

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«___» _____ 2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

**на тему: «Багатоміткова класифікація стилізованих зображень за
допомогою глибоких нейронних мереж»**

Виконав:

студент IV курсу, групи КА-13

Тюкалов Ніколай Сергійович

Керівник:

Асистент Древаль М.М.

Консультант з економічного розділу:

доцент, к.е.н. Рощина Н.В.

Консультант з нормоконтролю:

Асистент Канцедал Г.О.

Рецензент:

доцент к. СП, к.т.н. Безносик Олександр Юрійович

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

Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Оксана ТИМОЩУК

«__» _____ 2025 р.

ЗАВДАННЯ

на дипломну роботу студенту

Тюкалову Ніколаю Сергійовичу

1. Тема роботи «Багатоміткова класифікація стилізованих зображень за допомогою глибоких нейронних мереж», керівник роботи асистент Древаль М.М., затверджені наказом по університету від «26» травня 2025 р. №1759-с.
2. Термін подання студентом роботи 06.06.2025.
3. Вихідні дані до роботи: модель багатоміткової класифікації стилізованих зображень.
4. Зміст роботи: дослідження предметної області, огляд математичних основ до задачі багатоміткового класифікування з використанням глибоких нейронних мереж, моделювання, практичні результати та їх аналіз, висновки, список використаної літератури, додаток А, додаток Б.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): історичні зображення, схеми, графіки, ілюстрація використані для демонстрації роботи моделі класифікування, презентація.
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., доцент		

7. Дата видачі завдання 19.04.2025.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми ДР	13.04.2025-19.04.2025	Виконано
2	Пошук та аналіз наукової літератури	20.04.2025-27.04.2025	Виконано
3	Проведення обробки даних	28.04.2025-04.05.2025	Виконано
4	Проведення моделювання	05.05.2025-18.05.2025	Виконано
5	Робота над основною частиною ДР	19.05.2025-01.06.2025	Виконано
6	Проходження нормоконтролю	02.06.2025-06.06.2025	
7	Підготовка презентації		
8	Попередній захист дипломної роботи		
9	Захист дипломної роботи		

Студент

Ніколай ТЮКАЛОВ

Керівник

Максим ДРЕВАЛЬ

РЕФЕРАТ

Дипломна робота: 98 с., 12 рис., 8 табл., 2 додатків, 47 джерел.

КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ, СТИЛІЗОВАНІ ЗОБРАЖЕННЯ, ANIME ЗОБРАЖЕННЯ, БАГАТОМІТКОВА КЛАСИФІКАЦІЯ, КОМП'ЮТЕРНИЙ ЗІР, ГЛИБОКЕ НАВЧАННЯ, НЕЙРОННІ МЕРЕЖІ, DANBOORU, RESNET, MECHANISMS ATTENTION, SQUEEZE-AND-EXCITATION.

Об'єкт дослідження – великий незбалансований датасет стилізованих аніме-зображень, створений на основі даних з відкритого ресурсу Danbooru, що налічує понад 7 мільйонів зображень та характеризується детальною комплексною системою тегів.

Предмет дослідження – архітектури глибоких нейронних мереж, зокрема Attention-механізми, методи багатоміткової класифікації, а також стратегії роботи з даними, що мають значний дисбаланс класів та іншу специфічну проблематику.

Постановка задачі – спроектувати та натренувати модель багатоміткової класифікації стилізованих аніме-зображень з використанням сучасних архітектур нейронних мереж, яка здатна коректно класифікувати найпопулярніші теги.

Результатом роботи є функціональна модель багатоміткової класифікації оптимізована для роботи з аніме-зображеннями. Ця модель може бути інтегрована в існуючі або нові системи для автоматичного класифікування, пошуку та організації візуального контенту.

ABSTRACT

Thesis: **XX** pages, **XX** figures, **XX** tables., 2 appendancies, 47 references.

IMAGE CLASSIFICATION, STYLIZED IMAGES, ANIME IMAGES, MULTI-LABEL CLASSIFICATION, COMPUTER VISION, DEEP LEARNING, NEURAL NETWORKS, DANBOORU, RESNET, ATTENTION MECHANISMS, SQUEEZE-AND-EXCITATION.

The object of the study is a large, imbalanced dataset of stylized anime images, created based on data from the open-source resource Danbooru, comprising over 7 million images and characterized by a detailed, comprehensive tagging system.

The subject of the study encompasses deep neural network architectures, particularly Attention mechanisms, multi-label classification methods, and strategies for handling data with significant class imbalance and other specific challenges.

The task is to design and train a highly effective multi-label classification model for stylized anime images, capable of accurately classifying the most common tags, using modern deep neural network architectures.

The result of the work is a functional, trained multi-label classification model optimized for anime images. This model can be integrated into existing or new systems for automatic classification, searching, and organization of visual content.

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1 Актуальність задачі.....	10
1.1.1 Computer vision.....	10
1.1.2 Multilabel classification.....	14
1.2 Аналіз існуючих підходів та методів.....	15
1.2.1 Класичні підходи.....	15
1.2.2 Підходи на основі машинного навчання.....	16
1.3 Опис та аналіз набору даних.....	17
1.3.1 Про архив зображень Danbooru.....	17
1.3.2 Метадані.....	18
1.3.3 Проблематика набору даних.....	20
1.3.4 Етика та легальність.....	21
1.4 Формалізація задачі.....	22
1.5 Висновки до розділу 1.....	23
РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ МЕТОДІВ КЛАСИФІКАЦІЇ.....	24
2.1 Базові компоненти нейронних мереж.....	24
2.1.1 Принцип роботи нейронних мереж.....	24
2.1.2 Функції активації.....	27
2.1.3 Функції втрат.....	30
2.1.4 Оптимізатори.....	32
2.1.5 Регуляризація.....	34
2.2 Згорткові шари.....	36
2.3. Squeeze and Excitation.....	39

	7
2.4. Архітектура ResNet.....	40
2.5. Метрики оцінки продуктивності моделей.....	41
2.6. Висновки до розділу 2.....	43
РОЗДІЛ 3 РОЗРОБКА ТА ОЦІНКА МОДЕЛІ.....	44
3.1 Вибір мови програмування та фреймворків.....	44
3.2 Обробка датасету.....	45
3.3 Побудова архітектури моделі.....	48
3.4 Тренування та тестування та оцінка моделі.....	49
3.5 Висновки до розділу 3.....	55
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ.....	56
4.1 Постановка задачі проектування.....	56
4.2 Обґрунтування функцій програмного продукту.....	58
4.3 Обґрунтування системи параметрів програмного продукту.....	63
4.4 Аналіз експертного оцінювання параметрів.....	65
4.5 Аналіз рівня якості варіантів реалізації функцій.....	69
4.6 Економічний аналіз варіантів розробки ПП.....	70
4.7 Вибір кращого варіанту ПП техніко-економічного рівня.....	74
4.8 Висновки до розділу 4.....	75
ВИСНОВКИ.....	76
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	77
ДОДАТОК А ПРОГРАМНИЙ КОД.....	84
ДОДАТОК Б ГРАФІЧНІ МАТЕРІАЛИ.....	88

ВСТУП

Комп'ютерний зір (СМ) є одним з найбільш активно та динамічно розвиваючихся напрямків штучного інтелекту, який знайшов застосування в різноманітності індустрій та сфер. Засоби руху з автоматичним керуванням, контроль якості продукції, доповнена реальність, діагностика захворювань, пошук за зображенням – це все вимагає застосування комп'ютерного зору.

Незалежно від сфери застосування, найпоширенішою задачею комп'ютерного зору є класифікація зображення. Багатоміткова класифікація зображень є необхідною, коли нам необхідно ідентифікувати декілька об'єктів на зображенні одночасно, що є дуже реальною проблемою. Наприклад у системах автоматично керування засобів руху необхідно одночасно бачити всі дорожні знаки, світлофори, інші автівку і так далі.

В цій роботі ставиться на мету створити систему багатоміткової класифікації з великою кількістю класів для стилізованих зображень. В якості датасету обрано набір даних на основі зображень з Danbooru, найбільшого інтернет архиву аніме зображень. Кожне зображення має детальний опис з допомогою тегів, середня кількість тегів на зображення дорівнює 30. На мету буде поставлена задача створити модель з допомогою глибоких нейронних мереж, що здатна класифікувати топ-100 найпопулярніших тегів на сайті, використовуючи 25к зображень для навчання. Датасет пропонує як класичні виклики, такі як збалансованість тегів, так й унікальні: аніме стилістика, псевдо-ієрархічна структура тегів, абстрактні та комплексні теги.

Подібна модель багатоміткової класифікації може знайти наступні застосування: фільтрація та модерація зображень, допомога розмітки даних при створенні датасету, система рекомендації тегів на сайті та інші. Таким чином, розроблена система багатоміткової класифікації може стати цінним інструментом для організації контенту на Imageboard площадках.

В першому розділі роботи буде розглянуто історію та ринок Computer Vision, особливості та методи багатоміткової класифікації, Danbooru та датасет на основі ресурсу. Буде формалізовано задачу. У другому розділі буде детально розглянуто та обрано методи розв'язання задачі багатоміткової класифікації зображень. Третій розділ буде присвячений побудові, тренуванню, тестуванню та оцінці моделі. Четвертий розділ відведений під функціонально-вартісний аналіз розробки.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність задачі

1.1.1 Computer vision

Комп'ютерний зір (Computer Vision, CV) є однією з провідних галузей штучного інтелекту (Artificial Intelligence, AI, ШІ), яка використовує машинне навчання та нейронні мережі для надання комп'ютерам та системам здібності отримувати змістовну інформацію з електронних зображень, відео, та інших медіа джерел, та робити якісь дії на основі цієї інформації [1, 2].

Комп'ютерний зір в своїй суті імітує людський зір: люди використовують мозок, очі, та весь життєвий досвід для аналізу та ідентифікації того, що вони бачать, в той час як комп'ютер має нейронну мережу (або інший алгоритм) замість мозку, камери замість очей, та набори даних замість життєвого досвіду [1].

Передумови для появи комп'ютерного зору з'явилися в кінці 1950-х років, коли вчені Торстен Візел (Torsten Wiesel) та Девід Г'юбел (David Hunter Hubel) намагалися зрозуміти як мозок сприймає та опрацьовує візуальну інформацію. В ході експерименту у 1959 році нейрофізіологи показували зображення кішці та намагалися знайти відповідну кореляцію в мозку. Дослідження показало, що в першу чергу мозок аналізує контури та лінії [1, 3]. Інша важлива подія відбулася у 1957 році американський інженер Рассел Кірш (Russell Kirsch) розробив першу технологію, яка зробила можливим створювати цифрове зображення та передавати його у комп'ютерну систему – електронний барабанний сканер. [1, 2, 4]. Першим цифровим зображенням стало фото Валдена Кірша (Walden Kirsch) (рисунок 1.1), сина дослідника.



Рисунок 1.1 – Перше цифрове зображення в історії [4].

У 1960-х роках після того як штучний інтелект став окремою галуззю академічних досліджень почався розвиток безпосередньо комп'ютерного зору [2]. Технологія оптичного розпізнавання символів (Optical character recognition, OCR) була розроблена у 1970-х роках, вона вперше дозволила комп'ютерам розпізнавати друкований текст. Подалі було розроблено інтелектуальне розпізнавання символів (Intelligent Character Recognition, OCR), яке використовувало нейронні мережі та було здатне читати навіть письмовий текст [1, 2].

У 1980-х роках відбулася одна ключових подій в розвитку комп'ютерно зору, коли британський нейрофізіолог Девід Марр (David Marr) висунув ідею про ієрархічну природу зорового сприйняття та запропонував алгоритмічну модель, яка включала виявлення базових ознак зображення – контурів, кутів і кривих [5]. В цей самий період японський дослідник Кунихико Фукусіма (Kunihiko Fukushima, 福島 邦彦) створив Neocognitron – модель штучної нейронної мережі. Ця архітектура вперше використала ідеї згорткових шарів

для розпізнавання шаблонів, ставши теоретичною основою сучасних згорткових нейронних мереж (CNN) [6].

У 2000-х роках увага дослідників сфокусувалася на задачах розпізнавання об'єктів та облич [1, 2, 7]. Визначальною подією стала поява у 2010 році набору даних ImageNet, що містив мільйони розмічених зображень для тисяч класів об'єктів [1, 2, 8]. У 2012 році модель AlexNet, побудована на згортковій нейронній мережі (Convolutional Neural Network, CNN), здобула перемогу в конкурсі ImageNet Large Scale Visual Recognition Challenge (ILSVRC), значно знизивши помилку класифікації порівняно з попередніми підходами [1, 2, 9]. Цей результат започаткував нову еру глибокого навчання в обробці зображень, яка триває досі.

На поточний момент комп'ютерний зір закріпився в різноманітній кількості сфер та індустрій. Розглянемо приклади застосування комп'ютерного зору.

1. Автономне водіння. CV є критичним компонентом будь якої сучасної системи автокерування Автомобільні компанії такі як Tesla, Waymo, Cruise використовують глибокі згорткові нейронні мережі (CNN) у комбінації з lidar/радар-даними [10].
2. Охорона здоров'я. Комп'ютерний зір застосовується для обробки зображень МРТ, КТ, рентгену, ендоскопії. Моделі використовуються для виявлення захворювань, зменшують час та збільшують точність діагностування [11].
3. Промисловий контроль якості. Системи, що використовують CV, дозволяють автоматично виявляти дефекти в товарах або порушення технологічних норм [12].
4. Біометрична ідентифікація. Комп'ютерний зір широко використовується для біометричної ідентифікації, зокрема державними структурами. Наприклад в Україні один з етапів аторизії в державному застосунку ДІА є біометрична ідентифікація через камеру [13]. Іншим прикладом є влада Китаю, яка впроваджує систему масового

спостереження з використанням CV. Система ідентифікує людей, їх поведінку, та використовує ці дані в програмі соціального кредиту. [14].

5. Робототехніка. Компанії, такі як Boston Dynamics, застосовують технологію комп'ютерного зору для навігації роботів, розпізнавання об'єктів і взаємодії з навколишнім середовищем. [15].

6. Доповнена реальність. В AR-окулярах та VR-системах CV використовується для розпізнавання рук, положення тіла, і оточення, щоб точно накладати віртуальні об'єкти на реальні [16].

Світовий ринок комп'ютерного зору оцінюється по-різному залежно від джерела, оскільки методології збору та обробки аналітичних даних відрізняються. Проте на 2025 рік обсяг ринку, згідно з аналітикою Grand View Research (GVR), оцінюється у 23.62 мільярдів доларів США. Згідно з прогнозами, очікується, що ринок зростатиме із середньорічним темпом зростання (CAGR) 19.8% у період 2025–2030 років, досягаючи 58.29 мільярдів доларів США у 2030 році [17].

Інше джерело, Statista, наводить вищу початкову оцінку ринку на 2025 рік у 29.88 мільярдів доларів США. Згідно з їхніми даними, середньорічний темп зростання ринку складе 15.96% у період з 2025 до 2031 року, що дозволить досягти 72.66 мільярдів доларів США у 2031 році. Якщо екстраполювати дані Statista на 2030 рік, орієнтовна оцінка ринку складатиме близько 62.67 мільярдів доларів США [18].

Обидва джерела (GVR і Statista) виділяють головними регіонами зростання ринку комп'ютерного зору Північну Америку (особливо США) та Південно-Східну Азію. Grand View Research оцінює, що на останній регіон припадає 41% глобального ринку, з особливим акцентом на Китай, Японію та Південну Корею як ключових гравців [17].

1.1.2 Multilabel classification

Багатоміткова класифікація (Multi-label Classification, Multi-output classification, MLC) є окремим випадком задачі класифікації машинного навчання, в якому кожен приклад може належати до кількох класів одночасно. Це принципово відрізняється від традиційної задачі класифікації, де кожен об'єкт належить виключно одному класу [19].

У реальних задачах багатоміткова класифікація часто є більш природної ніж класична класифікація. Як приклад, якщо спробувати класифікувати статтю про обрання нового Папи Римського Лео XIV, скоріше за все її можна буде одночасно віднести до декількох категорій: “релігія”, “політика”, “світові новини”, і т.д. Аналогічна ситуація відбувається і в інших медиумах інформації: музичний трек скоріше за все поєднує декілька жанрів (наприклад, “електронна музика” та “танцювальна музика”), а фото має декілька об'єктів одночасно (наприклад, “дерево” та “квіти”).

Специфіка задачі багатоміткової класифікації породжує низку унікальних викликів, яких не існує в класичній задачі класифікації. Класи мають схильність перетинатися та, відповідно, корелювати (наприклад, скоріше за все клас “море” буде корелювати з класом “пляж”). Також типовою проблемою є сильний дисбаланс між класами, який неможливо розв'язати класичними методами. Наприклад, усі музичні треки класу “альтернативний рок” мають одночасно належати до ширшої категорії “рок”, але не всі треки з жанру “рок” відносяться до специфічного піджанру “альтернативний рок”. В класичній задачі класифікації цю проблему можна легко вирішити простим методом як Oversample (генерація або додаткове набрання дефіцитного класу), але багатоміткова класифікація вимагає нових підходів.

Багатоміткова класифікація також вимагає нових підходів в інших напрямках таких як метрики. Приклад: традиційно однією з ключових метрик для оцінки якості моделі при задачі класифікації є F1-score, який визначає

баланс між точністю (precision) та повнотою (recall). Традиційний підхід просто не працює адже виникає питання як рахувати precision та recall коли в тебе багато класів одночасно, отже в контексті нової задачі ця метрика розраховується у двох варіантах: F1-micro та F1-macro [20].

F1-micro обчислюється аналогічно звичайному F1-score, але TP (істинно позитивні), FP (хибно позитивні), та FN (хибно негативні) прогнози обчислюються глобально за всіма класами одночасно. F1-micro використовується для оцінки ефективності моделі на рівні індивідуальних міток, має схильність до переоцінки якості моделі через домінуючі класи.

F1-macro в свою чергу обчислює F1-score окремо для кожного класу, а потім бере середнє значення по всіх класах. Це забезпечує однакову вагу кожному класу незалежно від його частоти, що особливо важливо при сильному дисбалансі. Macro F1 дозволяє краще оцінити, наскільки добре модель працює з рідкісними класами, але вона також має схильність до недооцінки загальної якості моделі, якщо існує багато рідкісних класів.

1.2 Аналіз існуючих підходів та методів

1.2.1 Класичні підходи

Перед появою глибокого навчання багатоміткову класифікацію вирішували за допомогою класичних алгоритмів машинного навчання. Існує багато класичних методів, але вони втрачають свою актуальність зв'язку з розвитком та популяризацією нейронних мереж отже буде наведено лише декілька прикладів.

Binary Relevance (BR). Метод BR передбачає тренування окремої моделі для кожної мітки. Кожна модель фактично є класичним бінарним

класифікатором для певної мітки. Цей підхід простий у реалізації, але ігнорує кореляції між мітками, що може призводити до менш точної класифікації у випадках, коли мітки взаємопов'язані [21].

Classifier Chains (CC). Метод CC це розширення BR, яка намагається врахувати залежності між мітками. Головною ідеєю є створення ланцюгу класифікаторів, де кожен наступний класифікатор враховує прогнози класифікаторів перед ним як додаткові вхідні параметри. Даний підхід дійсно дозволяє враховувати зв'язки між класами, але має вразливість до порядку в яку відбуваються класифікації[21].

Label Powerset (LP). Метод LP трансформує задачу багатоміткової класифікації у задачу багатокласової класифікації, де кожна унікальна комбінація міток розглядається як окремий клас. Через те що цей метод створює окремий клас для кожної можливої комбінації він враховує всі можливі зв'язки, але це також створює надмірну кількість класів отже метод погано масштабується та ускладнює навчання [21].

1.2.2 Підходи на основі машинного навчання

Сучасні підходи до задачі багатоміткової класифікації ґрунтуються на використанні нейронних мереж. Типова архітектура сучасної багатоміткової нейронної мережі для класифікації зображень включає послідовність згорткових шарів (CNN), після яких йдуть повнозв'язні шари. Вихідним шаром зазвичай є бінарний вектор, розмірність якого дорівнює кількості класів. Як функцію активації найчастіше використовують сигмоїду. У якості функції втрат зазвичай застосовується бінарна крос-ентропія (BCE). Також поширене використання попередньо натренованих моделей (Transfer Learning), які адаптуються до конкретного завдання.

CNN (Convolutional Neural Networks) наразі є головним інструментом для вирішення більшості задач комп'ютерного зору на основі нейронних мереж. Кожен згортковий шар вивчає просторові патерни, починаючи з базових (лінії, кути), і переходить до більш абстрактних (обличчя, об'єкти, текстури).

У 2017 році був опублікований інноваційний attention-механізм у статті "Attention is All You Need" [22], який став основою для розвитку нових архітектур у глибокому навчанні. Спочатку attention застосовувався у сфері обробки природної мови (NLP), де продемонстрував виняткову здатність знаходити залежності між елементами вхідних даних.

Завдяки універсальності attention-механізму його почали адаптувати й до інших типів даних, зокрема зображень, що дало поштовх до розвитку attention-архітектур – зокрема візуальних трансформерів (Vision Transformers, ViT) [23].

1.3 Опис та аналіз набору даних

1.3.1 Про архив зображень Danbooru

Danbooru - це онлайн-платформа, яка функціонує як imageboard, спеціалізований на ілюстраціях в стилістиці аніме. Назва "Danbooru" походить від японського слова "danbooru" (段ボール), що означає "картонна коробка". Частина "booru" є грою слів від "board" (як в imageboard).

Основний функціонал сайту полягає у завантаженні зображень та їх анотуванні за допомогою тегів. За своєю суттю платформа є, ймовірно, найбільшим онлайн-архівом стилізованих зображень, зокрема в аніме-естетиці.

Danbooru був створений орієнтовно у 2005–2006 роках, однак через природу ресурсу та відсутність офіційної документації, точну дату запуску, та будь яку іншу інформацію, встановити складно. Платформа швидко набула популярності завдяки ліберальному підходу до модерації контенту. В результаті популярності Danbooru з'явилися схожі ресурси з різними специфіками, наприклад Safebooru, який спеціалізується на “безпечних” зображеннях в аніме стилістиці.

Об'єктивно оцінити розмір Danbooru складно, однак непрямі оцінки можна зробити на основі публічно доступних наборів даних, створених на базі цієї платформи, Danbooru2021 [24] і Danbooru2023 [25], які містять приблизно 4.9 млн та 6.8 млн зображень відповідно. Станом на 2025 рік можна обережно оцінити, що загальний обсяг архіву перевищує 7 мільйонів зображень.

Кількість існуючих тегів оцінити ще складніше, адже користувачі можуть вільно створювати нові в будь який момент. Система дозволила існуванню детальної системи опису зображень з допомогою тегів, ате також дозволяє створити новий тег випадково через орфографічну помилку. Безпечною оцінкою буде щонайменше мільйон унікальних (не помилкових тегів), що включає загальні описову теги, теги персонажів, теги художників, копірайт теги, та мета теги. Середня кількість тегів на одне зображення дорівнює приблизно 30 [25].

1.3.2 Метадані

Метадані Danbooru мають вичерпну інформацію про зображення, що робить їх дуже корисними в використанні для тренування моделей класифікації, генеративних моделей і т.д. Структура метаданих досить комплексна.

1. Теги. Кожне зображення детально описано та розмічено з використанням тегів, які розділені на 5 головних категорій:

- художник;
- авторські права (те на чому основане зображення, наприклад специфічний серіал чи гра);
- зображений персонаж;
- загальні описові теги (наприклад “1girl” та “solo”);
- мета тегів (наприклад “highres”, якщо зображення високої якості).

2. Загальна інформація:

- ID кожного зображення;
- ім'я користувача, який завантажив зображення;
- дата завантаження;
- те скільки об'єму пам'яті зображення займає, тип файлу (наприклад .jpeg чи .png), розмір зображення (кількість пікселів в ширину та висоту);
- URL посилання на джерело роботи;
- рейтинг контенту (safe, questionable, explicit);
- оцінка користувачів.

3. Інше:

- MD5 хеш для перевірки цілісності файлу;
- пов'язані зображення (наприклад, ескіз чи чорно-біла версія);
- підписи та діалоги;
- коментарі автора.

1.3.3 Проблематика набору даних

В цій роботі використовується фрагмент безпечного фільтрованого датасету Tagged Anime Illustrations [27] отже можна зосередитися виключно на технічній та математичній складності.

Основна проблематика датасету заключається в системі тегування. Через хаотичну природу сайту та відсутність централізації теги характеризуються наступними проблемами.

1. Перетини. Багато тегів мають значне семантичне перекриття, що призводить до високої кореляції між ними. Наприклад, теги "grayscale" та "monochrome" часто використовуються для опису одних і тих самих зображень.
2. Абстрактність. Деякі теги описують досить абстрактні або специфічні концепції, які можуть бути складними, або майже неможливими, для класифікації моделлю класифікації. Наприклад тег "ahoge", який означає специфічний елемент волосся в деяких підтипах аніме стилізації. Іншим прикладом є тег "alternate costume", який використовується коли персонаж з усталеним дизайном зображений в альтернативному одязі. Для того щоб класифікувати подібний тег модель має вивчити персонажа в класичному дизайні, та вміти класифікувати того самого персонажа в альтернативному одязі, що ускладнюється потенційними випадками коли фанатський дизайн є популярнішим за класичний.
3. Псевдоієрархічна структура. Хоча de jure ієрархії тегів не існує, спостерігається неявна ієрархія, що виникає досить природним чином з суті самих тегів. Наприклад зображення з тегом "sword" чи "spear" завжди мають мати тег "weapon".

4. Критичний дисбаланс класів. Найпопулярніші теги, такі як “1girl”, який наявний приблизно в 6.5 мільйонах зображень, зустрічаються магнітудно частіше ніж більш специфічні теги, такі як “white skirt”, що зустрічається лише скромних 98 тисяч разів.
5. Відсутність тегів. Теги, які мають бути на зображенні, можуть бути відсутні, навіть якщо вони є поширеними. Це створює ситуації, коли модель може бути наказана навіть при *de facto* коректній класифікації під час навчання.

Інша проблематика виникає саме з аніме стилістики. Наприклад, рівень гендерного диморфізму, на відміну від реального світу, може бути мінімальним в деяких типах аніме стилістики. Ця специфіка помітно ускладнює коректну класифікацію чоловіків та жінок, особливо при відсутності “підказок” (таких як наявність бороди у персонажа).

1.3.4 Етика та легальність

Автор роботи не є кваліфікованим юристом, отже цей підрозділ мають ціллю лише визначити існування певних питань. Існує дві перспективи які варто зазначити: легальність Danbooru як ресурсу, легальність та етичність створення наборів даних на основі цього ресурсу.

Danbooru містить велику кількість зображень, і важко перевірити законність кожного завантаження. Платформа покладається на користувачів у дотриманні авторських прав, але потенційно може містити контент, що порушує ці права. Отже в контексті цієї роботи ми залишимо imageboard в “зоні сірою легальності” без будь яких висновків.

Більш цікавим питанням є етичність та легальність створення наборів даних на основі Danbooru та, якщо розширити питання, умовно відкритих даних як таких. Популяризація дифузійних та генеративних нейронних мереж

посприяла актуалізації та гостроті цього питання. З одного боку лише завдяки великим обсягам умовно відкритої цифрової інформації з'явилися сучасні LLM (Large Language Models) та GAN (Generative Adversarial Network) моделі, з іншого боку можливо вони в цілому не мали права на існування. В контексті цієї роботи автор також утримується від відповіді і на це питання.

1.4 Формалізація задачі

Нехай маємо деяку множину зображень:

$$D = \{(x_i, y_i)\}_{i=1}^N,$$

де $x_i \in R^{H*W*C}$ – вхідне зображення з висотою H та шириною W пікселів, з кількістю каналів C ;

$y_i \in \{0, 1\}^K$ – бінарний вектор розміру K , що відповідає зображенню x_i .

Компоненти $y_{ij} = 1$, відповідають наявності класу j на зображенні x_i , якщо клас відсутній $y_{ij} = 0$. Кожне зображення може мати $[0; K]$ міток одночасно.

Задача багатоміткової класифікації полягає у навчанні функції $f_{\theta}(x): R^{H*W*C} \rightarrow [0, 1]^K$, яка для кожного зображення x_i прогнозує вектор $\hat{y}_i \in [0, 1]^K$, де $\hat{y}_{ij}$ є ймовірністю належності i -го зображення до j -го класу.

Процес навчання заключається в знаходженні оптимальних параметрів $\bar{\theta}$ шляхом мінімізації деякої функції втрат $L(y, \hat{y})$, яка є мірою розбіжності між передбаченими значеннями $\hat{y}$, та істини значенням y :

$$\bar{\theta} = \frac{1}{N} \sum_{i=1}^N L(y_i, f_{\theta}(x_i)) \rightarrow \min.$$

1.5 Висновки до розділу 1

У цьому розділі було розглянуто історію розвитку комп'ютерного зору починаючи від спроб вчених зрозуміти як мозок опрацьовує зображення, і закінчуючи поточним часом. Було згадано основні сфери та індустрії, де застосовується CV, та знайдено оцінку глобального ринку з прогнозом до 2031 року.

Було розглянуто поняття багатоміткової класифікації, відмінності від класичної класифікації, особливості, та складності. В першому розділі також наведені класичні підходи, такі як Binary Relevance, Classifier Chains та Label Powerset, та сучасні підходи зі застосуванням нейронних мереж, що базуються на CNN, та attention-механізмах.

Набір даних, та їх джерело ресурс Danbooru, також були описані в цьому підрозділі. Було наведено короткий опис сайту та його функціонал. Структура даних та метаданих детально описані в відповідному підрозділі. Питання легальності та етичності були згадані та описані, але залишені без відповіді. Складності набору даних було зазначено та описано.

Наступний розділ буде присвячений детальному опису та вибору методів розв'язання задачі. У подальших етапах дослідження буде створено модель багатоміткової класифікації, та проведена оцінка моделі.

РОЗДІЛ 2 ТЕОРЕТИЧНІ ОСНОВИ МЕТОДІВ КЛАСИФІКАЦІЇ

2.1 Базові компоненти нейронних мереж

2.1.1 Принцип роботи нейронних мереж

Нейронні мережі (Neural Networks) це парадигма машинного навчання, яка імітує роботу біологічних нейронних мереж головного мозку. Структура нейронних мереж складаються з з'єднання вузлів, які називаються штучними нейронами. Нейрони зазвичай об'єднуються в шари, які в свою чергу поділяються на три категорії: вхідний шар, прихований шар (або шари), та вихідний шар (рисунок 2.1) [26].

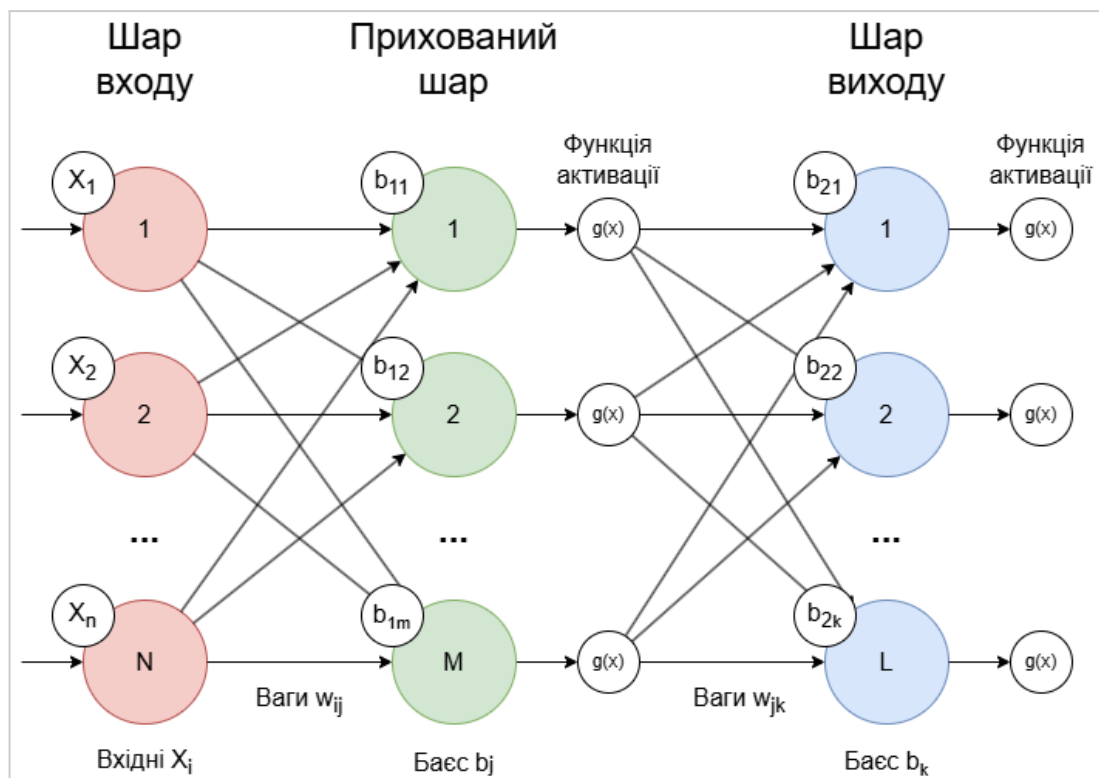


Рисунок 2.1 – Схема роботи нейронної мережі.

Як можна побачити на схемі вище (рис 2.1), кожен i -тий нейрон вхідного шару зі вхідним значенням x_i пов'язан з кожним j -тим нейроном схованого шару із вагою w_{ij} . Значення $z_{h,j}$ кожного j -того нейрону шару h , обчислюється за формулою (2.1) [26].

$$z_{h,j} = g\left(\sum_{i=1}^N (w_{ij}x_i) + b_{h,j}\right), \quad (2.1)$$

де $z_{h,j}$ – ітогове значення j -того неройну шару h ;

x_i – вхідне значення i -того неройну;

N – кількість нейронів вхідного шару;

w_{ij} – вага, яка пов'язує нейрони i та j ;

$b_{h,j}$ – байєс (bias) j -того неройну шару h ;

$g(\cdot)$ – функція активації.

Функція активації необхідна для створення нелінійності в нейронній мережі. Нелінійність в свою чергу необхідна адже інакше всю мережу можна було б звести до однієї лінійної трансформації, що в свою чергу означало б, що мережа лише здатна робити лише лінійні зв'язки між вхідними даними та вихідним шаром [26].

Цим чином обчислюються всі нейрони схованого шару, після чого обчислюється, або наступний схований шар, або вихідний шар (як на схемі вище (рис. 2.1)). Загалом цей процес називається прямим поширенням (Forward Pass) [26].

Альтернативно Forward Pass та активацію можна записати в загальному матричному вигляді (формула 2.2):

$$Z_i = g(W_i A_{i-1} + B_i) \quad (2.2)$$

де Z_i – вихідний вектор для i -того шару;

W_i – вектор ваг;

A_{i-1} – вектор вхідних значення попереднього шару (якщо попередній шар був вхідним, то A_{i-1} відповідає вектору вхідних даних X);

B_i – вектор байєсу,

$g(\cdot)$ – функція активації.

Після отримання вектора передбачення вихідного шару обчислюється функція втрат $L(y, \hat{y})$, де y – справжній вектор, $\hat{y}$ – вихідний вектор мережі. Обчислення функції втрат є першим кроком для навчання мережі. Процес оптимізації мережі ґрунтується на двох аспектах: зворотне поширення помилки (Backpropagation) та оптимізатор. Оптимізатор прийнято використовувати з гіпер параметром, який називається швидкістю навчання (Learning rate)[26].

Зворотне поширення помилки це алгоритм поетапного обчислення градієнтів функції втрат відносно параметрів мережі, починаючи з вихідного шару. На кожному кроці використовується правило ланцюга для обчислення, як зміна кожного параметра впливає на загальну помилку. Оптимізатор використовує знайдені градієнти для обчислення оптимальної зміни в параметрах. Математично початок Back propagation (вихідний шар) можна записати наступною формулою (формула 2.3):

$$\delta^{(L)} = \nabla_{a^{(L)}} L \odot g'(z^{(L)}), \quad (2.3)$$

де $\delta^{(L)}$ – помилка вихідного шару L ;

$g'(z^{(L)})$ – вектор поелементних похідних функції активації;

$\nabla_{a^{(L)}} L$ – градієнт функції втрат по відношенню до виходів шару L ;

$\odot$ – поелементний / Адамарів добуток.

Для всіх наступних шарів обчислення помилки відбувається наступним чином (формула 2.4):

$$\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) \odot g'(z^{(l)}), \quad (2.4)$$

де $(W^{(l+1)})^T$ – транспонована матриця ваг наступного шару;
 $\delta^{(l+1)}$ – помилка наступного шару;
 $g'(z^{(l)})$ – вектор похідних функцій активації.

Після обчислення помилки відповідного шару, можна обчислити градієнт ваг та байєсів цього шару за допомогою формул (2.5) та (2.6) відповідно [26].

$$\nabla_{W^{(l)}} L = \delta^{(l)} (a^{(l-1)})^T \quad (2.5)$$

$$\nabla_{b^{(l)}} L = \delta^{(l)} \quad (2.6)$$

2.1.2 Функції активації

Як було згадано в розділі вище (розділ 2.1.1), функція активації застосовується до зваженої суми входів нейронів і визначає його активацію. Мета використання функції активації заключається в введенні нелінійності в нейронну мережу, щоб запобігти можливості зведення мережі до однієї лінійної трансформації. Нелінійні задачі вимагають нелінійного підходу.

Існує також інша проблема пов'язана з використанням функцій активації, яку варто згадати – проблема згасаючих градієнтів (Vanishing gradient problem) [27]. Суть проблеми заключається в тому, що на етапі зворотного поширення помилки, при використанні деяких типів функцій активації (наприклад, sigmoid та tanh), значення градієнту зменшується з кожним шаром, і, відповідно, мережа навчається дуже повільно (або не навчається) на початкових шарах. Головна причина це використання функцій активації які надають на вихід значення $|x| < 1$, через те що багатократне перемноження їх похідних стрімко знижує порядок градієнту. Тепер розглянемо докладніше деякі з частих або важливих функцій активації.

ReLU (Rectified Linear Unit) одна з найпопулярніших функцій активації через простоту реалізації, низьку обчислювальну затратність, та ефективність. Також варто зазначити, що ReLU має деяку стійкість до проблеми загасаючого градієнту. ReLU була популяризована після її успішного застосування в AlexNet [9]. Формулювання ReLU та похідної наведені нижче, (формула 2.7) та (формула 2.8) відповідно. Похідна ReLU, для практичних цілей, не є похідною в строго математичному сенсі.

$$f(x) = \max(0, x), \quad (2.7)$$

$$f'(x) = 1, x > 0; 0, x \leq 0 \quad (2.8)$$

GELU (Gaussian Error Linear Unit) – це досить сучасна функція активації, яка дещо схожа з ReLU, але є в деякому сенсі її розвитком. Функція отримала популярність через успішне застосування в моделях обробки природної мови та трансформерах. На відміну від ReLU, GELU є гладкою функцією, що вважається сприятливим для більш плавного та стабільного навчання, особливо в архітектурах глибоких мереж. Очевидним недоліком є підвищення складності реалізації, та обчислювальна затратність (порівняно з ReLU). Формули GELU (формула 2.9) наведена нижче [28].

$$f(x) = x \cdot \Phi(x) \quad (2.9)$$

В формулі вище (формула 2.9) $\Phi(x)$ це кумулятивна функція розподілу стандартного нормального розподілу. Зазвичай на практиці GELU апроксимується через функцію сігмоїди (σ) (формула 2.10).

$$f(x) \approx x \cdot \sigma(1.702x) \quad (2.10)$$

Відповідно апроксимації самої функції, зазвичай використовують також апроксимацію її похідної (формула 2.11).

$$f'(x) \approx \sigma(1.702x) + x \cdot 1.702 \cdot \sigma(1.702x)(1 - \sigma(1.702x)) \quad (2.11)$$

Sigmoid (Сигмоїдальна функція) – історично одна з перших функцій активації, також є ключовою функцією активації для задачі багатоміткової класифікації (використовується для вихідного шару). Має слабкість до проблеми згасаючого градієнту, але зручна тим що переводить вихідні

нейрони в “псевдо ймовірності”. Формули функції (формула 2.12) та її похідної (формула 2.13) нижче.

$$f(x) = \frac{1}{1+e^{-x}} \quad (2.12)$$

$$f'(x) = f(x) \cdot (1 - f(x)) \quad (2.13)$$

Swish (або SiLU, Sigmoid Linear Unit) – це остання функція активації яку буде розглянути, і це також одна з останніх винайдених функцій, яку запропонувала компанія Google у 2017 році [29]. Ідейно вона схожа на GELU тим чином, що вона також намагається поєднати сильні сторони ReLU та Sigmoid функцій. Має схильність показувати кращі результати, порівняно з ReLU та іншими функціями, для деяких задач та архітектур. Формули функції (2.14) та її похідної (2.15) наведені нижче.

$$f(x) = x \cdot \sigma(x) \quad (2.14)$$

$$f'(x) = \sigma(x) + x \cdot \sigma(x)(1 - \sigma(x)) \quad (2.15)$$

Для додаткового порівняння функцій активації ма їх втрат нижче наведено графік (рисунок 2.2).

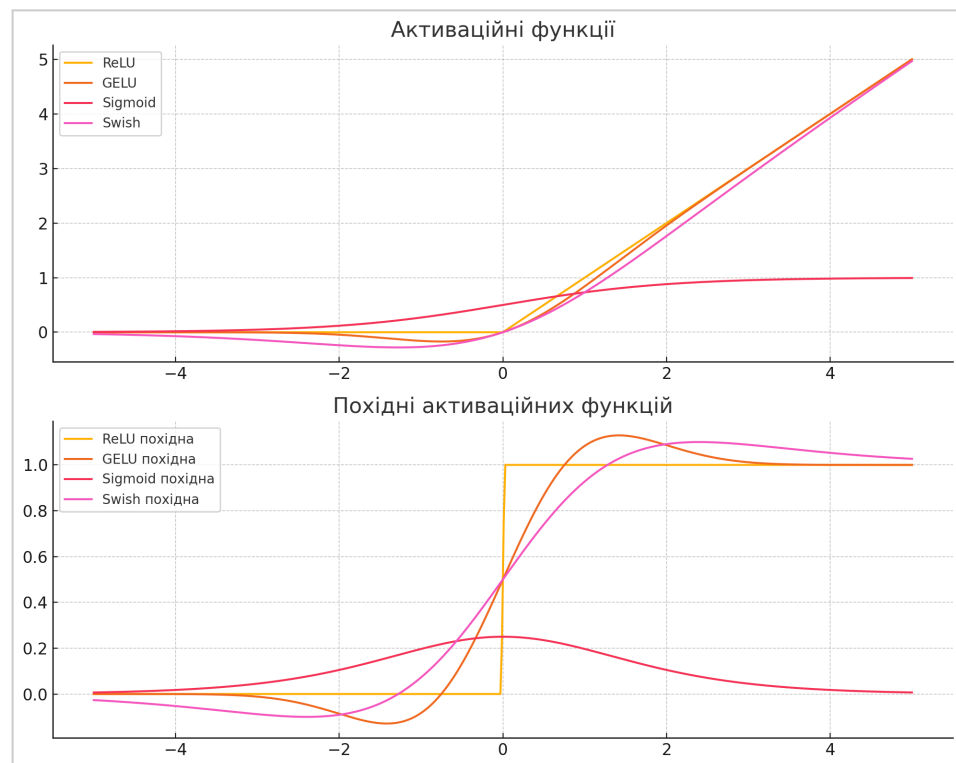


Рисунок 2.2 – Візуальна демонстрація згаданих функцій втрат та їх похідних

2.1.3 Функції втрат

Функція втрат (Loss Function, Cost Function, Error Function) – ключовий компонент процесу навчання нейронної мережі, який оцінює розбіжність між вихідним шаром (передбаченими значеннями) та істинними значеннями (Ground Truth). При коректному процесі навчання мережа намагається покроково мінімізувати функцію витрат, тим самим оптимізуючі саму модель. Саме через те, що саме функція втрат мінімізується, критично необхідно обирати коректну функцію втрат відповідно до задачі моделювання та особливостей набору даних. Функція витрат має коректно відображувати цілі навчання [26].

Існують, та активно розвиваються, велике різноманіття функцій втрат, кожна зі своєю конкретною специфікою та спеціалізацією. Для задачі багатоміткової класифікації необхідно знати про функцію втрат Binary Cross-Entropy (BCE), яка є відправною точкою для цієї задачі в цілому, та її модифікацією Focal loss.

Binary Cross-Entropy – це стандартна функція втрат для задач бінарної класифікації, а також задач багатоміткової класифікації. Ця функція вимірює розбіжність між розподілами ймовірності вектору передбачень (вихідний вектор) та вектору істинних значень (Ground Truth). BCE має базовий вигляд (формула 2.16), та розширений узагальнений випадок для багатоміткової класифікації (формула 2.17) [26].

$$L_{BCE}(y, \hat{y}) = -(y \cdot \log(\hat{y}) + (1 - y) \log(1 - \hat{y})), \quad (2.16)$$

де $y \in \{0, 1\}$ – істина бінарна мітка;

$\hat{y} \in [0, 1]$ – передбачена ймовірність мітки.

$$L_{BCE}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N (y_i \cdot \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)), \quad (2.17)$$

де y – вектор істинних міток;

y_i – мітка певного класу вектору істинних міток;

$\hat{y}$ – вектор ймовірностей передбачених міток;

$\hat{y}_i$ – відповідає класу y у векторі ймовірностей передбачених міток;

N – кількість класів.

Інтуїтивно формула можна зрозуміти наступним чином: якщо $y = 1$, то залишається лише доданок $-\log(\hat{y})$, відповідно чим ближче $\hat{y}$ до 1, тим менша втрата; якщо $y = 0$, то залишається лише $-\log(1 - \hat{y})$, відповідно чим ближче $\hat{y}$ до 0, тим менша втрата.

BCE має певні переваги, такі як простота реалізації та невисокі обчислювальні витрати, але ця функція також має суттєвий недолік – висока вразливість до дисбалансу класів в наборі даних. Для боротьби з проблемою дисбалансу даних була введена фокальна функція втрат (Focal Loss) [30].

Focal Loss – це модифікація BCE, яка фокусується на складних прикладах зменшенням ваги для прикладів, які класифікуються успішно, та збільшенням ваги для прикладів, які класифікуються менш успішно. Ця методика дозволяє мережі сфокусуватися або на важких прикладах, або на рідкісних прикладах. Формула функції наведена нижче (формула 2.18).

$$L_{Focal}(y, \hat{y}) = -\alpha_t (1 - \hat{p}_t)^{\gamma} \log(\hat{p}_t), \quad (2.18)$$

де $y \in \{0, 1\}$ – істина бінарна мітка;

$\hat{y} \in [0, 1]$ – передбачена ймовірність мітки;

$\hat{p}_t$ – ймовірність правильного класу (відповідно, якщо $y = 1$, то $\hat{p}_t = \hat{y}$

та $\hat{p}_t = 1 - \hat{y}$ інакше);

α_t – гіпер параметр балансування класів (вага класів),

γ – гіпер параметр фокусування на важких прикладах.

Інтуїтивно ця функція втрат працює наступним чином: якщо ймовірність правильного класу висока (тобто модель коректан та впевнена), то $(1 - \hat{p}_t)$ буде досить низьким, і ще додатково зменшиться завдяки γ . Додатково також використовуються ваги класів для подальшого підвищення важливості складних прикладів.

2.1.4 Оптимізатори

Оптимізатор є іншим важливим компонентом навчання нейронної мережі, який використовується для оптимальної корекції параметрів на етапі зворотного поширення помилки таким чином, щоб мінімізувати значення функції втрат. Від оптимізатора залежить швидкість та ефективність навчання, а також можливість моделі до узагальнення, тобто можливість запобігання локальних, та здатність знаходження глобальних мінімумів.

Стохастичний градієнтний спуск (GSD, Gradient Descend) – це базовий алгоритм оптимізації, який став відправною точкою для більш сучасних методів. SGD на основі градієнту та швидкості навчання обчислює зміни параметрів, відповідно деякої підмножини даних (batch) (формула 2.19) [26].

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla L(\theta_t), \quad (2.19)$$

де θ_{t+1} та θ_t це відповідно значення параметру на кроці t та $t+1$;

η це параметр швидкості навчання;

$\nabla L(\theta_t)$ це градієнт функції втрат L відносно параметру θ .

SGD є дуже простим, і через це в деякому сенсі обмеженим, оптимізатором. В реальних актуальних задачах SGD не є методом який прийнято використовувати.

Adam (Adaptive Moment Estimation) – це більш сучасний, потужний та, ймовірно, найбільш популярний оптимізатор нейронних мереж. Adam

поєднує принципи та переваги двох інших алгоритмів: RMSProp (використовує адаптивну швидкість навчання) та Momentum (облік середнього напрямку градієнтів). Кожна вага оновлюється відповідно оцінці першого (середнє значення, загальна траєкторія) (формула 2.20) та другого (дисперсія)(формула 2.21) моментів. Додатково також обчислюється корекція відхилення (Bias Correction) для m_t (формула 2.22) та v_t (формула 2.23) [31].

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L(\theta_t), \quad (2.20)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla L(\theta_t))^2, \quad (2.21)$$

$$\hat{m}_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L(\theta_t), \quad (2.22)$$

$$\hat{v}_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla L(\theta_t))^2, \quad (2.23)$$

де β_1 та β_2 це експоненційні коефіцієнти розпаду (зазвичай 0.9 та 0.999 відповідно);

$\nabla L(\theta_t)$ це градієнт функції втрат L відносно параметру θ .

Безпосередньо оновлення параметру відбувається за наступною формулою (формула 2.24):

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}, \quad (2.24)$$

де θ_t це оновлений параметр;

θ_{t-1} це параметр до оновлення;

η це швидкість навчання;

$\hat{m}_t$ та $\hat{v}_t$ це кореговані перший та другий моменти відповідно;

ϵ це константа близька до 0.

Існує також оптимізатор AdamW (Adam with Weight Decay), який є модифікацією оригінального методу. Основна відмінність заключається в виправленні застосування L2 регуляризації. В оригінальному методі регуляризація застосовувалася по суті глобально, але це не зовсім коректно

адже для кожного окремого параметру оптимальне значення регуляризації є різним. Відповідно AdamW застосовує L2 регуляризацію після основного етапу оновлення ваг [32].

Варто також згадати інноваційні оптимізатори SAM (Sharpness-Aware Minimization) та GSAM (Gradient-based Sharpness-Aware Minimization), які ставлять на мету знайти не просто мінімум, а плаский мінімум функції втрат, що вважається призводить до кращого узагальнення моделі. Основна ідея в зміні задачі мінімізації. Замість мінімізації втрати в точці, SAM та GSAM мінімізують максимальні втрати в околі цієї точки, таким чином ставиться на мету комплексна задача мінімізації самої функції втрат та “нерівності ландшафту” функції втрат. Обидва оптимізатори є перспективними, але дещо виходять за межі задачі цієї роботи [33, 34].

2.1.5 Регуляризація

Регуляризація (Regularization) була згадано вище (розділ 2.1.3) в контексті оптимізатора AdamW. Регуляризація – це методи боротьби з проблемою перенавчання. Перенавчання відбувається, коли модель починає вивчати набір даних замість узагальненої оптимізації вирішення задачі. Методи регуляризації спрямовані на те, щоб зробити модель більш стійкою до коливань у вхідних даних, тим самим покращуючи її узагальнюючу здатність [26].

Два найбільш розповсюджених методів регуляризації це L1 (Lasso Regression) (формула 2.25) та L2 (Weight Decay) (формула 2.26) регуляризації, які додають штраф до функції втрат моделі, базуючись на розмірі її ваг, заохочуючи модель використовувати менші та простіші ваги.

$$L_{new} = L_{old} + \lambda \sum_{i=1}^n |w_i|, \quad (2.25)$$

$$L_{new} = L_{old} + \lambda \sum_{i=1}^n w_i^2, \quad (2.26)$$

де L_{new} це кінцеве значення функції втрат;
 L_{old} це початкове значення функції втрат;
 λ це гіпер параметр сили регуляризації;
 w_i це ваги моделі, n це кількість параметрів моделі.

Наведене вище L1 та L2 регуляризації застосовуються до машинного навчання універсально. Існують також методи розроблені спеціально для нейронних мереж, наприклад Dropout. Суть Dropout заключається в тому, що на кожному етапі навчання випадковим чином “вимикаються” деякі нейрони в шарі з певною ймовірністю.

Іншим важливим методом регуляризації в контексті нейронних мереж є Batch Normalization, суть якого в нормалізації входів шару таким чином, щоб середнє значення було нульове, а дисперсія дорівнювала одиниці (в контексті кожного окремого батчу) [35].

Крім методів які напряду взаємодіють з компонентами моделі, існують також методи регуляризації, які взаємодіють з даними. Аугментація даних (Data Augmentation) це техніка внесення випадкових змін в дані з метою генерації псевдо нових прикладів для покращення здібності мережі до узагальнення.

Варто також ранню зупинку (Early Stopping), що є дуже простим методом боротьби з перенавчанням. Якщо мережа не може покращити результати на валідаційному наборі (зазвичай декілька епох поспіль), навчання зупиняється.

2.2 Згорткові шари

Згорткові шари (Convolutional Layers) є основою для побудови згорткових нейронних мереж (Convolutional Neural Networks, CNNs), які спеціалізуються на роботі з даними сіткової структури, в частості – зображеннями. Основний принцип згорткових мереж нагадує дослідження про те як мозок обробляє візуальну інформацію (розділ 1.1.1), поверхневі згорткові шари вчаться розпізнавати загальну фічі, такі як контури та лінії, а більш глибокі шари вчаться розпізнавати більш комплексні елементи. Процес “видобування” елементів та деталей даних називається Feature Extraction. На прикладі знизу (рисунку 2.3) зображена демонстрація як може виглядати видобування “фічей”. На відміну від повністю зв'язних шарів, де кожен нейрон в одному шарі з'єднаний з кожним нейроном у попередньому шарі, згорткові шари використовують операцію згортки (convolution), яка дозволяє мережі автоматично вивчати ієрархічні ознаки без необхідності ручного вилучення ознак. Саме використовуючи операцію згортки мережа може видобувати “фічі” для подальшого навчання.

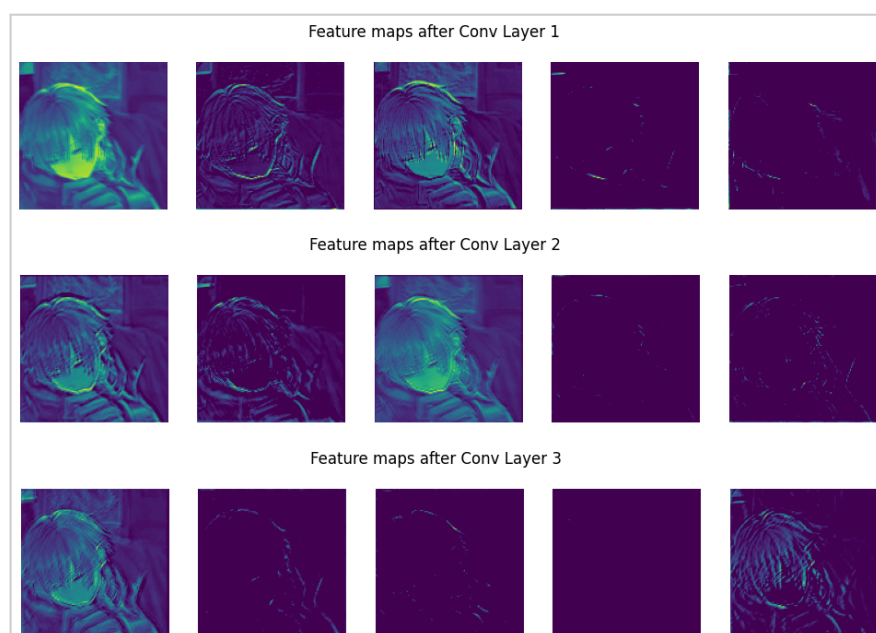


Рисунок 2.3 – Приклад Feature Extraction, ілюстрація від Asanokos [36]

Згортка (Convolution) може розглядатися в двох варіантах: неперервний та дискретний випадки. Математично згортка визначається як операція над двома функціями, яка породжує третю функцію, що описує, як одна функція "накладається" на іншу (формула 2.27):

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d\tau. \quad (2.27)$$

В контексті дискретних даних використовується відповідно дискретна інтерпретація операції згортки (формула 2.28):

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n - m]. \quad (2.28)$$

Безпосередньо для обробки кольорових зображень (3 канали) згортка записується наступним чином (формула 2.29):

$$S(i, j) = \sum_{c=1}^3 \sum_{m=0}^M \sum_{n=0}^N I_c(i + m, j + n) \cdot K_c(m, n) + b, \quad (2.29)$$

де $S(i, j)$ – значення вихідного пікселя;

c – кількість каналів;

$I_c(i + m, j + n)$ – вхідний піксель;

$K_c(m, n)$ – ядро згортки;

b – байєс, $M \times N$ це розмір ядра.

Ядро це по суті деяка матриця ваг, яка “проходить” по “поверхні” зображення для видобування “фічей”. Наведена формула використовується лише для обчислення деякої однієї згортки. Для отримання саме згорткового шару необхідно щоб фільтр “пройшов” по всьому зображенню. Відповідно формування саме згорткового шару можна визначити як всі можливі впорядковані операції згортки (формула 2.30):

$$O(I, K) = \{S(h, w)\}_{h,w}. \quad (2.30)$$

В реальних CNN зазвичай в шарі використовується декілька фільтрів для видобування найбільшої кількості корисних “фічей”. Ілюстрація

обчислення одного фільтру одноканального зображення розміром 3x4 ядром 2x2 наведена нижче (рисунок 2.4)

Як можна помітити, вихідна мапа “фічей” (Feature Map) має меншу розмірність, ніж вхідне зображення, що пов’язано з тим як працює операція згортки. Згортка не може обчислити значення для “крайних” пікселів через те, що вона також вимагає значення пікселів “в околі”. В цілому на розмір вихідного зображення впливають ще два параметра крім ядер: padding та strides. Padding це “рамка-буфер” нульових штучних пікселів навколо зображення, яка використовується для збереження оригінальної розмірності вхідних даних після операції згортки. Strides це “кроки” які робить фільтр по “поверхні” зображення.

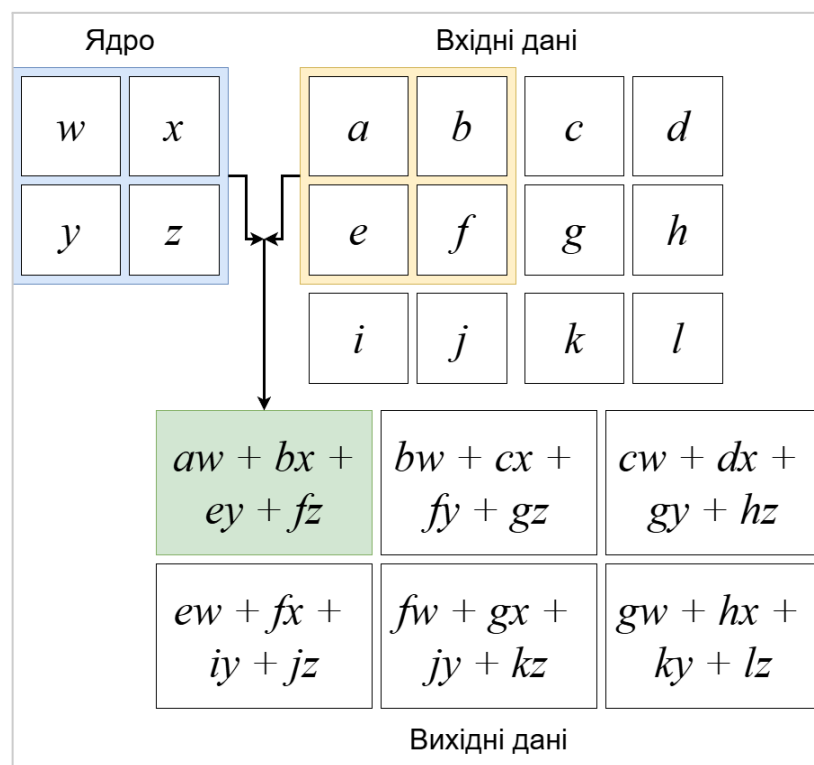


Рисунок 2.4 – Ілюстрація обчислення згортки даних 3x4 ядром 2x2

Обчислити розмір вихідного фільтру можна за наступними формулами відповідно для ширини (формула 2.31) та висота (2.32):

$$H' = \left\lfloor \frac{H + 2p + K_H}{s} \right\rfloor + 1, \quad (2.31)$$

$$W' = \left\lceil \frac{W + 2p + K_w}{s} \right\rceil + 1, \quad (2.32)$$

де $H \times W$ це розмір вхідного зображення;
 $H' \times W'$ це розмір вихідного зображення;
 $K_H \times K_W$ це розмір ядра;
 s – розмір кроку;
 p – розмір буферної зони.

Варто зазначити, що з принципу роботи та природи CNN впливає ряд важливих та корисних властивостей [26, 37].

1. Sparse Connectivity: на відміну від fully connected шарів, нейрони в CNN з'єднані лише з локальними ділянками вхідних даних, що дозволяє ефективно виявляти локальні ознаки, такі як краї чи текстури;
2. Parameter Sharing: для створення мапи “фічей” використовується лише одно ядро, відповідно кількість параметрів відносно невисока;
3. Translation Invariance: CNN зберігає здатність розпізнавати ознаки незалежно від їх положення на зображенні;
4. Ієрархічне вилучення ознак: багаторівнева ієрархічна структура ознак формується фактично автоматично завдяки природи CNN.

2.3. Squeeze and Excitation

Раніше в цій роботі (розділ 1.2.2) було згадано, що 2017 було представлено революційну роботу в глибоку навчанні – “Attention Is All You Need” [22]. Ця стаття ввела в новий механізм для NLP моделей, Attention блок, який дозволяє моделі “фокусувати увагу” на найбільш важливих аспектах вхідних даних. Замість того щоб обробляти всі ознаки однаково, механізм уваги фактично розставляє вагу пріоритету. Після видатного успіху

нового методу, його ідеї почали адаптувати для інших задач та типів даних. Squeeze-and-Excitation блок це один з варіантів адаптації Attention для CNN.

Squeeze-and-Excitation (SE) блок є простим механізмом уваги, який динамічно зважує важливість каналів у згорткових шарах [38]. На відміну від просторових механізмів уваги, які фокусуються на просторових локаціях (тобто конкретних фрагментах зображення), SE блок зосереджується на міжканальних взаємозв'язках, дозволяючи мережі зосереджуватися на найважливіших каналах (рисунок 2.5). Це покращує якість представлення ознак, не додаючи значних обчислювальних витрат.

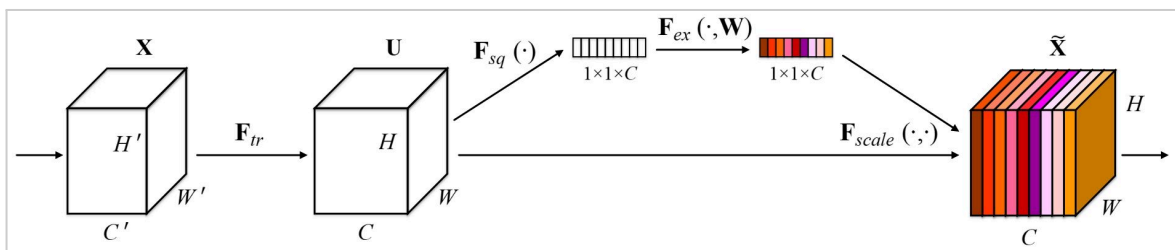


Рисунок 2.5 – SE блок [38]

SE блок складається з двох етапів.

1. Squeeze – з допомогою Global Average Pooling (GAP) кожна карта ознак перетворюється на скаляр, отримуючи таким чином вихідний вектор розміром $1 \times 1 \times C$, де C це кількість каналів.
2. Excitation – вихідний вектор попереднього етапу проходить через два повнозв'язних шари та вихідну сигмоїдну функції активації, отримуючи таким чином їх оцінку “важливості”.

2.4. Архітектура ResNet

ResNet (Residual Network, Залишкова мережа) архітектура була запропонована як вирішення для ряду проблем глибокого навчання: проблеми

зникаючого градієнту та проблеми деградації, які виникають при збільшенні розміру та глибини нейронної мережі. Дуже глибокі нейронні мережі є надскладними для традиційного тренування, але необхідні для комплексних задач. Головна ідея ResNet це фактично замінити задачу яку мережа вчиться виконувати на простішу, але ефективно аналогічну[39].

ResNet складається з блоків залишку (Residual block). Припустимо ми маємо деяку функцію $H(x)$ яку ми намагаємося апроксимувати нашою мережею, блок залишків замість функції $H(x)$ вчиться апроксимувати функцію $F(x)$ таку, щоб $H(x) = F(x) + x$, тобто ідея ResNet архітектури в тому, щоб замість тренування вирішення поставленої задачі напряду, мережа вчиться “доповненню до задачі” (рисунок 2.6), що є простішим завданням.

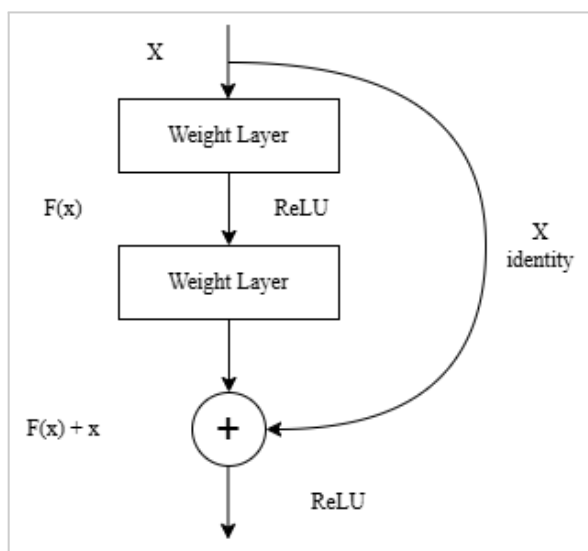


Рисунок 2.6 – ResNet Блок

2.5. Метрики оцінки продуктивності моделей

Для оцінки якості моделі необхідно підібрати коректні метрики. Багатоміткова класифікація суттєво відрізняється від класичної класифікації,

тому такі підходи як Confusion Matrix не дуже підходять, адже вони розраховані на один клас на один приклад.

Для багатоміткової класифікації здебільшого буде використано наступні метрики: F1-macro, F1-micro.

F1-macro це середнє значення F1-метрики, обчислене окремо для кожного класу, без урахування частоти класів, що робить її гарної оцінкою того наскільки добре модель впорюються з рідкісними класами. F1-macro (формула 2.33) обчислюється з використаннями $Precision_{macro}$ (формула 2.34) та $Recall_{macro}$ (формула 2.35)

$$F1_{macro} = 2 \cdot \frac{Precision_{macro} \cdot Recall_{macro}}{Precision_{macro} + Recall_{macro}} \quad (2.33)$$

$$Precision_{macro} = \frac{1}{N_{classes}} \sum_{k=1}^{N_{classes}} Precision_k \quad (2.34)$$

$$Recall_{macro} = \frac{1}{N_{classes}} \sum_{k=1}^{N_{classes}} Recall_k \quad (2.35)$$

F1-micro це глобальна метрика, яка обчислюється шляхом агрегації всіх TP, FP і FN перед обчисленням F1. На відміну від F1-macro, вона враховує частоту класів, тому вона репрезентує те наскільки гарно модель класифікує домінуючі класи (формула 2.36):

$$F1_{micro} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}, \quad (2.36)$$

де TP це сума істинно позитивних випадків всіх класів;

FP це сума хибно позитивних випадків всіх класів;

TN це сума істинно негативних випадків всіх класів.

2.6. Висновки до розділу 2

Було розглянуто та розібрано методи та інструменти побудови CNN моделі для багатоміткової класифікації. В якості архітектуру було обрано мережу ResNet для основи. Планується часткова реалізація Attention-архітектури через використання Squeeze-and-Excitation (SE) блоків. В зв'язку з дисбалансом даних було обрано використати спеціалізовану функцію витрат Focal Loss. Оцінка якості моделі буде проведено через метрики F1-micro, F1-macro та Binary Accuracy.

В зв'язку з обмеженням до обчислювальних можливостей, дана робота ставить на мету створення моделі на маленькому масштабі для доведення можливості реалізації задачі, та початкового поглибленого дослідження існуючої проблематики. Набір даних буде обмежено до 25к зображень, та 100 найпопулярніших загальних (описових) тегів.

РОЗДІЛ 3 РОЗРОБКА ТА ОЦІНКА МОДЕЛІ

3.1 Вибір мови програмування та фреймворків

Для реалізації задачі даній роботі було обрано мову програмування Python. Python є однією з найпопулярніших мов в сферах Data Science та Machine Learning через свою гнучкість, простоту та ефективність, що дозволяє його використовувати людям які не спеціалізуються в сфері комп'ютерної інженерії. Python також має велику кількість сучасних бібліотек, які спеціалізуються на машинному навчанні та обробці даних.

Розробка продукту, буде відбуватися на основі безкоштовного середовища від компанії Google, Colab. Colab є фактично онлайн аналогом Jupyter Notebook, який додає доступ до обчислювальних ресурсів запроваджених компанією Google, зокрема до потужних GPU та TPU ресурсів. Безкоштовна версія має ряд обмежень, які напряду впливають на подальший хід проекту: пам'ять середовища очищається з кінцем сесії, відповідно розмір датасету має бути достатньо малим, щоб не витратити забагато часу на початку кожної нової сесії виключно на завантаження набору даних; доступ до обчислювальних ресурсів GPU та TPU лімітований в 4 години, і відновлюється непередбачувану кількість днів, ресурси не гарантовані.

В якості фреймворку для моделювання нейронної мережі було обрано TensorFlow Keras, через те що він інтегрований в Google Colab, простий в використанні, та потужний.

3.2 Обробка датасету

За основу взят набір даних Tagged Anime Illustrations [40], який містить 337039 унікальних зображень. Дані пройшли попередню обробку: були відібрані виключно безпечні (SFW, non-explicit) зображення, всі ілюстрації нормалізовано до розміру 512x512 пікселів. Всі зображення в даній версії були створені до 2017 року включно, відповідно в датасеті не існує зображень в нових підтипах постійно еволютуючій стилістиці, також не існує зображень нових персонажів. Наведемо приклад випадкового зображення з датасету (рисунок 3.1).



Рисунок 3.1 – випадкове зображення з датасету. Автор – Inuno Rakugaki [41].

Зображенню вище відповідають наступні теги (перші 30): 1girl, >:), apron, black_hat, blue_hair, blue_skirt, character_name, cowboy_shot, dated, food, frilled_skirt, frills, fruit, hair_between_eyes, hand_on_hip, hat, hat_leaf, hinanawi_tenshi, inuno_rakugaki, long_hair, looking_at_viewer, over_shoulder, peach, puffy_short_sleeves, puffy_sleeves, rainbow_order, red_eyes, shirt, short_sleeves, skirt.

З прикладу можна побачити декілька особливостей датасету, що були зазначені в першому розділі: корелюючі теги (наприклад, “puffy_sleeves”, “puffy_short_sleeves”, та “short_sleeves”), абстрактні теги (наприклад, “>:”), який скоріше за все є аналогом тегу “smile”). В цілому, як видно з прикладу, система тегів дуже деталізована, що надає великий потенціал для машинного навчання різних задач, але це також робить задачу багатоміткової класифікації дуже складною.

В зв’язку з зазначеними обмеженнями, розмір зображень буде зменшено до 256x256, а кількість тегів лише до 100 найпопулярніших загальних тегів. Виведемо топ-20 тегів (рисунок 3.2).

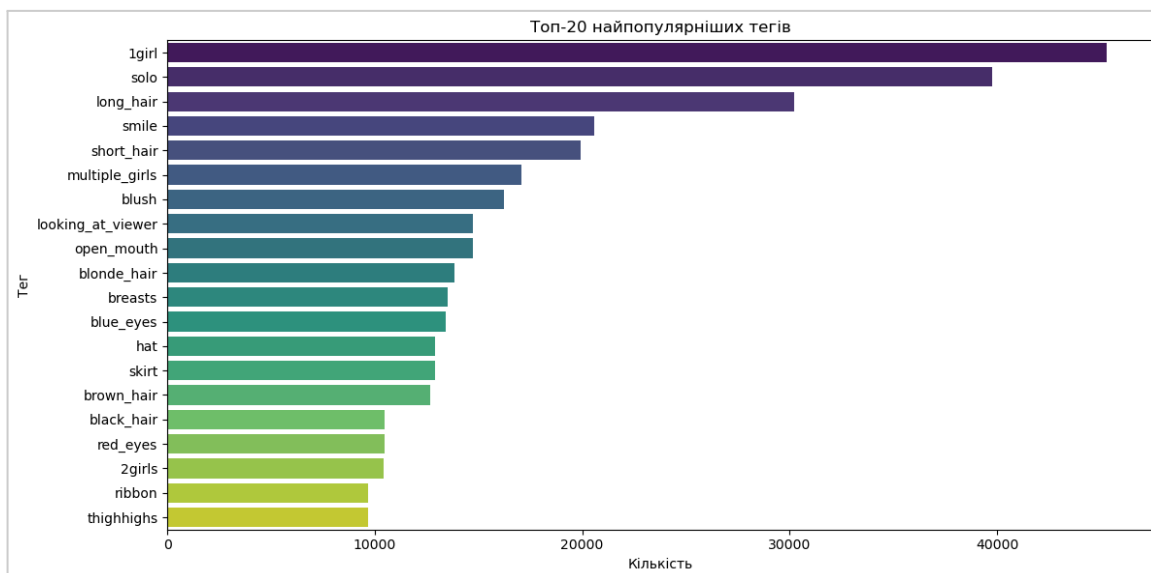


Рисунок 3.2 – Топ-20 тегів та їх кількість в обраному фрагменті датасету

Як можна побачити з рисунку вище (рисунок 3.2), популярність тегів спадає дуже швидко, і це враховуючи, що це все ще топ-20 найпопулярніших тегів серед тисяч існуючих. Це фрагмент датасету, але така картина бути повторюватися при створенні будь якого достатньо великого випадково піднабіру датасету. Давайте розглянемо популярності всіх ілюстрацію топ-100 тегів для подальшої ілюстрації (рис 3.3).

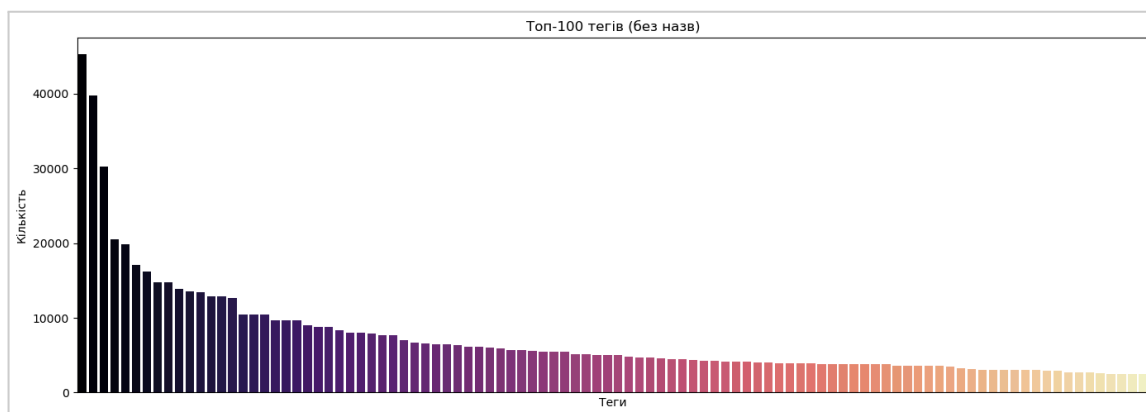


Рисунок 3.3 – Популярність топ-100 тегів

Як можна побачити з графіків (рисунку 3.2 та рисунку 3.3), в датасеті суворий дисбаланс тегів. Цікавим також є дуже явна різниця кількості зображень з чоловіками та жінками (рис. 3.4). Як можна побачити нижче, зображення з жінками займають приблизно 85% датасету, що додатково ускладнює навчання моделі коректно класифікувати зображення з чоловіками, та відрізняти жінок від чоловіків.

На чому закінчується огляд датасету. Підготовка даних, буде заключатися виключно в зменшенні розмірів зображень, та використанні фрагменту даних, та міток. Автор вважає, що фільтрування абстрактних тегів, та тегів з високою кореляцією не є вірним підходом в цій задачі адже подальший потенційний розвиток моделі має розширитися на всі можливі зображення, та максимально можливу кількість тегів, тому за мету роботи варто поставити дослідження та використання методів, що дозволяють

ефективно боротися з проблематикою датасету для подальшого потенційного скейлінгу моделі.

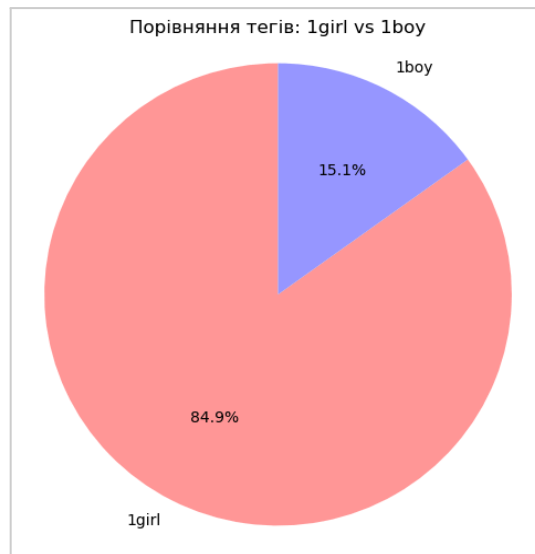


Рисунок 3.4 – Гендерний розподіл зображень.

3.3 Побудова архітектури моделі

Архітектура моделі базується на модифікації ResNet архітектури з використанням SE (Squeeze and Excitation) блоків для часткової реалізації Attention-архітектури. В якості функції втрат було обрано Focal Loss. Оптимізатором був обраний Adam. Вихідний шар використовує Sigmoid функцію активації. Всього в моделі 17,663,492 параметрів, 17,650,820 з яких можна тренувати. Основна архітектура моделі має наступну структуру.

1. Вхідний шар (Input): Приймає зображення розмірності 256x256 пікселів з трьома кольоровими каналами (RGB).
2. Початковий CNN блок: Спочатку йде згортковий шар, після якого йде batch normalization, функція активації ReLu.
3. MaxPooling2D.

4. ResNet блоки: Чотири ResNet блока з послідовним збільшенням кількості фільтрів. В кожному ResNet блоці також присутній SE блок.
5. GlobalAveragePooling2D
6. Dropout
7. Вихідний шар (Dense): Повністю зв'язаний шар з 100 вихідними нейронами та функцією активації Sigmoid.

3.4 Тренування та тестування та оцінка моделі

Після компіляції моделі, було приступлено до запуску тренування. Модель навчалася протягом 50 епох. По закінченню навчання модель було оцінено за F1-micro та F1-macro метриками на валідаційному наборі, також було обчислено бінарну точність (таблиця 3.1).

Таблиця 3.1 – Кінцеві метрики на валідаційному наборі

F1-score (Macro)	0.1865
F1-score (Micro)	0.3778
Binary accuracy	0.9020

Як видно вище (таблиця 3.1), суто з точку зору метрик модель не є ідеальною. F1-micro показує, що модель має деякий помітний успіх з більш популярними класами, в той час як F1-macro явно показує, що модель має суттєві проблеми з більш рідкими класами. Binary accuracy є контроль метрикою, яка показує, що навчання триває успішно (вона завищена через велику кількість коректних істино хибних прогнозів, але вона зростає та є високою, якщо модель навчається успішно під час тренування). Спочатку

розглянемо декілька демонстративних прикладів результатів прогнозів моделі, а потім ще раз оцінимо якість моделі, та проведемо порівняння з існуючими альтернативами. Розглянемо перший простий приклад (рис. 3.5).



Рисунок 3.5 – Класифікація тегів на зображенні коміксу. Автор Noise-111/Kakao. [42]

Як можна побачити вище, це абсолютно успішний випадок, модель навіть впоралася настільки гарно, що знайшла тег, який не був вказаний серед “справжніх” тегів. Даний приклад дозволяє підняти тему того, що досить часто не всі теги є вказаними, тому модель може бути наказана, а метрика погіршена, навіть якщо результат абсолютно коректний. Оцінити частоту відсутності тегів складно, але це досить часте явище. Розглянемо менш вдалий приклад (рис. 3.6).

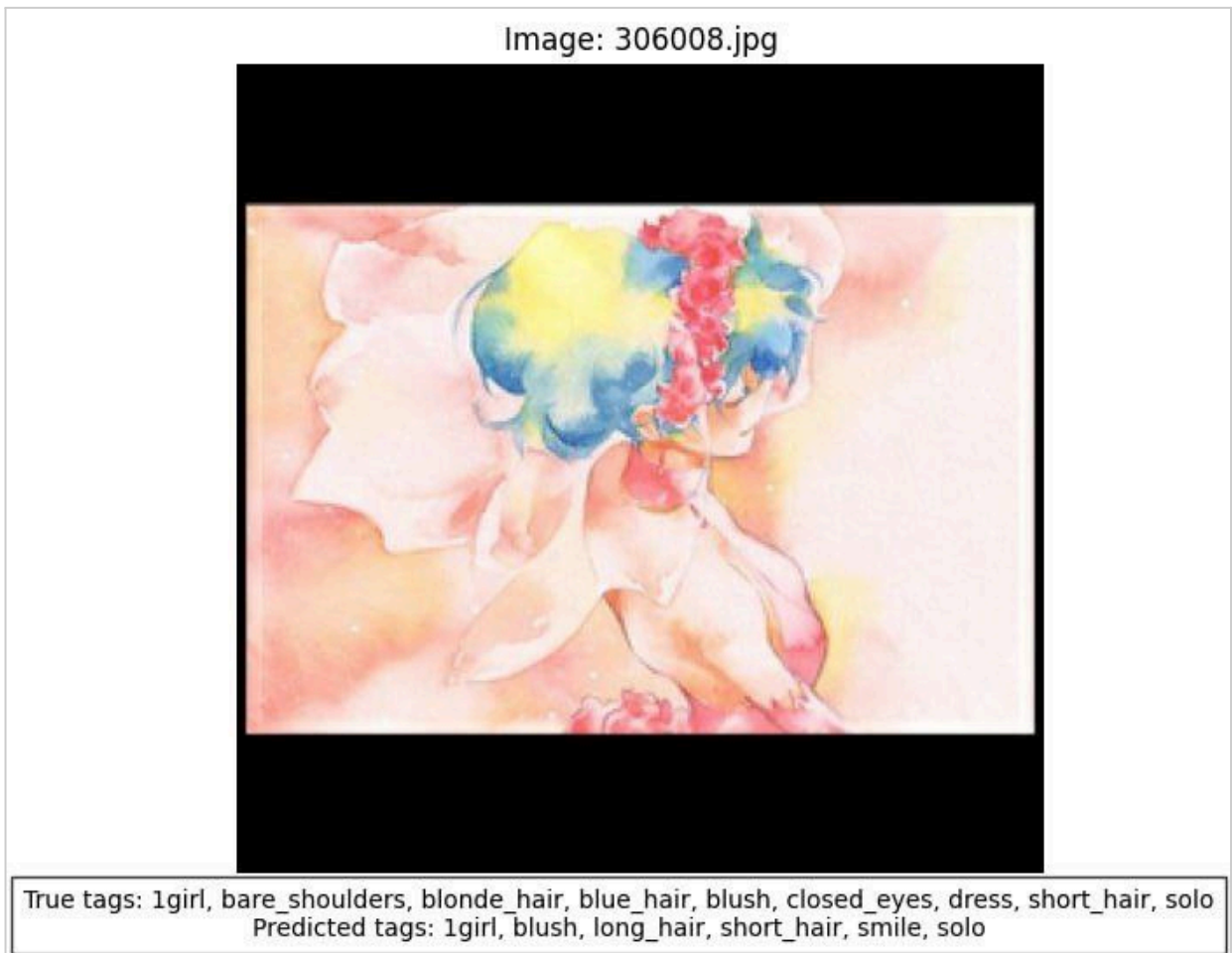


Рисунок 3.6 – Прогноз тегів на зображенні дівчині. Автор невідомий [43].

Розглянувши другий приклад, можна побачити, що навіть коли мережа має труднощі, вона робе прогнози в коректному напрямку.

Розглянемо декілька прикладів класифікації зображень з чоловіками. Нижче наведений третій приклад (рис. 3.7). На малюнку можна побачити хлопця якого невірно класифіковано як дівчину. З інших передбачених тегів можна зробити висновок, що модель прийняла плащ за довге чорне волосся і відповідно зробила висновок, що на зображенні дівчина.

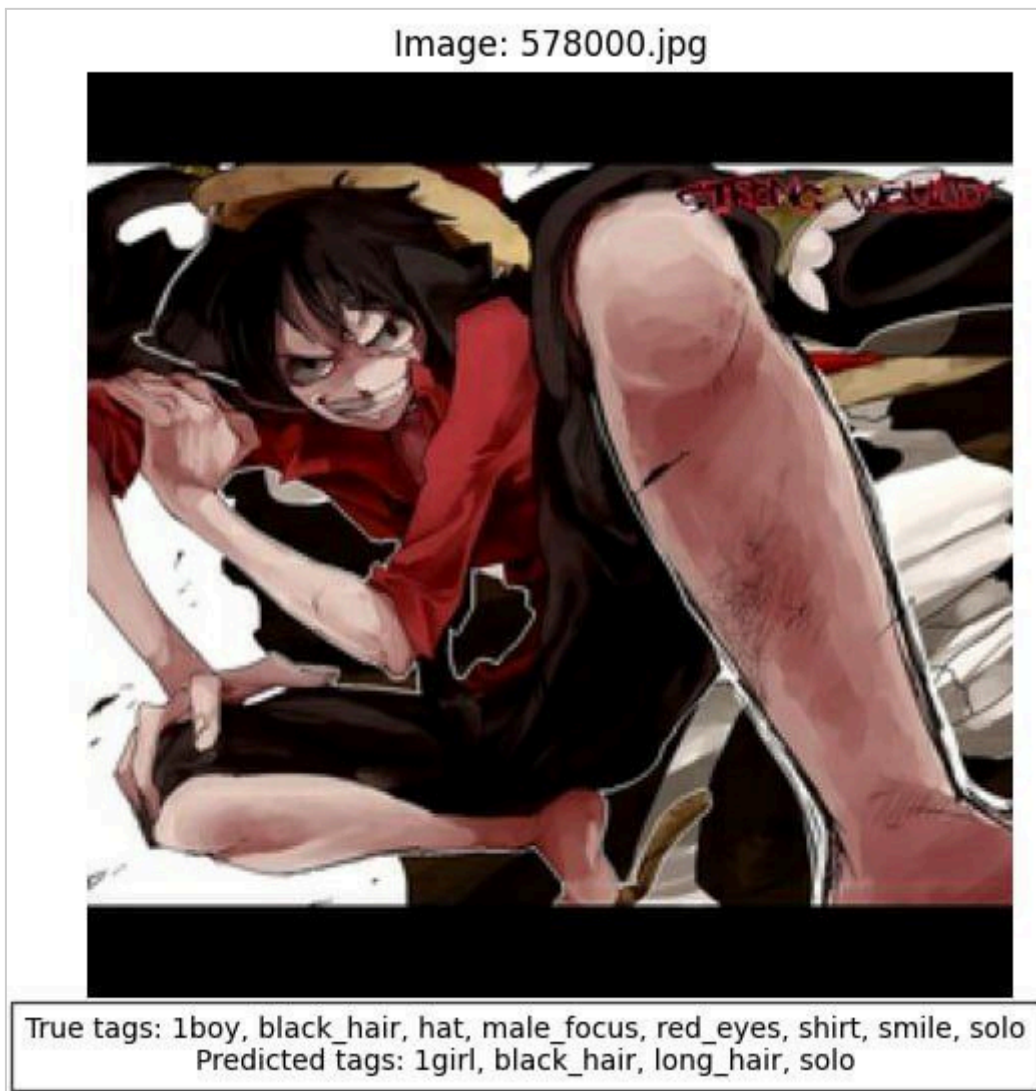


Рисунок 3.7 – Класифікація зображення Манкі Д. Луффі. Автор – Маррі [44].

Розглянемо вдалий приклад класифікації чоловіків (рис. 3.8). На зображенні можна побачити хлопця та дівчину одночасно, відповідно їх було правильно класифіковано. Модель скоріше за все намагалася використати тег “multiple_boys” в якості мітки наявності кількох персонажів на зображенні одночасно (як протилежність “solo”). Було знайдено відсутній тег “smile”. Також варто зазначити, що мережа скоріше спрогнозувала наявність тегу “blue_eyes” через кореляцію з тегом “blonde_hair”.



Рисунок 3.8 – Класифікація Асуни та Кіріто. Автор – Shinonome Jun [45].

Розглянемо останній приклад (рис. 3.9) (через кількість тегів їх буде подано як текст, а не фрагмент зображення), коли всі (або більшість) тегів передбачені коректно, але їх недостатньо.

True tags: 1girl, bangs, brown_hair, eyebrows_visible_through_hair, hair_between_eyes, hair_ribbon, heart, long_hair, looking_at_viewer, ribbon, short_sleeves, simple_background, skirt, smile, solo, white_background, yellow_eyes

Predicted tags: 1girl, bangs, blush, long_hair, looking_at_viewer, school_uniform, simple_background, skirt, smile, solo, thighhighs, white_background.

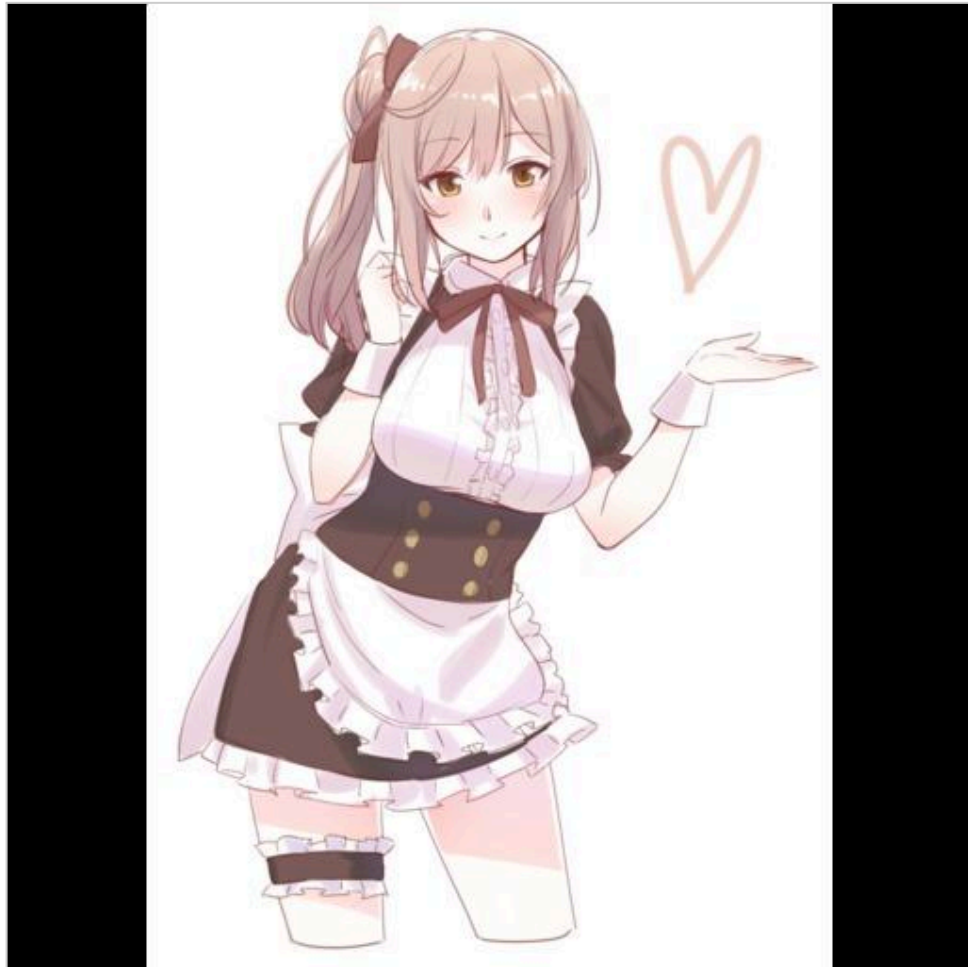


Рисунок 3.9 – Зображення горнічної. Автор – amatu08 [46].

Як можна побачити з прикладу, абсолютна більшість тегів, що було класифіковано, коректні (11/12), але є ще майже третина тегів, що мережа не класифікувала. Таким чином, незважаючи на те, що приклад емпірично є дуже успішним, він є досить посереднім з точки зору метрик.

Тепер розглянемо для порівняння іншу модель класифікації Camie Tagger [47], яку було знайдено в інтернеті (таблиця 3.2). В таблиці наведені характеристики моделей, та метрики на категорії загальних тегів. Якщо не зосереджуватися на абсолютно різних ступенях масштабу моделей, метрики відносно схожі, особливо F1-масго. Можна зробити тимчасовий висновок, що відносно потенційної можливої якості, результат роботи задовільний.

Таблиця 3.2 – Порівняння моделей багатоміткової класифікації

Модель	Кількість зображень	Кількість тегів	Кількість параметрів	F1-micro	F1-macro
Власна	25к	100	17м	0.3778	0.1865
Camie Tagger	7м	70к	214м	0.576	0.204

3.5 Висновки до розділу 3

В цьому розділі було спроектовано, побудовано, та натреновано модель багатоміткової класифікації аніме зображень. Було наведено опис архітектури моделі. За результатами аналізу метрик, розборі прикладів, та порівнянні з аналогічними модель, було проведено оцінку результатів роботи.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

В даному розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що розробляється.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу (ФВА).

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки моделі багатоміткової класифікації аніме-зображень. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема

підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по аніме-зображеннях.

Технічні вимоги до програмного продукту є наступні.

1. Висока точність класифікації: модель повинна забезпечувати максимально можливу точність присвоєння множинних міток аніме-зображенням.
2. Оптимальна швидкість прогнозування: модель має бути здатною до швидкої обробки нових зображень для класифікації (видача прогнозів), щоб забезпечити ефективну інтеграцію в системи, що працюють у режимі, близькому до реального часу.
3. Ефективне використання ресурсів: Модель повинна мати мінімальні вимоги до обчислювальних ресурсів (ОЗП, дисковий простір, потужність CPU/GPU) для навчання та виконання класифікації/прогнозування, що дозволить її розгортання на типових робочих станціях або серверах.
4. Можливість адаптації та донавчання: Архітектура моделі має дозволяти її подальше донавчання на нових даних або адаптацію під специфічні підмножини аніме-зображень, що є важливим для подальшого розвитку проекту.
5. Сумісність та інтегрованість: Модель повинна бути розроблена з використанням поширених фреймворків та стандартів, що забезпечить легку інтеграцію в інші програмні системи (наприклад, веб-додатки, бази даних зображень).
6. Мінімальні витрати на розробку та підтримку: Рішення та технології, обрані для розробки моделі, повинні забезпечувати оптимальний баланс між продуктивністю та економічною ефективністю розробки та подальшого супроводу, з урахуванням статусу MVP.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка моделі, яка дозволяє виконувати багатоміткову класифікацію аніме-зображень. Беручи за основу цю функцію, можна виділити наступні.

1. F_1 – вибір використання бібліотеки для побудови моделі на мові Python.
2. F_2 – вибір архітектури нейронної мережі та підходу до навчання.
3. F_3 – вибір методу оцінки та валідації результатів класифікації.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1 .

1. (А) Python (з бібліотекою TensorFlow).
2. (Б) Python (з бібліотекою PyTorch).
3. (В) Python (без використання базової бібліотеки).

Функція F_2 .

1. (А) Архітектура на основі ResNet
2. (Б) Архітектура на основі ConvNext
3. (В) Власна архітектура

Функція F_3 .

1. (А) Ручна експертна валідація.
2. (Б) Автоматизована валідація на розмічених наборах даних.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

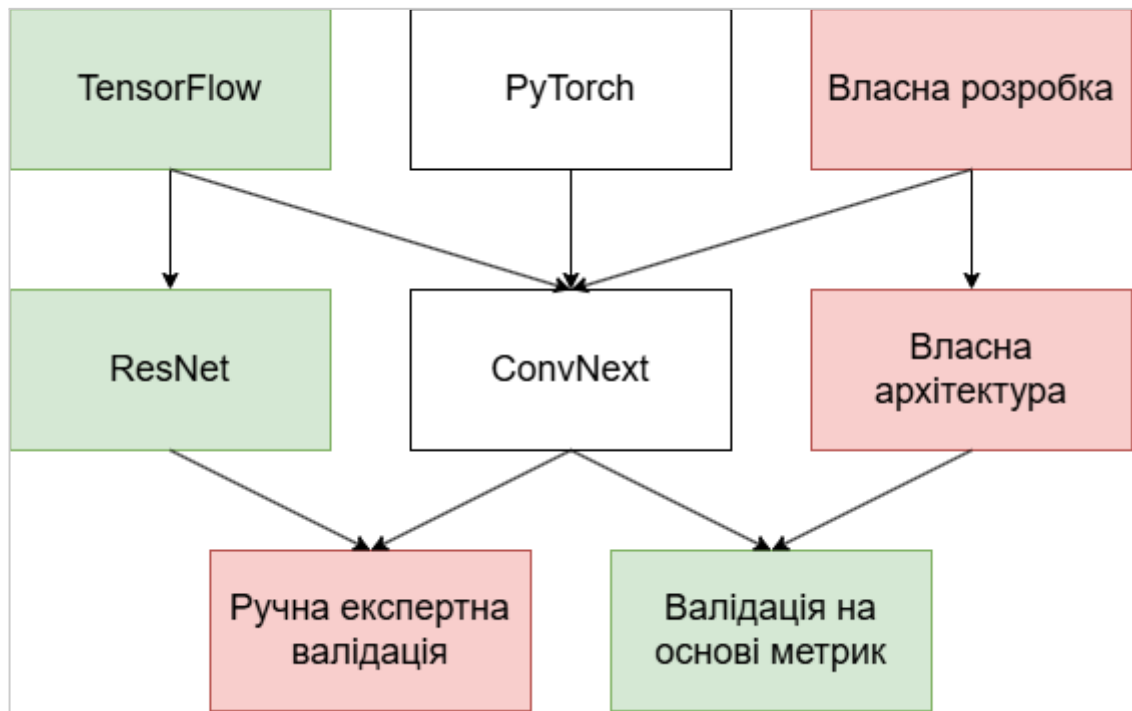


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 – Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	A	Широка екосистема, велика спільнота, багата документація. Висока простота використання. Наявність Keras API для швидкого прототипування.	Не підтримується на ОС Windows. Поступово втрачає актуальність. Менша гнучкість у низькорівневих операціях порівняно з PyTorch.

Продовження таблиці – 4.1

F_1	B	Більша гнучкість та динамічність. Зручний для дослідницьких проектів та швидкого прототипування.	Менша зрілість екосистеми. Підвищена складність використання.
	B	Повний контроль над кодом, можливість оптимізувати кожен частину моделі під специфічні потреби.	Надзвичайно висока складність розробки. Величезні часові витрати на реалізацію, відлагодження та підтримку.
F_2	A	Доведена ефективність для класифікації зображень. Багато попередньо навчених моделей доступні для трансферного навчання.	Може бути досить ресурсомісткою для дуже великих зображень або обмежених обчислювальних ресурсів.
	B	Нова та потужна архітектура. Ефективно масштабується.	Може потребувати більш сучасного апаратного забезпечення для ефективного навчання. Висока комплексність.

Кінець таблиці – 4.1

F_2	B	Максимальна гнучкість та можливість адаптації до унікальних характеристик аніме-зображень.	Високий ризик отримати неоптимальний результат.. Складно гарантувати високу точність та стабільність.
F_3	A	Забезпечує глибокий якісний аналіз помилок та нюансів класифікації. Дозволяє отримати унікальні інсайти.	Надзвичайно часомістка та ресурсомістка для великих обсягів даних. Суб'єктивність оцінки. Немає стандартизованої метрики для порівняння.
	B	Забезпечує об'єктивність та відтворюваність результатів. Дозволяє швидко оцінювати ефективність моделі на великих обсягах даних.	Не завжди може виявити всі нюанси та неочевидні помилки, які помітить людина.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці моделі деякі варіанти реалізації функцій варто відкинути, оскільки вони не відповідають поставленим перед дипломною роботою

задачам, особливо з огляду на статус MVP та обмеженість ресурсів. Ці варіанти відзначені у морфологічній карті червоним кольором.

Функція F_1 . Між TensorFlow та PyTorch обираємо (А) Python (з бібліотекою TensorFlow). Хоча PyTorch пропонує більшу гнучкість для досліджень, TensorFlow є більш поширеним у промисловості, має кращу підтримку для розгортання та оптимізації, що є важливим для перспективної інтеграції моделі. Крім того, TensorFlow має розвинений Keras API, що дозволяє швидко прототипувати та навчати моделі. Обраний варіант А.

Функція F_2 . Між ResNet та ConvNeXt обираємо (А) Архітектура на основі ResNet. ResNet є добре вивченою, перевіреною та широко використовуваною архітектурою з великою кількістю попередньо навчених моделей. Варто також зазначити, що ResNet є досить концептуально простою моделлю, що полегшує реалізацію та модифікування. Хоча ConvNeXt є новітнішим, він може вимагати більше ресурсів та експериментів для оптимального налаштування. Обраний варіант А.

Функція F_3 . Обираємо (б) Автоматизована валідація на розмічених наборах даних. Це єдиний об'єктивний та масштабований метод для оцінки ефективності моделі. Використання стандартизованих метрик на незалежному тестовому наборі даних дозволяє кількісно оцінити якість моделі та порівняти її з існуючими рішеннями, що є ключовим для демонстрації роботи MVP. Обраний варіант Б.

Таким чином, будемо розглядати такий варіант реалізації ПП:

$$F1a - F2a - F3b.$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри.

1. $X1$ – метрика F1-micro.
2. $X2$ – метрика F1-macro.
3. $X3$ – час навчання моделі.
4. $X4$ – кількість параметрів моделі.

Гірші, середні і кращі значення параметрів вибираються на основі загальних вимог до якості моделей машинного навчання та умов, що характеризують експлуатацію подібних моделей, як показано у таблиці 4.2. Ці значення є гіпотетичними, але відображають бажані діапазони.

Таблиця 4.2 – Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
F1-micro	$X1$	Відсотки	30	50	80
F1-macro	$X2$	Відсотки	20	30	40
Час навчання моделі	$X3$	Години	24	12	4
Кількість параметрів моделі	$X4$	Мільйони	100	50	10

За даними таблиці 4.3 будуються графічні характеристики параметрів –
рис. 4.2 – рис. 4.5.

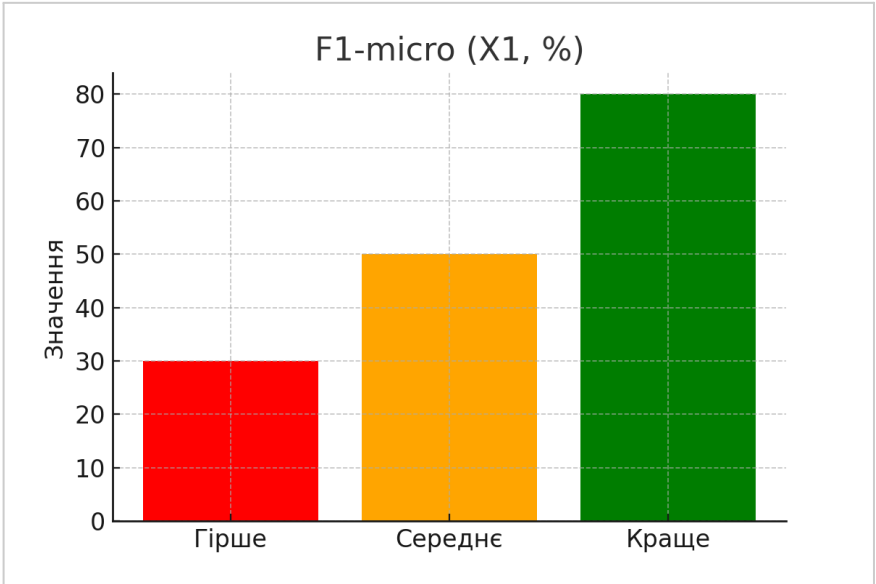


Рисунок 4.2 – X1, F1-micro

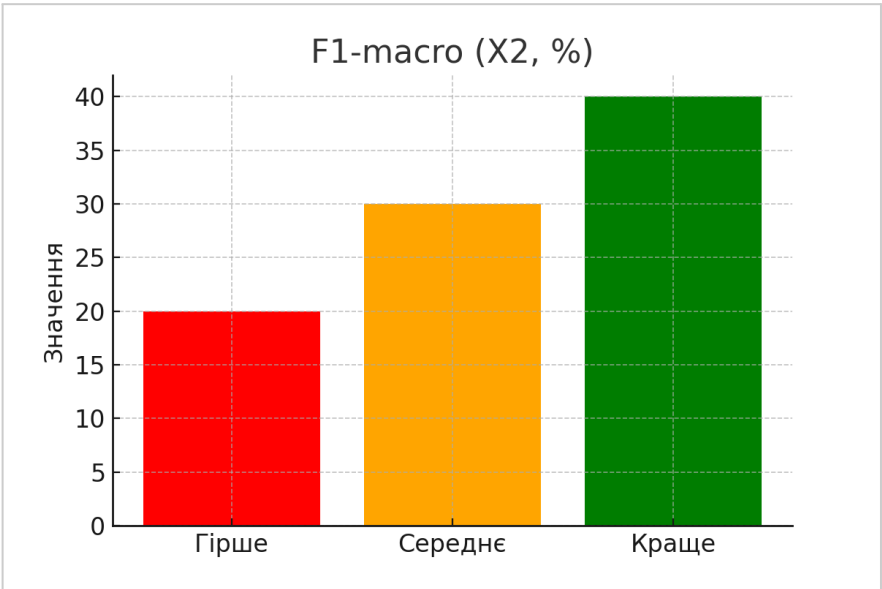


Рисунок 4.3 – X2, F1-macro

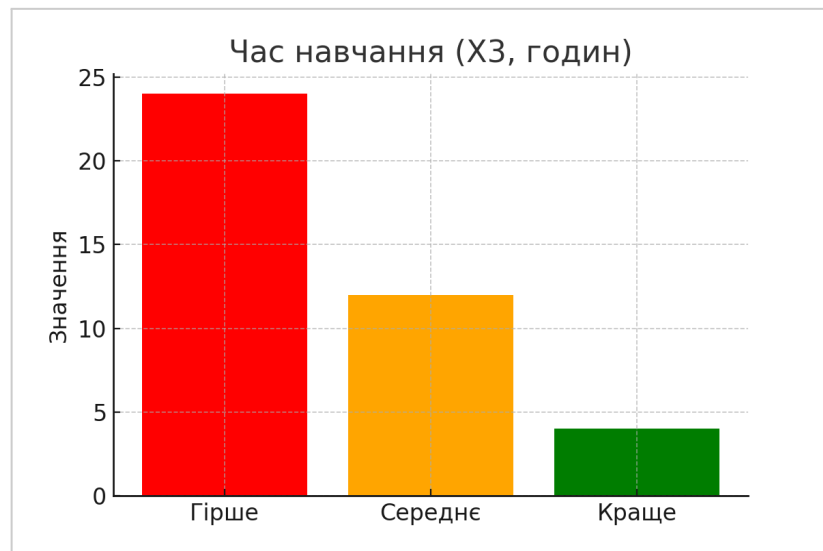


Рисунок 4.4 – Час навчання моделі

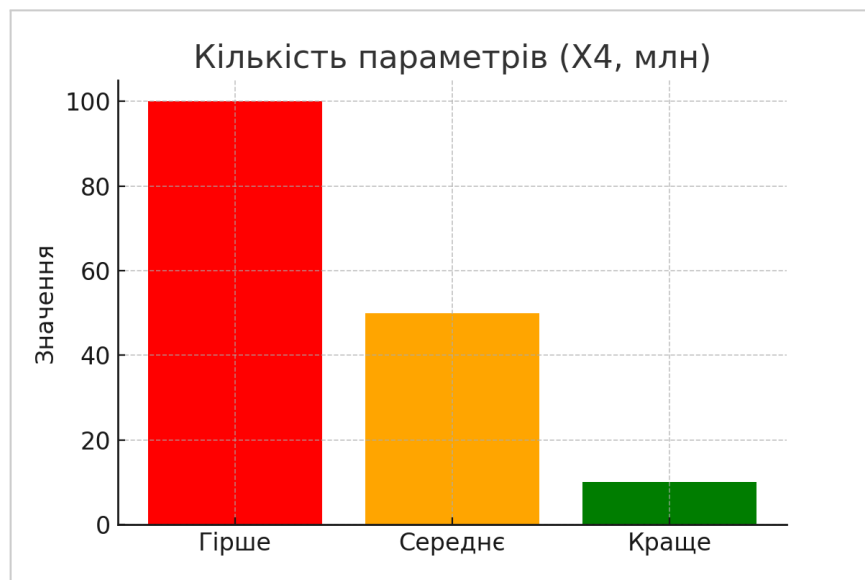


Рисунок 4.5 – X4, Кількість параметрів моделі

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні

Для перевірки оцінок, визначимо наступні параметри.

1. Сума рангів кожного з параметрів (формула 4.1) і загальна сума рангів (формула 4.2):

$$R_i = \sum_{j=1}^N r_{ij} \quad (4.1)$$

$$\frac{Nn(n+1)}{2} = 70, \quad (4.2)$$

де N – число експертів;

n – кількість параметрів.

2. Середня сума рангів (формула 4.3):

$$T = \frac{1}{n} R_{ij} = 17,5. \quad (4.3)$$

3. Відхилення суми рангів кожного параметра від середньої суми рангів (формула 4.4):

$$\Delta_i = R_i - T. \quad (4.4)$$

4. Загальна сума квадратів відхилення (формула 4.5):

$$S = \sum_{i=1}^N \Delta_i^2 = 205. \quad (4.5)$$

Порахуємо коефіцієнт узгодженості (формула 4.6):

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 205}{7^2(4^3-4)} = 0,837 > W_k = 0,67. \quad (4.6)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати заносимо у таблицю 4.4.

Таблиця 4.4 – Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	<	<	>	<	>	1.5
X1 і X3	>	>	>	>	>	>	>	>	1.5
X1 і X4	>	>	>	>	>	>	>	>	1.5
X2 і X3	>	>	>	>	>	>	>	>	1.5
X2 і X4	>	>	>	>	>	>	>	>	1.5
X3 і X4	<	>	<	<	<	<	>	<	0.5

Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається по формулі (формула 4.7):

$$a_{ij} = \begin{cases} 1.5 & \text{при } X_i > X_j \\ 1.0 & \text{при } X_i = X_j \\ 0.5 & \text{при } X_i < X_j \end{cases} \quad (4.7)$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості K_{bi} за наступними формулами (формула 4.8)(формула 4.9):

$$K_{bi} = \frac{b_i}{\sum_{i=1}^N b_i} \quad (4.8)$$

$$b_i = \sum_{i=1}^N a_{ij} \quad (4.9)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами (формула 4.10)(формула 4.11):

$$K_{bi} = \frac{b_i'}{\sum_{i=1}^N b_i'} \quad (4.10)$$

$$b_i' = \sum_{i=1}^N a_{ij} b_j' \quad (4.11)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i	K_{bi}	b_i^1	K_{bi}^1	b_i^2	K_{bi}^2
X1	1	1.5	1.5	1.5	5.5	0,34	21.25	0.36	77.85	0.36
X2	0.5	1	1.5	1.5	4.5	0,28	16.25	0.28	59.12	0.27
X3	0.5	0.5	1	0.5	2.5	0,16	9.25	0.16	34.16	0.16
X4	0.5	0.5	1.5	1	3.5	0,22	12.25	0.20	44.87	0.21
Всього:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6) (формула 4.12):

$$K_K(j) = \sum_{i=1}^n K_{bi,j} B_{i,j}, \quad (4.12)$$

де n – кількість параметрів;

K_{bi} – коефіцієнт вагомості i -го параметра;

B_i – оцінка i -го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	F1	F2		F3
Варіант реалізації функції	А	А		Б
Параметри	X1	X2	X3	X4
Абсолютне значення параметра	50	30	12	50
Бальна оцінка параметра	26	23	9	12
Коефіцієнт вагомості параметра	0.36	0.27	0.16	0.21
Коефіцієнт рівня якості	9.36	6.21	1.44	2.52

За даними з таблиці 4.6 за формулою (формула 4.13):

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \dots + K_{\text{ТУ}}[F_{zk}], \quad (4.13)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 9.36 + 6.21 + 1.44 = 17.01,$$

$$K_{K2} = 9.36 + 6.21 + 2.52 = 18.09.$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості пов'язаний з реалізацію основних завдань.

1. Розробка моделі класифікації.
2. Обробка вхідних даних.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритмів, які використовуються в завданні 1 належать до групи 3; а в завданні 2 – до групи 1.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як (формула 4.14):

$$T_o = T_p \cdot K_p \cdot K_{CK} \cdot K_M \cdot K_{CT} \cdot K_{CT.M}, \quad (4.14)$$

де T_p – трудомісткість розробки ПП;

K_p – поправочний коефіцієнт;

K_{CK} – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

K_{CT} – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{CT.M}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_p = 42$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_{\Pi} = 1.8$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 2: $K_{СК} = 1.53$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ} = 0.6$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 42 \cdot 1.8 \cdot 0.6 \cdot 1.53 = 69,4 \text{ людино-днів.}$$

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_p = 33$ людино-днів, $K_{\Pi} = 1.5$, $K_{СК} = 1.6$, $K_{СТ} = 0.5$:

$$T_2 = 33 \cdot 1.5 \cdot 1.6 \cdot 0.5 = 39.6 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_0 = (69,4 + 39.6) \cdot 8 = 872 \text{ людино-годин.}$$

В розробці бере участь один програміст з окладом 60000 грн. Визначаємо середню зарплату за годину за формулою (формула 4.15):

$$C_{\text{ч}} = \frac{M}{T_m \cdot t} = \frac{60000}{21 \cdot 8} = 357.14 \text{ грн. /год.,} \quad (4.15)$$

де M – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

Тоді, розраховуємо заробітну плату (формула 4.16):

$$C_{\text{зп}} = C_{\text{ч}} \cdot T_i \cdot K_d, \quad (4.16)$$

де $C_{\text{ч}}$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

K_d – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I. } C_{\text{ЗП}} = 357.14 \cdot 1227.2 \cdot 1.2 = 525942.8 \text{ грн.}$$

$$\text{II. } C_{\text{ЗП}} = 357.14 \cdot 868.88 \cdot 1.2 = 372291.4 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I. } C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 525942.8 \cdot 0.22 = 115707.4 \text{ грн.}$$

$$\text{II. } C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 372291.4 \cdot 0.22 = 81904.1 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. (C_M)

Так як одна ЕОМ обслуговує одного програміста з окладом 40000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_T = 12 \cdot M \cdot K_3 = 12 \cdot 60000 \cdot 0.2 = 144000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{\text{ЗП}} = C_T \cdot (1 + K_3) = 144000 \cdot (1 + 0.2) = 144000 \cdot 1.2 = 172800 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 172800 \cdot 0.22 = 38016 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25 000 грн.

$$C_A = K_{\text{ТМ}} \cdot K_A \cdot Ц_{\text{ПР}} = 1.05 \cdot 0.25 \cdot 25000 = 6562.5 \text{ грн.}$$

де $K_{\text{ТМ}}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A – річна норма амортизації;

$Ц_{\text{ПР}}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{\text{ТМ}} \cdot Ц_{\text{ПР}} \cdot K_P = 1.05 \cdot 25000 \cdot 0.05 = 1312.5 \text{ грн.,}$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$T_{\text{ЕФ}} = (D_K - D_B - D_C - D_P) \cdot t_3 \cdot K_B = 745,6 \text{ години,}$$

де D_K – календарна кількість днів у році;

D_B, D_C – відповідно кількість вихідних та святкових днів;

D_p – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_C \cdot K_3 \cdot C_{\text{ЕН}} = 745.6 \cdot 0.4 \cdot 0.2 \cdot 9.43 = 562.48 \text{ грн.},$$

де N_C – середньо-споживана потужність приладу;

K_3 – коефіцієнтом зайнятості приладу;

$C_{\text{ЕН}}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_H = C_{\text{ПР}} \cdot 0.67 = 25000 \cdot 0.67 = 16750 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_A + C_P + C_{\text{ЕЛ}} + C_H, \quad (4.17)$$

$$C_{\text{ЕКС}} = 172800 + 38016 + 6562.5 + 1312.5 + 157.17 + 16750 = 235698.17 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{ЕКС}} / T_{\text{ЕФ}} = 235698.17 / 745.6 = 316.12 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_M = C_{\text{М-Г}} \cdot T, \quad (4.18)$$

$$\text{I. } C_M = 316.12 \cdot 1227.2 = 387926.8 \text{ грн.}$$

$$\text{II. } C_M = 316.12 \cdot 868.88 = 274577.8 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{\text{ЗП}} \cdot 0.67, \quad (4.19)$$

$$\text{I. } C_H = 525942.8 \cdot 0.67 = 352381.7 \text{ грн.}$$

$$\text{II. } C_H = 372291.4 \cdot 0.67 = 249435.2 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{\text{ПП}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{М}} + C_{\text{Н}}, \quad (4.20)$$

$$\text{I. } C_{\text{ПП}} = 525942.8 + 115707.4 + 387926.8 + 352381.7 = 1381958.7 \text{ грн.}$$

$$\text{II. } C_{\text{ПП}} = 372291.4 + 81904.1 + 274577.8 + 249435.2 = 978208.5 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{\text{К}j} / C_{\text{Ф}j}, \quad (4.20)$$

$$K_{\text{ТЕР}1} = 20,4 / 1381958.7 = 0.00001476,$$

$$K_{\text{ТЕР}2} = 16,08 / 978208.5 = 0.00001644.$$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{\text{ТЕР}2} = 0.00001644$.

Після виконання функціонально-вартісного аналізу програмного комплексу, що розробляється, можна зробити висновок, що з альтернатив, які залишилися після першого відбору, оптимальним є другий варіант реалізації програмного продукту. Хоча його коефіцієнт якості трохи нижчий, ніж у першого варіанта, його значно менша вартість розробки його значно менша вартість розробки.

Цей варіант реалізації програмного продукту має такі параметри:

1. Python (з бібліотекою TensorFlow).
2. Архітектура на основі ResNet.
3. Автоматизована валідація на розмічених наборах даних.

Цей варіант виконання програмного комплексу може гарантувати оптимальний баланс між якістю та вартістю реалізації, що робить його привабливим вибором для проекту.

4.8 Висновки до розділу 4

У цій частині дипломної роботи ми провели комплексний аналіз функціональних аспектів та економічної ефективності розроблюваного програмного продукту. Спочатку ми оцінили важливість ключових параметрів якості, а потім розраховали, наскільки добре кожен варіант реалізації функцій відповідає цим критеріям. Паралельно було проведено детальний економічний аналіз трудомісткості розробки, що дозволило визначити загальну вартість кожного варіанта та обрати найкращий.

ВИСНОВКИ

У рамках даної дипломної роботи було успішно досліджено проблему багатоміткової класифікації стилізованих зображень із застосуванням глибоких нейронних мереж. Була спроектована, побудована та натренована модель багатоміткової класифікації на основі ResNet-архітектури, посиленої блоками Squeeze-and-Excitation. Це дозволило ефективно працювати з великим об'ємом даних, складними візуальними патернами та дисбалансом в даних.

Для навчання моделі було використано датасет з 25 тисяч стилізованих аніме-зображень, а класифікація проводилася за 100 найпопулярнішими тегами. Фінальна модель, що має 17 мільйонів параметрів, продемонструвала такі ключові показники якості: $F1\text{-micro} = 0.3778$ та $F1\text{-macro} = 0.1865$. Незважаючи на складнощі, пов'язані з великим дисбалансом класів та багатомітковою природою даних, досягнуті результати є прийнятними та підтверджують ефективність обраного підходу. Детальний розбір прикладів класифікації підтвердив здатність моделі коректно розпізнавати найбільш поширені теги. Також модель довела здатність знаходити тегів, які не було вказано користувачами в оригінальних даних.

В першому розділі було розглянуто історію комп'ютерного зору від початку і до поточного моменту. Було розглянуто основні методи та сфери застосування комп'ютерного зору.

В другому розділі було детально розібрано: математичні основи нейронних мереж та згорткових шарів, архітектуру ResNet, Squeeze-and-Excitement блоки, сучасні актуальні та перспективні розробки в сфері класифікації зображень нейронними мережами.

В третьому розділі було: оброблено набір даних, побудовано та натреновано модель багатоміткової класифікації, проведено оцінку моделі.

В рамках подальшого дослідження пропонується використання більшої кількості даних, розширення кількості тегів, посилення архітектури мережі з допомогою новітніх архітектур (таких як ConvNext), застосування візуальних трансформерів, імплементація оптимізатора GSAM, спробувати адаптувати функцію втрат Unified Loss для задачі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is computer vision? *IBM*. 27.06.2021. URL: <https://www.ibm.com/think/topics/computer-vision> (date of access: 17.05.2025).
2. Introduction to Computer Vision. *University of San Diego*. URL: <https://onlinedegrees.sandiego.edu/introduction-to-computer-vision/> (date of access: 17.05.2025).
3. Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *Journal of Physiology*, Vol. 148, no. 3, P. 574–591.
4. The First Digital Image. National Institute of Standards and Technology. URL: <https://www.nist.gov/mathematics-statistics/first-digital-image> (date of access: 17.05.2025).
5. Marr D. Vision: A Computational Investigation into the Human Representation and Processing of Visual Information. San Francisco : W. H. Freeman and Company, 1982. 398 P.
6. Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*. 1980. Vol. 36, P. 193–202.
7. Viola P., Jones M. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA, 8–14 dec. 2001.
8. ImageNet: A large-scale hierarchical image database / J. Deng та ін. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, 20–25 jul. 2009.

9. Krizhevsky, A., Sutskever, I., & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, Vol. 25, P. 1097–1105. 2012.
10. Artificial intelligence-driven computer vision for autonomous vehicles: A review / Nuraini Diah Novianti та ін. *Journal of Traffic and Transportation Engineering (English Edition)*, Vol. 11(3), P. 389–412.
11. A survey on deep learning in medical image analysis / Litjens, G та ін. *Medical Image Analysis* , Vol. 42, P. 60–88. 2017.
12. Wariach M., Reinbacher T. Visual Inspection AI: a purpose-built solution for faster, more accurate quality control. *Google Cloud. Blog*. 23.06.2021. URL: <https://cloud.google.com/blog/products/ai-machine-learning/improve-manufacturing-quality-control-with-visual-inspection-ai> (date of access: 18.05.2025).
13. Як отримати Дія.Підпис? Міністерство цифрової трансформації України. URL: <https://paperless.diia.gov.ua/instruction/yak-otrimati-diyapidpis> (дата звернення : 17.05.2025).
14. Mozur P. Inside China's Dystopian Dreams: A.I., Shame and Lots of Cameras. *The New York Times*. 08.07.2018. URL: <https://www.nytimes.com/2018/07/08/business/china-surveillance-technology.html> (date of access: 18.05.2025).
15. Integrating Computer Vision (CV) models with the Spot robot. *Boston Dynamics*. URL: <https://support.bostondynamics.com/s/article/Integrating-Computer-Vision-CV-models-with-the-Spot-robot> (date of access: 17.05.2025).
16. Computer Vision in AR and VR – The Complete Guide. *Viso.ai*. URL: <https://viso.ai/computer-vision/augmented-reality-virtual-reality/> (date of access: 18.05.2025).

17. Computer Vision Market Size, Share & Trends Analysis Report By Component (Hardware, Software), By Product (Smart Camera-Based, PC-Based), By Application, By Vertical, By Region, And Segment Forecasts, 2025 – 2030. *Grand View Research*. URL: <https://www.grandviewresearch.com/industry-analysis/computer-vision-market> (date of access: 17.05.2025).
18. Computer Vision – Worldwide. *Statista*. URL: <https://www.statista.com/outlook/tmo/artificial-intelligence/computer-vision/worldwide> (date of access: 17.05.2025).
19. Tsoumakas, G., Katakis, I., & Vlahavas, I. Mining multi-label data. *The Data Mining and Knowledge Discovery Handbook* / O. Maimon, L. Rokach. Springer, 2005. P. 667–685.
20. Bhatnagar, R., Sharma, A. A comprehensive review on multi-label image classification: *Approaches and applications*. *Electronics*, Vol. 14(21), no. 9863. 2024
21. Token-based Image Representation and Processing for Computer Vision. *arXiv*. 2021. URL: <https://arxiv.org/abs/2102.07113> (date of access: 17.05.2025).
22. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. Attention Is All You Need. *arXiv*. 2017. URL: <https://arxiv.org/abs/1706.03762> (date of access: 17.05.2025).
23. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Georgiou, G., & Gelly, S. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv*. 2020. URL: <https://arxiv.org/abs/2010.11929> (date of access: 17.05.2025).
24. Gwern Branwen. Danbooru2021. *Gwern*. 14.8.2021. URL: <https://gwern.net/danbooru2021> (date of access: 17.05.2025).

25. nyanko7. Danbooru2023. *Hugging Face*. URL: <https://huggingface.co/datasets/nyanko7/danbooru2023> (date of access: 17.05.2025).
26. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Cambridge, MA: MIT Press.
27. Bengio, Y., Simard, P., & Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*. 1994. Vol. 5, no. 2. P. 157–166.
28. Hendrycks, D., & Gimpel, K. Gaussian Error Linear Units (GELUs). arXiv. 2016. URL: <https://arxiv.org/abs/1606.08415> (date of access: 17.05.2025).
29. Ramachandran, P., Zoph, B., & Le, Q. V. Searching for Activation Functions. *arXiv*. 2017. URL: <https://arxiv.org/abs/1710.05941> (date of access: 17.05.2025).
30. Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. Focal Loss for Dense Object Detection. *Proceedings of the IEEE International Conference on Computer Vision*. 2017. P. 2980–2988.
31. Kingma, D. P., & Ba, J. Adam: A Method for Stochastic Optimization. *arXiv*. 2014. URL: <https://arxiv.org/abs/1412.6980> (date of access: 17.05.2025).
32. Loshchilov, I., & Hutter, F. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations (ICLR)*. 2019.
33. Foret, P., Kleiner, A., Mobahi, H., & Neyshabur, B. Sharpness-Aware Minimization for Efficiently Finding Flat Minima. *International Conference on Learning Representations (ICLR)*. 2020.
34. Liu, X., Liu, C., Chen, Z., Wang, C., Li, S., & Li, W. GSAM: Gradient-based Sharpness-Aware Minimization for Efficiently Finding Flat Minima. In *Neural Information Processing Systems (NeurIPS)*. 2022.

35. Ioffe, S., & Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. URL: <https://arxiv.org/abs/1502.03167> (date of access: 17.05.2025).
36. SafeBooru. 23.05.2017. URL: <https://safebooru.org/index.php?page=post&s=view&id=2233782> (date of access: 29.05.2025).
37. LeCun, Y., Bengio, Y., & Hinton, G. Deep learning. *Nature*. 2015. Vol. 521, no. 7553. P. 436–444.
38. Hu, J., Shen, L., & Sun, G. Squeeze-and-Excitation Networks. *arXiv*. 2017. URL: <https://arxiv.org/abs/1709.01507v4> (date of access: 29.05.2025).
39. He, K., Zhang, X., Ren, S., & Sun, J. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf (date of access: 29.05.2025).
40. O'Neill, M. Tagged Anime Illustrations. *Kaggle*. URL: <https://www.kaggle.com/datasets/mylesoneill/tagged-anime-illustrations> (date of access: 17.05.2025).
41. rakugakiinuno. *Twitter*. 12.12.2016. URL: <https://twitter.com/rakugakiinuno/status/808309181988028417> (date of access: 17.05.2025).
42. かかお@例大祭T-27a. *pixiv*. 13.9.2013. URL: <https://www.pixiv.net/en/artworks/33855769> (date of access: 17.05.2025).
43. Safebooru. 9.03.2010. URL: <https://safebooru.org/index.php?page=post&s=view&id=271007> (date of access: 17.05.2025).
44. mapi. *Safebooru*. 1.03.2010. URL: <https://safebooru.org/index.php?page=post&s=view&id=94146> (date of access: 17.05.2025).

45. Shinonome un. *Danbooru.* URL:
<https://danbooru.donmai.us/posts/1256001> (date of access: 17.05.2025).
46. amatuyu08. *Twitter.* 11.10.2016. URL:
<https://x.com/amatuyu08/status/785767347344674816> (date of access:
17.05.2025).
47. Camais03. Camie-tagger. *Hugging Face.* URL:
<https://huggingface.co/Camais03/camie-tagger> (date of access: 17.05.2025).

ДОДАТОК А ПРОГРАМНИЙ КОД

```

@register_keras_serializable()
class SEBlock(layers.Layer):
    def __init__(self, ratio=0.25, **kwargs):
        super(SEBlock, self).__init__(**kwargs)
        self.ratio = ratio

    def build(self, input_shape):
        self.channels = input_shape[-1]
        self.squeeze_units = max(1, int(self.channels *
self.ratio))

        self.squeeze = layers.Dense(self.squeeze_units,
activation='relu')
        self.excite = layers.Dense(self.channels,
activation='sigmoid')

    def call(self, inputs):
        x = tf.reduce_mean(inputs, axis=[1, 2], keepdims=True)
        x = self.squeeze(x)
        x = self.excite(x)
        return inputs * x

    def get_config(self):
        config = super().get_config()
        config.update({
            "ratio": self.ratio
        })
        return config

```

```

@register_keras_serializable()
def resnet_block(x, filters, kernel_size=3, stride=1,
use_se=False):
    shortcut = x

    # Conv1
    x = layers.Conv2D(filters, kernel_size, strides=stride,
padding='same',
                        kernel_initializer='he_normal')(x)
    x = layers.BatchNormalization()(x)
    x = layers.ReLU()(x)

    # Conv2
    x = layers.Conv2D(filters, kernel_size, padding='same',
                        kernel_initializer='he_normal')(x)
    x = layers.BatchNormalization()(x)

    # Shortcut connection
    if stride != 1 or shortcut.shape[-1] != filters:
        shortcut = layers.Conv2D(filters, 1, strides=stride,
kernel_initializer='he_normal')(shortcut)
        shortcut = layers.BatchNormalization()(shortcut)

    # SE Block
    if use_se:
        x = SEBlock()(x)

    x = layers.Add()([x, shortcut])
    x = layers.ReLU()(x)

```

```

return x

@register_keras_serializable()
def unified_focal_loss(y_true, y_pred, alpha=0.5, gamma=2.0):

    bce = tf.keras.backend.binary_crossentropy(y_true, y_pred)

    p_t = y_true * y_pred + (1 - y_true) * (1 - y_pred)
    focal_weight = tf.pow(1.0 - p_t, gamma)

    loss = alpha * focal_weight * bce
    return tf.reduce_mean(loss)

def build_model(input_shape=(256, 256, 3), num_classes=100,
use_se=True):
    inputs = layers.Input(shape=input_shape)

    x = layers.Conv2D(64, 7, strides=2, padding='same',
                      kernel_initializer='he_normal')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.ReLU()(x)
    x = layers.MaxPooling2D(3, strides=2, padding='same')(x)

    # ResNet блоки
    filters = [64, 128, 256, 512]
    strides = [1, 2, 2, 2]

    for i, (f, s) in enumerate(zip(filters, strides)):
        for j in range(2 if i < 2 else 3): # Більше блоків у
глибоких шарах

```

```

        x = resnet_block(x, f, stride=s if j == 0 else 1,
use_se=use_se)

```

```

    # Фінальні шари

```

```

    x = layers.GlobalAveragePooling2D()(x)

```

```

    x = layers.Dropout(0.5)(x)

```

```

    outputs = layers.Dense(num_classes, activation='sigmoid')(x)

```

```

    model = models.Model(inputs, outputs)

```

```

    return model

```

```

model = build_model(input_shape=(256, 256, 3), num_classes=100,
use_se=True)

```

```

model.compile(

```

```

    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),

```

```

    loss=unified_focal_loss,

```

```

    metrics=[

```

```

        'binary_accuracy',

```

```

        metrics.F1Score(average='macro', name='f1_macro'),

```

```

        metrics.F1Score(average='micro', name='f1_micro')

```

```

    ]

```

```

)

```

ДОДАТОК Б ГРАФІЧНІ МАТЕРІАЛИ

Багатоміткова класифікація стилізованих зображень за допомогою глибоких нейронних мереж

Виконав:

Студент IV курсу Тюкалов Н. С. групи КА-13.

Керівник:

Асистент Древаль М.М.

Актуальність

2

Комп'ютерний зір (CV) є одним з найбільших напрямків машинного навчання, який знайшов застосування в різноманітності індустрій та сфер.

Незалежно від сфери, головним типом задачі комп'ютерного зору є класифікація та її варіації. Дуже реальною проблемою є необхідність виявити наявність великої кількості класів або об'єктів одночасно. Ця задача називається **багатомітковою класифікацією**.

Багатоміткова класифікація тегів зображень в аніме-стилістиці є проблемою з унікальними труднощами та викликами. Модель з подібною специфікою може бути інтегрована в image-board платформи для сортування, пошуку та модерації контенту.

Об'єкт дослідження

3

Об'єкт дослідження – великий незбалансований датасет стилізованих аніме-зображень, створений на основі даних з відкритого ресурсу Danbooru, що характеризується детальною комплексною системою тегів.

Предмет дослідження

4

Предмет дослідження – архітектури глибоких нейронних мереж, зокрема Attention-механізми, методи багатоміткової класифікації, а також стратегії роботи з комплексними та проблемними даними.

Мета дослідження

5

Мета дослідження – адекватна модель багатоміткової класифікації стилізованих аніме-зображень.

Постановка задачі

6

Постановка задачі – спроектувати та натренувати модель багатоміткової класифікації аніме-зображень з використанням сучасних архітектур нейронних мереж, яка здатна адекватно класифікувати найпопулярніші теги.

Вхідні дані

7

В якості набору даних в даній роботі використовується датасет на основі архиву аніме-зображень Danbooru.

Danbooru це image-board платформа яка існує з 2006 року, і спеціалізується на аніме контенті. За оцінкою цей ресурс має більше 7 мільйонів зображень.

Для категоризації зображень використовується комплексна система тегів. Кількість тегів оцінити складно, але кількість “справжніх” тегів вимірюється щонайменше в тисячах.

Безпосередньо для моделі використовувався піднабір з 25к зображень та топ-100 тегів, в зв'язку з обмеженими обчислювальними ресурсами.

Структура даних

8

Кожне зображення детально описано та розмічено з використанням тегів, які розділені на 5 головних категорії:

- художник;
- авторські права;
- зображений персонаж;
- загальні описові теги (наприклад “1girl” та “solo”);
- мета теги (наприклад “highres”, якщо зображення високої якості).

Приклади оброблених даних з датасету

9

Tags: 1girl, black_hair, collarbone, green_eyes, long_hair, solo



Tags: black_hair, blonde_hair, blue_eyes, gloves, jewelry, multiple_boys



Проблемність даних

10

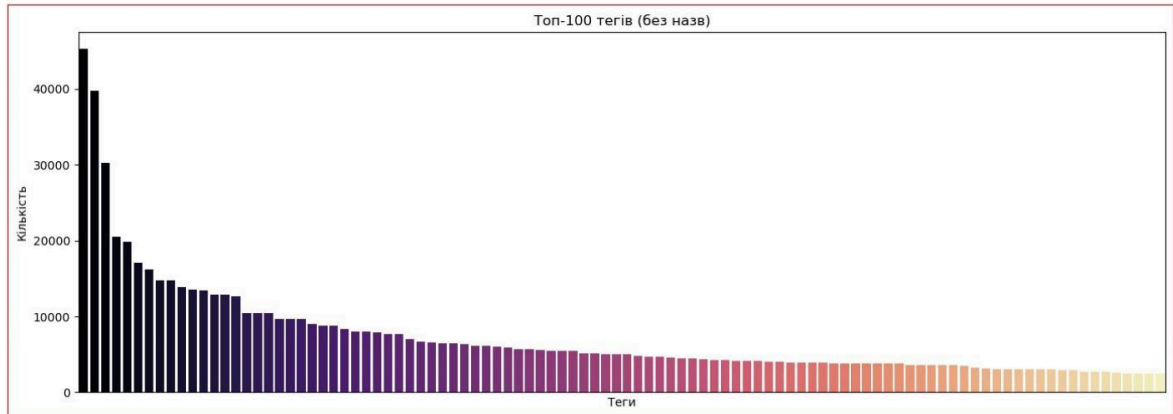
Дані характеризуються наступними проблемами:

- 1) перетини тегів;
- 2) абстрактність тегів;
- 3) псевдоієрархічна структура;
- 4) критичний дисбаланс класів;
- 5) відсутність тегів.

Також існує ряд проблем, який пов'язаний з самою стилістикою, наприклад невиражений гендерний диморфізм.

Ілюстрація дисбалансу популярності тегів

11



Архітектура моделі

12

1. Вхідний шар (Input): 256x256, 3 канали.
2. Початковий CNN блок: згортковий шар, batch normalization, ReLu.
3. MaxPooling2D.
4. ResNet: Чотири ResNet блока з послідовним збільшенням кількості фільтрів. В кожному ResNet блоці також присутній SE блок.
5. GlobalAveragePooling2D
6. Dropout
7. Вихідний шар: Dense на 100 нейронів, Sigmoid.

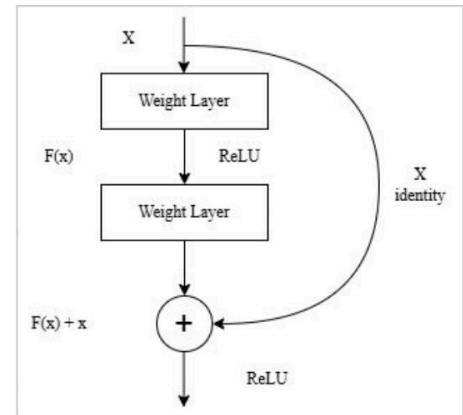
ResNet

13

ResNet це тип архітектури NN.

ResNet складається з блоків залишку (Residual block).

Основна ідея ResNet архітектури в тому, щоб замість тренування вирішення поставленої задачі напряму, мережа вчиться “доповненню до задачі”, що є простішим завданням.

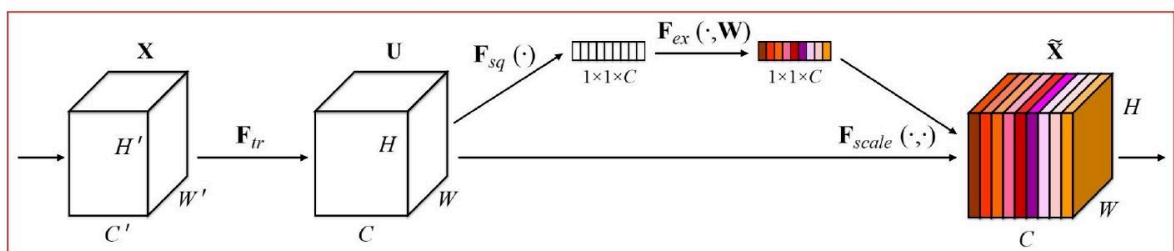


Squeeze-and-Excitation

14

Squeeze-and-Excitation (SE) блок є простим механізмом уваги, який динамічно зважує важливість каналів у згорткових шарах.

SE складається з двох етапів: **Squeeze** (видобування ознак з GAP) та **Excitation** (ранжування ознак Dense шарами та Sigmoid виходом).



Focal Loss

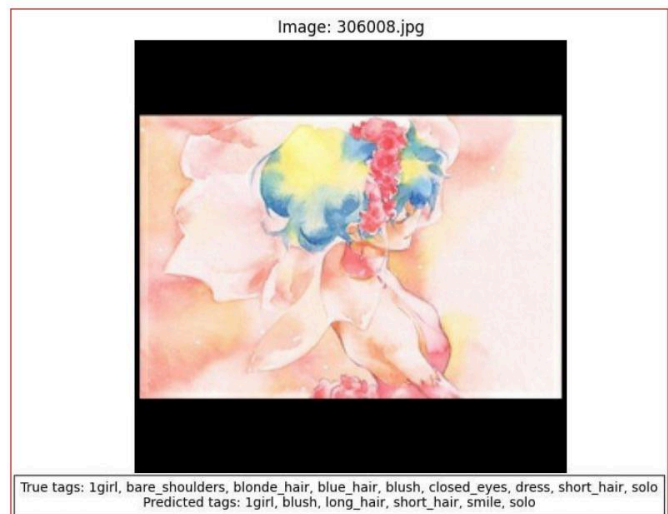
15

Focal Loss – це функція втрат, яка фокусується на складних прикладах зменшенням ваги для прикладів, які класифікуються успішно, та збільшенням ваги для прикладів, які класифікуються менш успішно.

$$L_{Focal}(y, \hat{y}) = -\alpha_t (1 - \hat{p}_t)^\gamma \log(\hat{p}_t)$$

Приклади результату роботи

16



Вихідні метрики

17

Для порівняння взята незалежна модель, яка натренована, по суті, на цьому ж датасеті.

Модель	Кількість зображень	Кількість тегів	Кількість параметрів	F1-micro	F1-macro	Розмір
Власна	25к	100	17м	0.3778	0.1865	200 мб
Camie Tagger	7м	70к	214м	0.576	0.204	850 мб

Висновки

18

У рамках даної дипломної роботи було успішно досліджено проблему багатоміткової класифікації стилізованих зображень із застосуванням глибоких нейронних мереж.

Була спроектована, побудована та натренована модель багатоміткової класифікації на основі ResNet-архітектури, посиленої блоками Squeeze-and-Excitation.

Було досягнуто адекватних метрик оцінки моделі.

Подальші дослідження

19

В рамках подальшого дослідження пропонується:

- 1) використання більшої кількості даних;
- 2) розширення кількості тегів;
- 3) пошук оптимального методу обробки даних;
- 4) використати новітні архітектури, таких як ConvNext;
- 5) застосування візуальних трансформерів;
- 6) імплементація оптимізатора GSAM;
- 7) продовжити пошук оптимальної функції втрат.

Використані технології та інструменти

20

1. Середовище Google Collab .
2. Фреймворк TensorFlow.
3. Стандартні Python бібліотеки.

Дякую за увагу!