_val)
normalized_vectors = (classes_vectors - mean_val) / std_val
normalized_vectors = min_max_normalize(normalized_vectors)
seen_predicates_vectors = normalized_vectors[:40]
unseen_predicates_vectors = normalized_vectors[40:]
print(np.min(normalized_vectors), np.max(normalized_vectors))
cnn_model.trainable = False
feature_extractor_model = Model(inputs=cnn_model.input, outputs=cnn_model.get_layer('flatten').output)
batch_size = 12
seen_dict_r = reduce_data(seen_dict)
unseen_dict_r = reduce_data(unseen_dict)
seen_generator = DataGeneratorCNN(seen_dict, batch_size, do_shuffle=False)
unseen_generator = DataGeneratorCNN(unseen_dict, batch_size, do_shuffle=False)
seen_features = seen_generator.extract_features(feature_extractor_model)
print('Seen features are extracted successfully.')
unseen_features = unseen_generator.extract_features(feature_extractor_model)
print('Unseen features are extracted successfully.')
class DataGeneratorZSL(Sequence):
    def __init__(self, data_dict, data_features, predicates_vectors, batch_size, mean_value=None, std_value=None,
do_shuffle=True, num_workers=12):
        self.data_dict = data_dict
        self.data_features = data_features
        self.predicates_vectors = predicates_vectors
        self.batch_size = batch_size
        self.do_shuffle = do_shuffle
        self.mean_value = mean_value
```

```
        self.std_value = std_value
        self.num_workers = num_workers
        self.flat_indices = [(file_path, label) for label, file_paths in data_dict.items() for file_path in file_paths]
        self.on_epoch_end()
    def normalize_features(self, features, mean_value, std_value):
        if mean_value is not None and std_value is not None:
            return (features - mean_value) / std_value
        return features
    def __len__(self):
        return int(np.ceil(len(self.flat_indices) / self.batch_size))
    def __getitem__(self, index):
        batch_indices = self.flat_indices[index * self.batch_size:(index + 1) * self.batch_size]
        batch_features = [self.data_features[self.flat_indices.index(i)] for i in batch_indices]
        batch_features = [self.normalize_features(features, self.mean_value, self.std_value) for features in batch_features]
        batch_labels = [list(self.data_dict.keys()).index(i[1]) for i in batch_indices]
        for label_index in batch_labels:
            if label_index >= len(self.predicates_vectors):
                raise IndexError(f"Index {label_index} is out of bounds for axis 0 with size {len(self.predicates_vectors)}")
        batch_vectors = [self.predicates_vectors[label_index] for label_index in batch_labels]
        X = np.array(batch_features)
        y = np.array(batch_vectors)
        return X, y
    def on_epoch_end(self):
        if self.do_shuffle:
            combined = list(zip(self.flat_indices, self.data_features))
            random.shuffle(combined)
            self.flat_indices, self.data_features = zip(*combined)
    def generate_predictions(self, model):
        predictions = []
        with ThreadPoolExecutor(max_workers=self.num_workers) as executor:
            futures = [executor.submit(self._predict_batch, model, i) for i in range(len(self))]
            for future in futures:
                batch_predictions = future.result()
                predictions.extend(batch_predictions)
        return np.array(predictions)
    def _predict_batch(self, model, index):
        features, _ = self[index]
```

```
        try:
            if tf.config.list_physical_devices('GPU'):
                with tf.device('/GPU:0'):
                    batch_predictions = model.predict(features, verbose=0)
            else:
                with tf.device('/CPU:0'):
                    batch_predictions = model.predict(features, verbose=0)
        except tf.errors.ResourceExhaustedError:
            print(f"Error: Resource Exhausted on batch {index}. Falling back to CPU.")
            with tf.device('/CPU:0'):
                batch_predictions = model.predict(features, verbose=0)
        return batch_predictions
    def evaluate_model(self, model, top_ns=[1, 3, 5]):
        predictions = self.generate_predictions(model)
        accuracies = {}
        for top_n in top_ns:
            accuracies[f'top_{top_n}'] = self.classify_images(predictions, top_n=top_n)
        return accuracies
    def classify_images(self, Y_pred, top_n=1):
        accuracies = {cls: [] for cls in self.data_dict.keys()}
        for i, y in enumerate(Y_pred):
            similarities = cosine_similarity(
                tf.cast(y.reshape(1, -1), dtype=tf.float32),  # Ensure float type
                tf.cast(self.predicates_vectors, dtype=tf.float32)  # Ensure float type
            )
            top_n_indices = np.argsort(similarities[0])[-top_n:]  # Get the indices of the top-n predictions
            true_class = list(self.data_dict.keys()).index(self.flat_indices[i][1])
            if true_class in top_n_indices:
                accuracies[self.flat_indices[i][1]].append(1)
            else:
                accuracies[self.flat_indices[i][1]].append(0)
        mean_accuracies = {cls: np.mean(acc)*100 if acc else 0 for cls, acc in accuracies.items()}
        return mean_accuracies
all_features = np.concatenate((seen_features, unseen_features))
mean_value = np.mean(all_features)
std_value = np.std(all_features)
batch_size = 12
```

```
seen_generator_ZSL = DataGeneratorZSL(seen_dict, seen_features, seen_predicates_vectors, batch_size,
mean_value=mean_value, std_value=std_value, do_shuffle=True)

unseen_generator_ZSL = DataGeneratorZSL(unseen_dict, unseen_features, unseen_predicates_vectors, batch_size,
mean_value=mean_value, std_value=std_value, do_shuffle=True)

correlation_matrix = np.corrcoef(unseen_matrix.T, rowvar=False)

plt.figure(figsize=(12, 10))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', xticklabels=testclasses,

        yticklabels=testclasses, annot_kws={"fontsize": 16})

plt.show()

class CosineSimilarityLayer(Layer):

    def __init__(self, output_dim, **kwargs):

        self.output_dim = output_dim

        super(CosineSimilarityLayer, self).__init__(**kwargs)


    def build(self, input_shape):

        self.kernel = self.add_weight(name='kernel',

                                      shape=(input_shape[-1], self.output_dim),

                                      initializer='glorot_uniform',

                                      trainable=True)

        super(CosineSimilarityLayer, self).build(input_shape)

    def call(self, inputs):

        x = inputs

        x_normalized = K.l2_normalize(x, axis=-1)

        kernel_normalized = K.l2_normalize(self.kernel, axis=0)

        cosine_similarity = K.dot(x_normalized, kernel_normalized)

        return cosine_similarity

    def compute_output_shape(self, input_shape):

        return (input_shape[0], self.output_dim)

    def get_config(self):

        config = super(CosineSimilarityLayer, self).get_config()

        config.update({"output_dim": self.output_dim})

        return config

def create_zsl_model(input_shape, output_shape, activation, kernel_regularizer):

    model = Sequential([

        Input(shape=(input_shape,), name='input'),

        Dense(512, activation=activation, kernel_regularizer=kernel_regularizer, name='dense_2'),

        BatchNormalization(name='bn_2'),

        Dense(256, activation=activation, kernel_regularizer=kernel_regularizer, name='dense_3'),
```

```
        BatchNormalization(name='bn_3'),
        Dense(output_shape, activation=activation, kernel_regularizer=kernel_regularizer, name='dense_5'),
        BatchNormalization(name='bn_5'),
        CosineSimilarityLayer(output_dim=output_shape, name='output')
    ], name='zsl_model')
    return model
K.clear_session()
gc.collect()
trained_zsl_path = data_folder_path / add_info_to_strings('ZSL_Model.keras', target_size)
history_zsl_path = data_folder_path / add_info_to_strings('ZSL_Model_History.pkl', target_size)
if all(os.path.exists(path) for path in (trained_zsl_path, history_zsl_path)):
    zsl_model = load_model(trained_zsl_path, custom_objects={'f1_score': f1_score, 'CosineSimilarityLayer':
CosineSimilarityLayer})
    print(f"Trained ZSL model loaded successfully from {trained_zsl_path}.")
    history_zsl_dict = load_data(history_zsl_path)
    print(f"Training history dictionary loaded successfully from {history_zsl_path}.")
else:
    print("Trained model or history not found. Training a new model...")
    batch_size = 12
    learning_rate = 0.0001
    epochs = 100
    activation = 'relu'
    loss = 'mse'
    kernel_regularizer = l2(0)
    metrics = ['accuracy', AUC(name='auc'), Precision(name='precision'), Recall(name='recall'), f1_score]
    optimizer = Adamax(learning_rate=learning_rate)
    input_shape = seen_features.shape[1]
    num_classes = seen_predicates_vectors.shape[1]
    zsl_model = create_zsl_model(input_shape, num_classes, activation, kernel_regularizer)
    zsl_model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
    early_stopping_loss = EarlyStopping(monitor='loss', patience=5, restore_best_weights=True, min_delta=0,
mode='min', baseline=None, verbose=0)
    early_stopping_val_loss = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=False, min_delta=0,
mode='min', baseline=None, verbose=0)
    history_zsl = zsl_model.fit(seen_generator_ZSL, epochs=epochs, batch_size=batch_size,
validation_data=unseen_generator_ZSL, callbacks=[early_stopping_loss, early_stopping_val_loss], verbose=1)
    if early_stopping_loss.stopped_epoch > 0:
        print(f"\nTraining stopped at epoch {early_stopping_loss.stopped_epoch + 1} due to training loss criteria.")
    elif early_stopping_val_loss.stopped_epoch > 0:
```

```
        print(f"\nTraining stopped at epoch {early_stopping_val_loss.stopped_epoch + 1} due to validation loss criteria.")
    else:
        print("\nTraining stopped due to completing all epochs.")
    zsl_model.save(trained_zsl_path)
    print(f"Trained ZSL model saved successfully at {trained_zsl_path}.")
    history_zsl_dict = history_zsl.history
    save_data(history_zsl_dict, history_zsl_path)
    print(f"Training history dictionary saved successfully at {history_zsl_path}.")
plot_metrics(history_zsl_dict)
unseen_class_accuracies = unseen_generator_ZSL.evaluate_model(zsl_model, top_ns=[1, 2, 3])
def plot_unseen_class_accuracies(class_accuracies, top_n=1):
    top_n_key = f'top_{top_n}'
    if top_n_key not in class_accuracies:
        raise ValueError(f"No accuracies found for top-{top_n}")
    accuracies = class_accuracies[top_n_key]
    print(accuracies)
    sorted_indices = np.argsort(list(accuracies.values()))
    sorted_class_accuracies = [list(accuracies.values())[i] for i in sorted_indices]
    sorted_x_labels = [list(accuracies.keys())[i] for i in sorted_indices]
    fig, ax = plt.subplots(figsize=(6.5, 4.5))
    sorted_indices = list(sorted_indices)
    sorted_class_accuracies = list(sorted_class_accuracies)
    fig.patch.set_facecolor('#ffffff')
    ax.set_facecolor('#ffffff')
    classes = np.arange(len(sorted_class_accuracies))
    bars = ax.barh(classes, sorted_class_accuracies, color='#f55c5c', height=0.667, zorder=3)
    mean_accuracy = np.mean(list(accuracies.values()))
    ax.axvline(mean_accuracy, color='#f44a4a', linestyle='--', linewidth=1)
    for bar in bars:
        width = bar.get_width()
        if width == 100:
            ax.annotate(f'{width:.1f}%',
                        xy=(width, bar.get_y() + bar.get_height() / 2),
                        xytext=(-39, -1),
                        textcoords="offset points",
                        ha='left', va='center', fontsize=12, fontweight='bold', color='black')
        elif width >= 12:
```

```
            ax.annotate(f'{width:.1f}%',
                        xy=(width, bar.get_y() + bar.get_height() / 2),
                        xytext=(-34, -1),
                        textcoords="offset points",
                        ha='left', va='center', fontsize=12, fontweight='bold', color='black')
        else:
            ax.annotate(f'{width:.1f}%',
                        xy=(0, bar.get_y() + bar.get_height() / 2),
                        xytext=(2, -1),
                        textcoords="offset points",
                        ha='left', va='center', fontsize=12, fontweight='bold', color='black')
    ax.set_ylabel('Назва класу', fontsize=14)
    ax.set_xlabel('Точність', fontsize=14)
    ax.set_title(f'Точність по класам ZSL (топ-{top_n})', fontsize=14)
    ax.set_ylim([-0.5, len(classes) - 0.5])
    ax.set_xlim([0, 101])
    ax.set_yticks(classes)
    ax.set_yticklabels(sorted_x_labels, fontsize=12)
    ax.grid(axis='x', linestyle='--', alpha=0.7, zorder=0)
    formatter = FuncFormatter(lambda x, pos: f'{int(x)}%')
    ax.xaxis.set_major_formatter(formatter)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    plt.tight_layout()
    plt.show()
print(f"Total: {sum(unseen_class_accuracies['top_1'].values()) / 10:.2f} %")
print(f"Total: {sum(unseen_class_accuracies['top_2'].values()) / 10:.2f} %")
print(f"Total: {sum(unseen_class_accuracies['top_3'].values()) / 10:.2f} %")
for i in range(1, 4):
    plot_unseen_class_accuracies(unseen_class_accuracies, top_n=i)
plot_seen_class_accuracies(seen_class_accuracies, seen_dict)
show_random_images_from_classes_by_accuracy(unseen_dict, list(unseen_class_accuracies['top_1'].values()),
accuracy_range=(0, 100), word='найкраще')
```