

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»  
Факультет прикладної математики  
Кафедра прикладної математики**

«До захисту допущено»  
Завідувач кафедри  
\_\_\_\_\_ Олег Чертов  
«\_\_» \_\_\_\_\_ 2024

**Дипломна робота  
на здобуття ступеня бакалавра  
за освітньо-професійною програмою  
«Наука про дані та математичне моделювання»  
спеціальності 113 «Прикладна математика»  
на тему: «Математичне та програмне забезпечення для автоматичного  
тегування зображень»**

Виконав:  
студент IV курсу, групи КМ-01  
Скороденко Дмитро Олександрович

Керівник:  
Доцент, к. т. н., доцент кафедри ПМА  
Сирота Сергій Вікторович

Консультант з нормоконтролю:  
Старший викладач  
Мальчиков Володимир Вікторович

Рецензент:  
Доцент, канд. техн. наук, доцент кафедри ПЗКС  
Онай Микола Володимирович

Засвідчую, що в цій дипломній роботі  
немає запозичень із праць інших авторів  
без відповідних посилань  
Скороденко Дмитро Олександрович

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»  
Факультет прикладної математики  
Кафедра прикладної математики**

Рівень вищої освіти - перший (бакалаврський)

Спеціальність - 113 «Прикладна математика»

Освітньо-професійна програма «Наука про дані та математичне моделювання»

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_ Олег Чертов

«\_\_\_» \_\_\_\_\_ 2024

**ЗАВДАННЯ**

**на дипломну роботу студента**

Скороденку Дмитру Олександровичу

1. Тема роботи «Математичне та програмне забезпечення для автоматичного тегування зображень», керівник роботи Сирота Сергій Вікторович, канд. техн. наук, доцент, затверджені наказом по університету від «30» травня 2024 р. №2205-с .
2. Термін подання студентом роботи «10» червня 2024 р.
3. Вихідні дані до роботи: спроектована модель для маркування зображень повинна забезпечити якість опису зображення краще або на рівні із існуючими рішеннями згідно із тестовими метриками.
4. Зміст роботи: виконати огляд існуючих рішень задачі маркування зображення, провести моделювання на основі аналізу існуючих рішень, імплементація моделі, тренування / тестування моделі, аналіз ефективності компонентів, аналіз ефективності моделі у порівнянні із існуючими рішеннями.
5. Перелік обов'язкового ілюстративного матеріалу: діаграма архітектури композитної моделі, розподіл лейблів у тренувальному/тестовому датасеті, числові

характеристики датасету, графіки процесу тренування для кожної із компонент моделі, таблиця для порівняння ефективності компонентів моделі, таблиця для порівняння ефективності маркування у порівнянні з існуючими рішеннями, ілюстративні приклади роботи моделі.

6. Дата видачі завдання «05» лютого 2024 р.

**Календарний план**

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення та збір літератури за темою "маркування зображень"	25.01.2024	+
2.	Проведення аналізу особливостей існуючих систем маркування зображень	15.02.2024	+
3.	Вибір та підготовка датасету	25.02.2024	+
4.	Проектування моделі маркування зображень	10.03.2024	+
5.	Проведення тестового тренування моделі та відладка програми	25.03.2024	+
6.	Проведення валідації результатів тестового тренування	28.03.2024	+
7.	Проведення тренування фінальної версії моделі	14.04.2024	+

8.	Проведення порівняльного аналізу компонентів моделі	15.04.2024	+
9.	Проведення порівняння ефективності з існуючими рішеннями	15.04.2024	+
10.	Підготовка першої версії дипломної роботи	08.05.2024	+
11.	Оформлення пояснівальної записки	20.05.2024	+

Студент \_\_\_\_\_

Дмитро СКОРОДЕНКО

Керівник роботи \_\_\_\_\_

Сергій СИРОТА

## АНОТАЦІЯ

Дипломну роботу виконано на 50 аркушах, вона містить 3 додатки та перелік посилань на використані джерела з 23 найменувань. У роботі наведено 11 рисунків та 4 таблиці.

**Актуальність теми:** В сучасному світі технології розвиваються надзвичайно швидко. Швидкість цього розвитку можна виміряти об'ємами даних, яким оперують люди. Так для 2000-х років було цілком достатньо мати дискети із максимальним вмістом до 10-15 МБ. Станом на 2024 рік, існують різні накопичувачі від 16 ГБ до декількох десятків ТБ. Чому це важливо? Для того щоб переносити велику кількість даних потрібно, щоб генерувалось ще більше даних. І це дійсно так, якщо наприклад розглянути сучасні хостинги зображень, бази даних медзакладів, бази даних супутникових знімків і тд. то всюди ми побачимо уже від сотень тисяч до сотень мільйонів зображень, при чому швидкість появи нових зображень стрімко зростає. Основними причинами такого росту є: збільшення кількості людей та стрімка цифровізація більшості аспектів людської життєдіяльності. Такий вибуховий ріст у швидкості появи нових зображень створює проблему структуризації зображень. Для вирішення цієї проблеми можна присвоїти кожному зображеню лейблі, які загально описують його вміст. Без таких лейблів будь яка структура, яка містить у собі велику кількість зображень перетвориться у звалище. Саме тому важливо мати швидкий та якісний метод для автоматичного маркування зображень.

**Мета дослідження:** Метою даної роботи є розробка ПЗ для маркування зображень (шпалерів робочого столу) для покращення системи категоріального пошуку зображень (шпалерів робочого столу).

**Завдання дослідження:** Створення моделі, яка виконуватиме маркування шпалерів робочого столу на основі двох модальностей даних: зображення та шумних тегів.

Для досягнення цієї мети було виконано наступні завдання:

- Проведено аналіз існуючих рішень
- Виконано моделювання
- Проведено тренування нейронної мережі
- Проведено аналіз ефективності компонентів моделі
- Проведено порівняльний аналіз якості і повноти опису розглянутої моделі відносно існуючих рішень на основі тестових метрик
- Проведено аналіз ілюстративних прикладів роботи моделі

**Об'єкт дослідження:** Об'єктом дослідження є маркування зображень, та методи покращення маркування зображення. Для порівняння ефективності маркування серед існуючих рішень було обрано моделі, для яких обраховані тестові метрики для того ж датасету, який обрано в даній роботі. До множини існуючих рішень належать: SR-CNN-RNN [1], Resnet-SRN [2], MS-CMA [3], Query2Label [4], Resnet-CPSP [5] та ін.

**Предмет дослідження:** Предметом дослідження є множина шпалерів робочого столу в якості основної модальності даних, та додаткова інформація (надані людьми шумні теги) в якості додаткової.

#### **Методи дослідження:**

- \* Теорія системного аналізу
- \* Проектування систем Data Science / Deep learning
- \* Проектування інформаційних систем
- \* Обробка та аналіз зображень та тегів на основі методів глибинного навчання
- \* Теорія алгоритмів
- \* Аналіз даних та математична статистика

**Кінцевий результат:** Кінцевим результатом даної роботи є математичне та програмне забезпечення, архітектура моделі, вагові коефіцієнти натренованої моделі та код програмного забезпечення, в якому реалізовано дану роботу.

**Ключові слова:** Маркування зображень, глибинне навчання, нейронні мережі, згорткові нейронні мережі, багатошарові персепtronи, композиція нейронних мереж.

## ABSTRACT

The thesis presented in 50 pages. It contains 3 appendixes and bibliography of 23 references, 11 figures and 4 tables are given in the thesis.

**Topic relevance.** In modern world technologies are progressing with very high speed. The speed of this progress can be measured by volume of stored data which people operate with. In this regard for the 2000s, it was quite enough to have diskettes with the maximum volume of up to 10-15 MB. As of 2024, there are various drives from 16GB to several tens of TB. Why is this important? In order to transfer large amounts of data, even more data should be generated. And it really is, let's for example consider modern image hostings, databases of medical institutions, databases of satellite images, etc. then everywhere we will see from hundreds of thousands to hundreds of millions of images, and the rate of appearance of new images is rapidly increasing. The main reasons for this growth are: an increase in the number of people globally and the rapid digitalization of every aspect of our everyday lives. Such explosive growth in speed the appearance of new images creates a problem of image structuring. To solve this problem, you can assign each image with labels that generally describe its content. Without such labels, any structure that contains a large number of images will be like a waste disposal site, where you can't find anything. That's why it's important to have a fast and quality method for automatic image labeling.

**Research goal:** The purpose of this work is to create deep learning model for automatic image labelling, specifically wallpapers, to improve categorical search using output labels.

**Research objectives:** Creating system, which will label images (wallpapers), given data of two modalities: image and associated noisy tags.

To reach said objective the following tasks were completed:

- Conducted analysis of existing solutions
- Modeled image labeling system
- Conducted model training

- Conducted analysis of performance gain from each component of composite system
- Conducted comparative analysis of model's labeling effectiveness with existing solutions
- Conducted analysis of illustrative examples of system's result

**Research object:** The object of research is image labeling and methods to improve it. To conduct comparative analysis to existing solution, only models which are trained/tested on specific dataset were considered as 'existing solutions'. To 'existing solutions' belong following models: SR-CNN-RNN [1], Resnet-SRN [2], MS-CMA [3], Query2Label [4], Resnet-CPSP [5] etc.

**Research subject:** The subject of this work is subset of images which are called 'wallpapers'. These are the images that you see as background image when you use your computer. The main modality of data is image. Human provided tags would be additional modality.

### **Research methods:**

- \* System analysis
- \* Data Science / Deep learning
- \* Projecting of informational systems
- \* Processing and analysis of images and tags using methods of deep learning
- \* Theory of algorithms
- \* Data analysis and mathematical statistics

**End result of research:** The end result of research is mathematical and software implementation, architecture of model, weights of trained models and source code of software which implements this work.

**Keywords:** Image labeling, deep learning, neural networks, convolutional neural networks, multilayer perceptrons, composition of neural networks.

## ЗМІСТ

Перелік умовних позначень, скорочень і термінів .....	3
Вступ .....	4
1 Постановка задачі .....	6
2 Огляд існуючих рішень задачі маркування зображення .....	7
2.1 Базове рішення .....	7
2.2 Додаткова модальності даних .....	8
2.3 Кількість лейблів .....	10
2.4 Висновки до розділу .....	12
3 Моделювання .....	13
3.1 Модель VCN.....	13
3.2 Модель MLP .....	14
3.3 Модель LP .....	15
3.4 Модель LQP .....	16
3.5 Процес тренування.....	17
3.5.1 Цільові функції .....	17
3.5.2 Тренування VCN.....	18
3.5.3 Тренування MLP .....	19
3.5.4 Тренування LP .....	19
3.5.5 Тренування LQP .....	19
3.6 Процес тестування .....	20
3.6.1 Тестування класифікаційних моделей (VCNN, MLP, LP) .....	20
3.6.2 Тестування композитної моделі.....	20
3.7 Висновки до розділу .....	21
4 Експерименти .....	22
4.1 Датасет .....	22
4.1.1 Множина цільових класів .....	24
4.1.2 Числові характеристики датасету .....	25
4.2 Тестові метрики .....	26

	2
4.3 Тренування.....	28
4.3.1 Параметри навчання .....	28
4.3.2 Процес навчання .....	29
4.4 Висновки до розділу .....	30
5 Аналіз результатів .....	31
5.1 Аналіз компонентів моделі .....	31
5.2 Аналіз впливу додаткової інформації .....	32
5.3 Порівнення з існуючими рішеннями.....	34
5.4 Демонстративні приклади.....	36
5.5 Висновки до розділу .....	37
Висновки .....	38
Перелік посилань .....	39
Додаток А Лістинг програм .....	42
Додаток Б Додаткові приклади.....	67
Додаток В Ілюстративний матеріал .....	68

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CNN - Згорткова нейронна мережа (Convolutional Neural Network)

DNN - Глибинна нейронна мережа (Deep Neural Network)

RNN - Рекурсивна нейронна мережа (Recursive Neural Network)

ViT - Візуальні трансформери (Visual Transformers)

Задача класифікації - це задача, яка вирішує проблему приналежності чогось виключно до одного класу із довільного набору класів.

Задача маркування - це задача, яка вирішує проблему приналежності чогось до декількох класів із довільного набору класів.

Лейбл (Label) - синонім слова ground truth для класифікації

Модель - нейронна мережа

Тег (Tag) - шумна інформація надана користувачем у формі тексту

## ВСТУП

Задача класифікації - це одна із основних задач в аналізі зображень, вона полягає у присвоєнні кожному зображенню один із класів. Таким чином дане формулювання накладає обмеження - зображення містить тільки один об'єкт. Поява DNN [6] та її подальший розвитком у CNN [7, 8] разом із створенням великих датасетів як-от ImageNet [9] дало змогу вирішувати задачу класифікації зображень значно швидше і якісніше ніж люди.

Зрозуміло, що зображення - це той тип даних, який у абсолютної більшості випадків містить більше одного об'єкта, і відповідно більше одного класу для класифікації. Для поглиблення опису існує задача маркування зображень (image labeling). На відміну від класифікації, вона полягає у маркуванні зображення більше ніж одним класом. Таким чином повнота опису зображення кратно зростає у порівнянні із звичайною класифікацією, однак привносить декілька складних завдань.

1) Наявність декількох класів у одного зображення створює можливість описувати значно ширший спектр візуальної інформації: різні об'єкти, стилі, дії, і тд. Це створює потребу у розгляді додаткових джерел інформації, так як одного лише зображення вже недостатньо. Поява великих хостингів зображень таких як Imgur, Flickr, та ін., де користувачі можуть як завантажувати різноманітні зображення, так і додавати до них описову інформацію у вигляді тегів / анотацій, дала змогу створити досить різноманітні датасети: ImageNet [9], MS-COCO [10], NUS-WIDE [11], та ін. Також існують і інші види датасетів, наприклад: рентгенівські знімки та додаткова інформація (інші аналізи пацієнта, історія хвороб ...), супутникові знімки та додаткова інформація у вигляді метаданих, геолокацій тощо. Таким чином задача якісного маркування зображення вже охоплює значно більший спектр даних ніж просто зображення.

2) Маркування зображень передбачає динамічну к-сть промаркованих класів, так для опису зображенням із широким спектром понять необхідно 5-6 класів, для зображення із простим вмістом - 2-3 класи.

3) Маркування зображень потребує оцінки якості проведеного маркування. Оскільки будь який датасет буде містити в собі дизбаланс класів в тій чи іншій мірі, важливо оцінювати маркування із урахуванням цього.

4) Маркування зображень значно складніша задача ніж класифікація і відповідно зростає складність моделей. З однієї сторони складніша модель потенційно здатна покращити якість маркування, з іншої - може сильно збільшити як час на виконання маркування, так і час затрачений на тренування системи. До складних систем належать ті, які використовують трансформери та/або мають велику к-сть параметрів. Отже, важливо обрати певний баланс відносно складності моделі.

Все це робить задачу маркування зображення досить складною.

## 1 ПОСТАНОВКА ЗАДАЧІ

Метою даної дипломної роботи є розробка математичного та програмного забезпечення для маркування зображень (шпалерів робочого столу) для покращення системи категоріального пошуку зображень (шпалерів робочого столу).

До множини лейблів (цільових класів) віднесено 81 поняття із датасету NUS-WIDE [11].

До множини тегів (додаткових даних) віднесено 1000 найбільш частих тегів із  $\approx 5000$  тегів у датасеті NUS-WIDE [11].

Для досягнення цієї мети було виконано наступні завдання:

- Проведено аналіз існуючих рішень
- Виконано моделювання
- Проведено тренування нейронної мережі
- Проведено аналіз ефективності компонентів моделі
- Проведено аналіз впливу максимальної кількості тегів на якість опису зображення
- Проведено порівняльний аналіз якості і повноти опису розглянутої моделі відносно існуючих рішень на основі тестових метрик
- Проведено аналіз ілюстративних прикладі роботи моделі

## 2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ЗАДАЧІ МАРКУВАННЯ ЗОБРАЖЕННЯ

Розглянемо ключові аспекти задачі маркування зображення.

### 2.1 Базове рішення

Базовим рішенням в задачі маркування зображення є аналіз основної модальності даних - зображення. У абсолютній більшості існуючих робіт використовується CNN (Convolutional Neural Network). Застосовуються різні архітектури даної моделі ResNet [12], AlexNet [13], GoogleNet [14], ResNext [15].

В якості базового рішення також можна використовувати ViT [16]. Дано модель використовує трансформери, і аналізує зображення по частинам (patches).

Спільним між CNN та ViT є те, зазвичай їх рідко тренують з нуля, так як для цього необхідно багато ресурсів. Саме тому використовують вже натреновані (pretrained) моделі на великому датасеті, здебільшого ImageNet [9]. Для адаптації моделі до обраного контексту така модель дотреновується (fine tune), замінюючи існуючий класифікатор. Це працює завдяки тому, що всі архітектури сучасних CNN та ViT моделей містять десятки мільйонів параметрів, даючи широку репрезентацію зображень. Для CNN - це ієрархічне представлення: перші шари репрезентують базові особливості, а останні - більш специфічні особливості. Для ViT - це представлення, яке будується на основі взаємозв'язків між різними частинками зображення за допомогою трансформерів. Такі репрезентації зображень дозволяють адаптувати модель під різні задачі після проведення підгонки (finetune).

Обидва підходи мають свої переваги та недоліки, і вибір між ними залежить від конкретної задачі, доступних даних та обчислювальних ресурсів, так як ViT, попри всі свої переваги потребує в середньому більше даних та ресурсів.

## 2.2 Додаткова модальність даних

Більш нові роботи також розглядають додаткові джерела інформації для підвищення якості та повноти маркування зображень. Існує два основних підходи:

а) *Аналіз додаткової інформації.* Даний підхід аналізує додаткову до зображення інформацію. Це може бути як текстова інформація (теги / анотації) [17, 1], так і метадані зображення [18, 19]. Очевидним недоліком даного методу є потреба у цій додатковій інформації, яку можуть мати не всі зображення, а відсутність даної інформації знижує точність результуючого маркування.

б) *Аналіз цільових класів.*

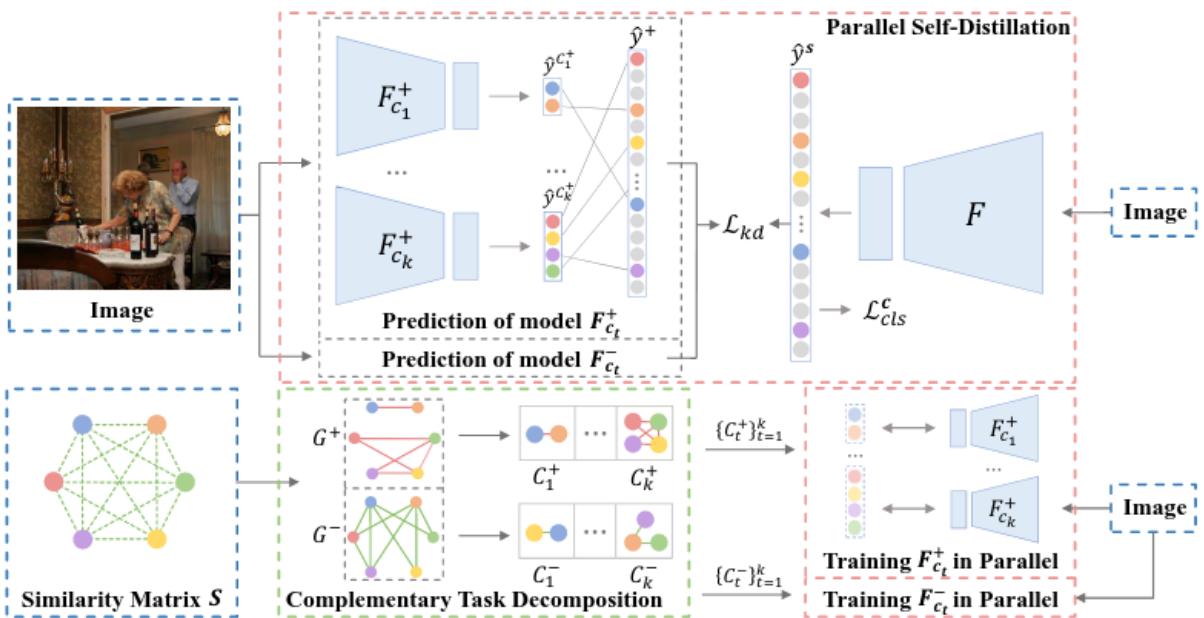


Рисунок 2.1 – Приклад архітектури моделі, яка використовує графове представлення цільових класів Resnet-CPSP [5].

На відміну від загальної інтерпретації класів для задачі маркування (коли кожен клас - це незалежна сутність), даний підхід аналізує зв'язки між цільовими класами, створюючи нову модальність на основі набору цільових класів [2]. Більш новим та узагальненим підходом є технологія word2vec (або аналогічне рішення), яка дозволяє впорядкувати слова у певному векторному просторі таким чином, що їх просторове

значення має прямий зв'язок із їх семантичним значенням. Наприклад модель Resnet-CPSD [5] використовує для аналізу класів граф, в якому вказуються класи (поняття) які часто знаходяться на одному зображенні (co-occurrence; наприклад: риба, вода) та класи які рідко знаходяться на одному зображенні (dis-occurrence; наприклад: риба, пустеля). Перевагою даного підходу є те, що йому не потрібні ніякі нові дані крім зображень та відповідних їм цільових класів, а нова модальності, яка репрезентує зв'язок між класами створюється під час тренування системи. Основним недоліком таких систем полягає у високій складності, і як наслідок - довше по часу тренування / розпізнавання.

## 2.3 Кількість лейблів

Результатом роботи класифікаційної моделі є вектор ймовірностей, який репрезентує приналежність до класів. Для задачі класифікації вибір результату на основі цього вектора очевидний - клас із найбільшою ймовірністю, однак для задачі маркування все складніше.

Image				
Truth	clouds grass house sky	leaf plants sky	animal grass	person
Top 5 pred	clouds grass house road sky	clouds grass leaf plants sky	animal grass horses plants sky	animal clouds person road sky
Model pred	clouds grass sky	plants sky	animal grass horses	person

Рисунок 2.2 – Приклад адаптивної кількості лейблів, на основі роботи моделі на датасеті NUS-WIDE. 'Truth' - правдиве маркування, 'Top 5 pred' - ілюстрація вибору top  $k$ , при  $k = 5$ , 'Model pred' - приклад роботи моделі

Більшість наведених вище робоїт розглядають задачу вибору к-сті лейблів як найкращі  $k$  (top  $k$ ) маркувань (Розділ 4.2), де  $k$  - наперед задана константа. Очевидно, що такий вибір к-сті класів не є оптимальним, так як більш змістовні зображення будуть містити менше описової інформації і навпаки - менш змістовні будуть містити лишню інформацію, яка до того ж може не мати нічого спільногого із цим

зображенням (Рис. 2.2)

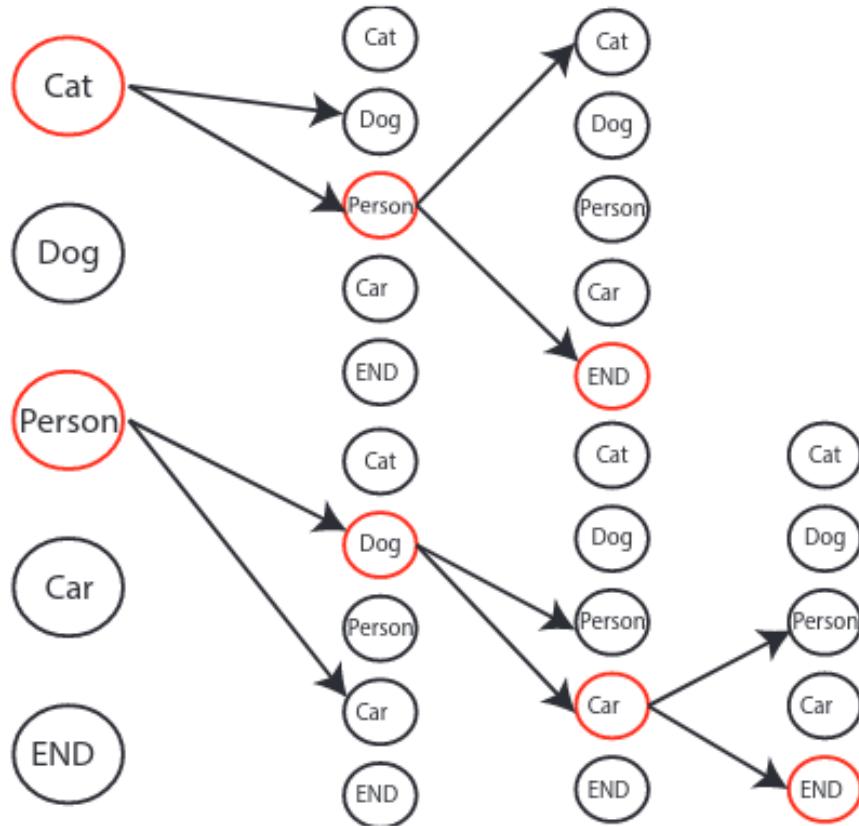


Рисунок 2.3 – Приклад застосування алгоритму "beam search", для пошуку найбільш оптимального маркування зображення на основі найвищої ймовірності, який реалізовано у системі CNN-RNN [20].

Один із підходів як-от CNN-RNN [20], використовує RNN для аналізу візуальних даних (visual features) та автоматично виконує як задачу маркування, так і задачу динамічного вибору кількості лейблів, однак в силу особливості RNN є певні обмеження накладенні на порядок кодування класів. При чому важливо відмітити, що вибір динамічної кількості лейблів за допомогою RNN має суттєвий недолік - залежність від балансу цільових класів у даних.

Найновіші моделі [4, 5, 3], які розглядають зв'язок між цільовими класами, обирають к-сть цільових класів на основі порогового значення threshold (Розділ 4.2). Даний підхід є ефективним рішенням для вибору кількості лейблів, однак він релевантний тільки для цього типу моделей, так як вектор ймовірностей, який

отримується на виході даної моделі досить сильно дискретизований, тобто для позитивного лейблу, який маркується 1 ймовірність буде  $\approx 0.7 - 0.9$ , а для негативного, тобто 0 ймовірність  $\approx 0.1 - 0.3$ .

## 2.4 Висновки до розділу

Було проведено огляд існуючих рішень задачі маркування зображення, відносно ключових аспектів рішення даної задачі. Варто зазначити, що розглянуті рішення концентруються на загальному рішенні задачі маркування зображень, і, в абсолютній більшості, не використовують додаткові дані.

В якості базового рішення між ViT та CNN варто використовувати CNN, так як вони є більш ефективними з точки зору обчислень та інтерпретуються краще.

Для покращення якості маркування зображень буде залучено додаткову інформацію. Так як дана робота націлена на маркування певної підмножини зображень, а саме: шпалерів робочого столу, - то для покращення маркування будуть використовуватись надані користувачами теги. Варто зазначити, що множина тегів має бути значно ширшою ніж множина лейблів (цільових класів) у співвідношенні приблизно 10 до 1. На відміну від аналізу цільових класів, даний підхід є значно простішим та досить ефективним, хоч і потребує наявності додаткових тегів.

Розглянуті рішення націлені на розв'язок загальної задачі маркування зображення, тому в більшості випадків використовується метод 'top 3'. Даний підхід цілком задовільний для обрахування тестових метрик, однак є незадовільним для прикладного застосування (так як зображення будуть містити завжди 3 лейбли). Інший метод розглядає порогове значення, яке підходить для вибору динамічної кількості лейблів, однак містить значні недоліки у прикладному застосуванні. Таким чином доцільно використовувати метод 'top k'.

### 3 МОДЕЛЮВАННЯ

На основі проведного аналізу альтернатив, дана робота пропонує розглянути модель, яка розглядає дві модальності даних: зображення та текстові теги.

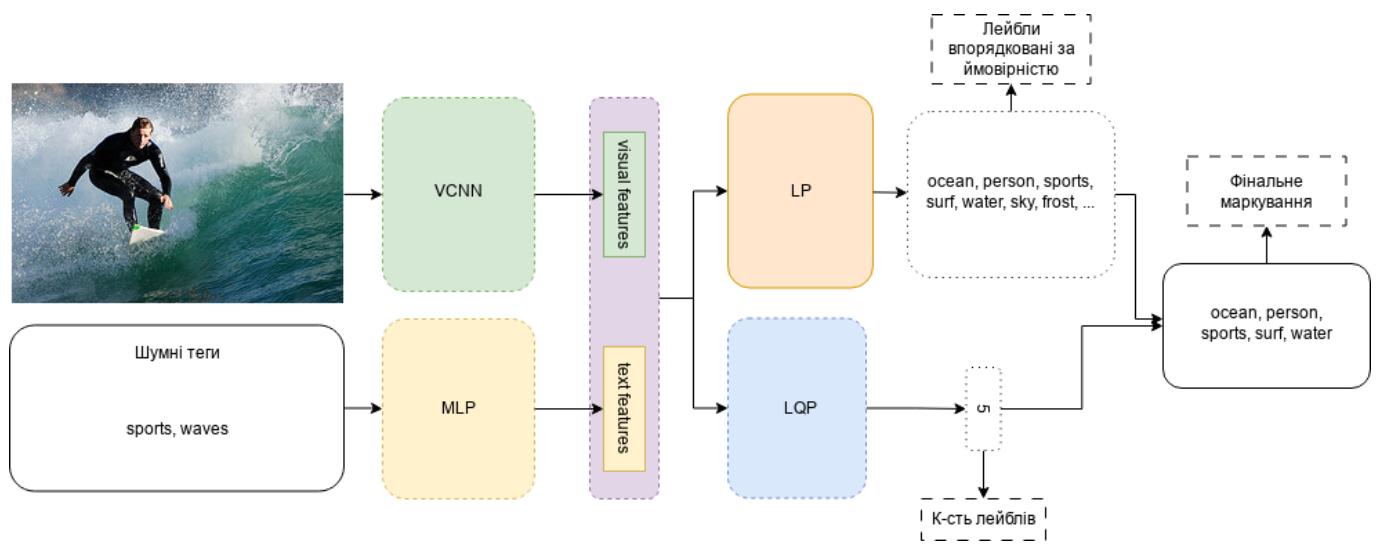


Рисунок 3.1 – Структура композитної нейронної мережі

Структура даного рішення складається із чотирьох компонентів (Рис. 3.1).

#### 3.1 Модель VCNN

Модель VCNN (Рис. 3.2) призначена для вивчення особливостей (features) із зображення.

Отримує на вхід пікселі зображення  $I$ , у формі матриці розмірності  $(B, C, W, H)$ , де  $B$  - к-сть зображень у групі для тегування,  $C$  - к-сть каналів у зображеннях зазвичай 1 або 3, Grey або RGB відповідно,  $W, H$  - розмірність зображень.

За базове рішення використовується ResNext101\_32x8d [15] (сучасна версія resnet), так як у порівнянні із широко застосуваною моделлю Resnet [12] вона

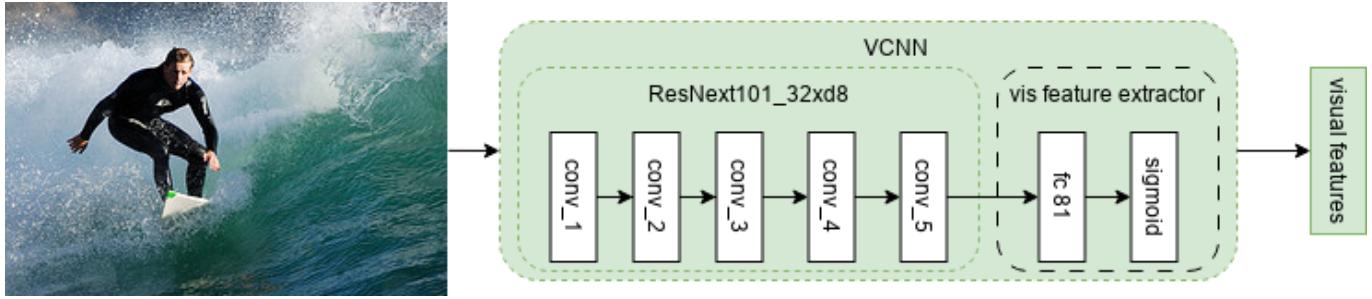


Рисунок 3.2 – Архітектура моделі VCNN

краще описує зображення, та легше дотреновується при використанні натренованої на ImageNet [9] версії.

На виході даної моделі ми отримуємо вектор вірогідностей  $vf$  (visual feature vector), який вказує вірогідність маркування зображення класом  $j$  на основі візуальної інформації.

### 3.2 Модель MLP

MLP (Рис. 3.3) - аналізує текстові особливості (text features) тегів до зображення.

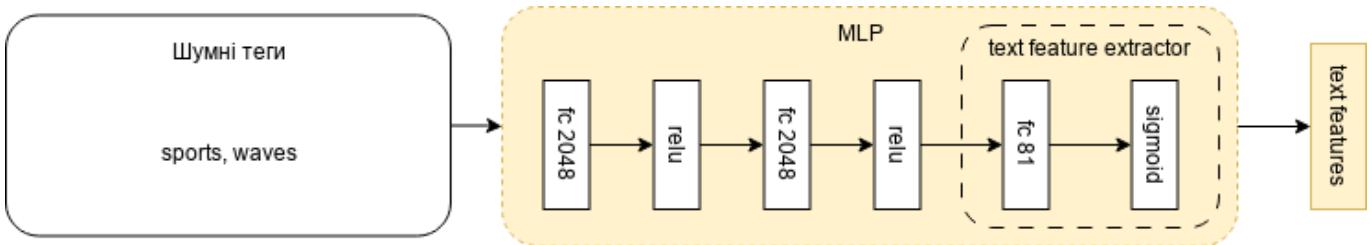


Рисунок 3.3 – Архітектура моделі MLP

Теги до зображення  $i$  репрезентуються як бінарний вектор  $I = [1,0,1,0,\dots,N]$ , де 1 - це наявність тегу, а  $N$  - к-сть тегів.

Головна причина вибору звичайної MLP моделі для аналізу текстової інформації - це те, що вхідна інформація - це шумні теги (наприклад: для фото кота - теги "Канада", "Кіт"). Важливим правилом щодо ефективності цього рішення є

кількісне співвідношення між множиною цільових класів та тегів, воно має бути приблизно 1 до 10.

На виході даної моделі ми отримуємо вектор  $tf$  вірогідностей (text feature vector), який вказує вірогідність маркування зображення класом  $j$  на основі текстової інформації.

### 3.3 Модель LP

LP (Рис. 3.4) - аналізує вектор вірогідності  $v$ , який є композицією векторів  $vf$  та  $tf$ :  $v = [vf, tf]$ .

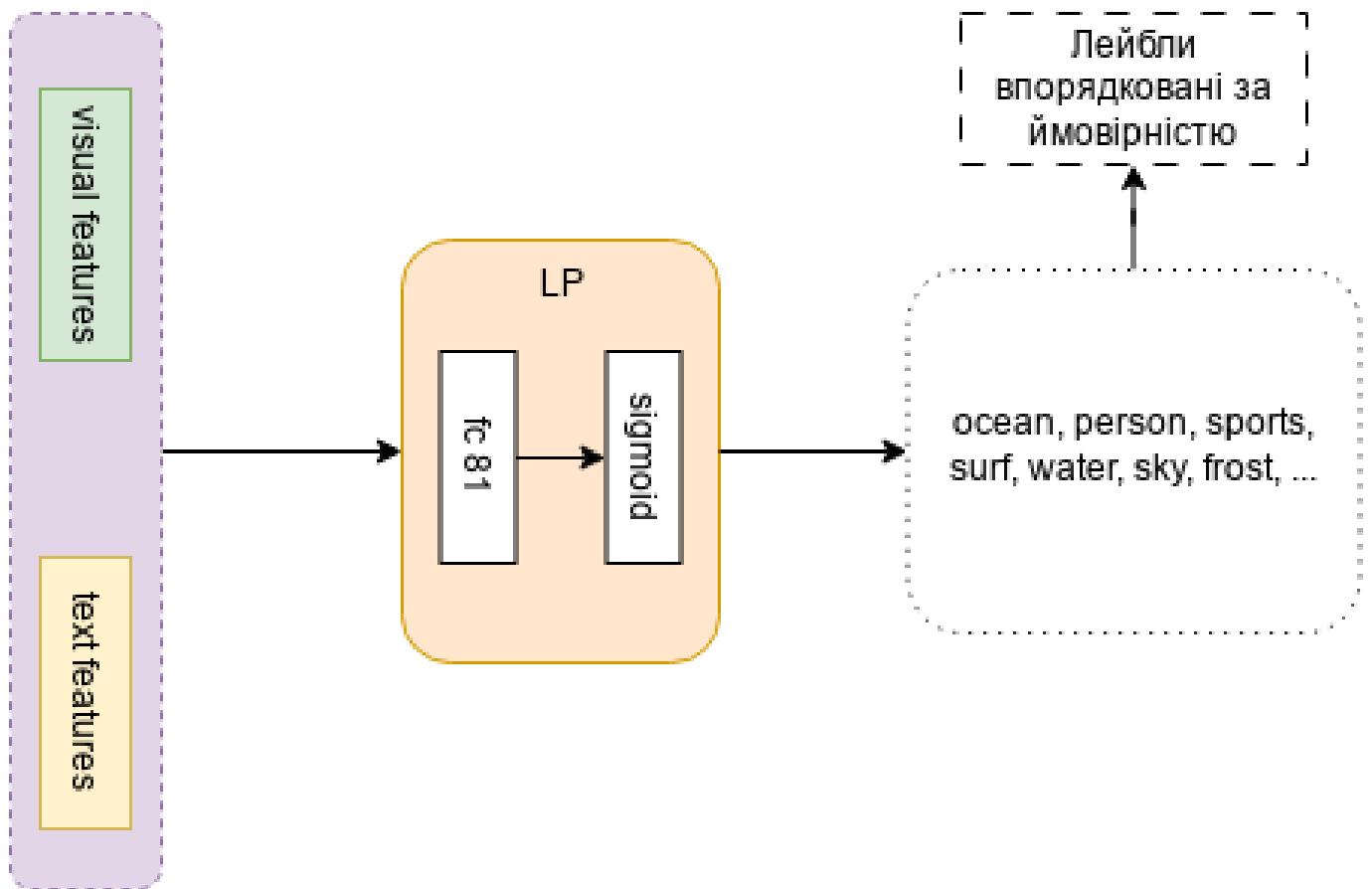


Рисунок 3.4 – Архітектура моделі LP

На виході даної моделі ми отримуємо вектор вірогідностей, який комбінує

інформацію отриману як із візуальної так і з текстової інформації.

### 3.4 Модель LQP

Модель LQP (Рис. 3.5) аналізує кількість лейблів на основі вектору вірогідностей  $v$ , який є композицією векторів  $vf$  та  $tf$ :  $v = [vf, tf]$ .

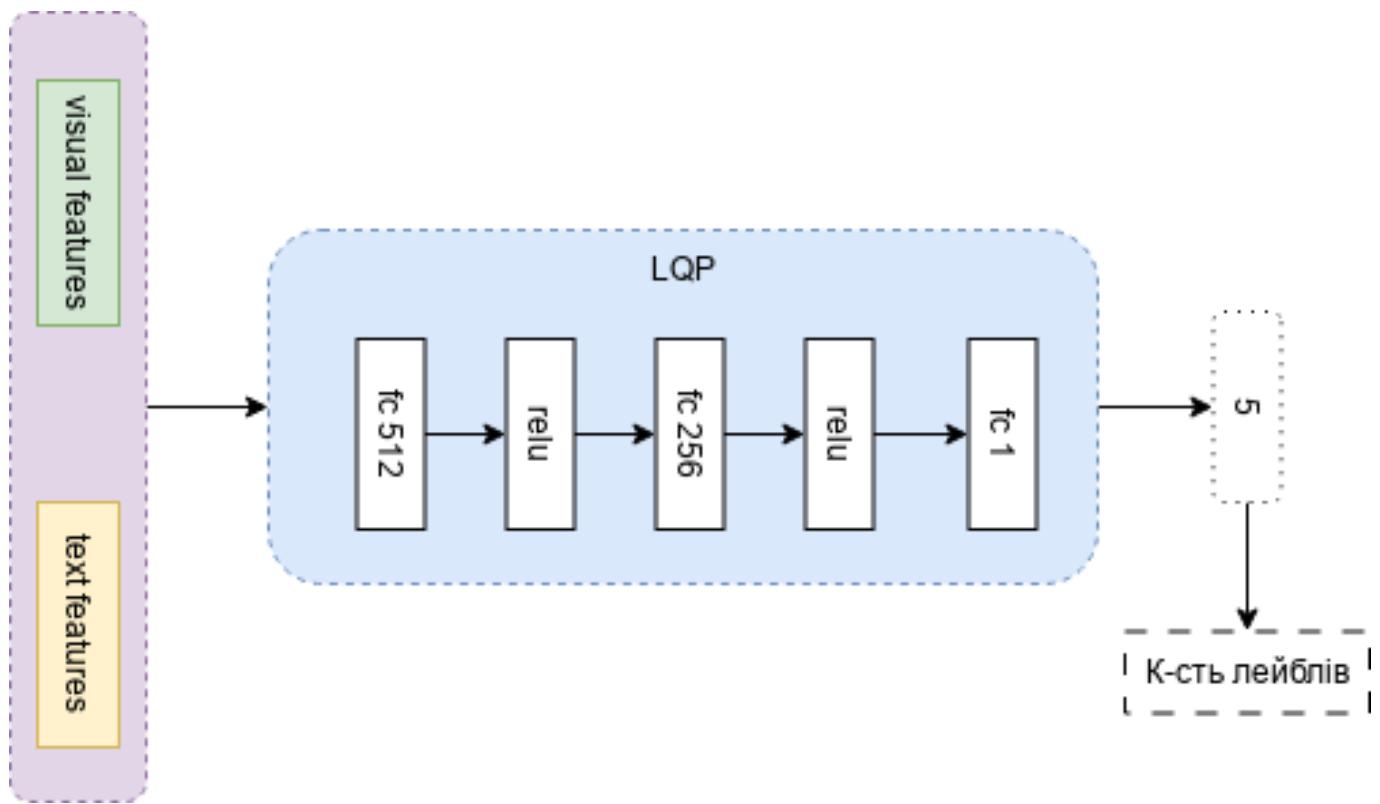


Рисунок 3.5 – Архітектура моделі LQP

Існує два підходи до визначення к-сті за допомогою нейронних мереж: класифікація та регресія. LQP - регресійна модель.

Оскільки регресійні моделі досить швидко перенавчаються (overfitting), то необхідно задіяти регуляризацію. В даній роботі, в якості регуляризатора задіяні Dropout шари [21], із вірогідністю відкидання (dropout rate) 0.5.

На виході даної моделі є число, яке вказує на кількість лейблів у зображенні.

### 3.5 Процес тренування

Модель складається із декількох компонентів, що створює декілька проблемних місць під час тренування: досить багато параметрів, дві різні цільові функції, проблема затухаючого градієнта, - тому тренування відбувається у декілька стадій, у якому кожна із моделей тренується окремо (деякі з них можна тренувати синхронно).

#### 3.5.1 Цільові функції

Для початку варто розглянути цільові функції (функція втрат, loss function). Дані функції є базовим компонентом глибинного навчання.

В даній роботі використовуються дві функції: BCEWithLogitsLoss та MSELoss.

#### BCEWithLogitsLoss

Для тренування класифікаційних моделей (VCNN, MLP, LP) вихідні логіти  $z_{ij}$  для групи (batch) зображень  $I_N$  при  $i = 1\dots N$ ,  $j = 1\dots C$ , де  $N$  - кількість зображень в групі,  $C$  - кількість цільових класів, цільова функція має вигляд:

$$\mathcal{L}_{cls} = \frac{1}{NC} \sum_i^N \sum_j^C y_{ij} \cdot \ln(\sigma(z_{ij})) + (1 - y_{ij}) \cdot \ln(1 - \sigma(z_{ij})) \quad (3.1)$$

, де  $y_{ij} = 1$  якщо зображення  $i$  анатоване класом  $j$ , інакше -  $y_{ij} = 0$ , а  $\sigma(\cdot)$  - це сигмоїdalна активаційна функція

#### MSELoss

Для тренування регресійної моделі LQP вихідні логіти  $z_i$  для групи (batch)

зображень  $I_N$  при  $i = 1\dots N$ , де  $N$  - кількість зображень в групі, цільова функція має вигляд:

$$\mathcal{L}_{reg} = \frac{1}{N} \sum_i^N (y_i - z_i)^2 \quad (3.2)$$

, де  $y_i$  - це кількість лейблів для зображення  $I_i$ .

### 3.5.2 Тренування VCNN

Тренування моделі ResNext [15] з нуля є досить складною задачою (дана модель має  $\approx 80M$  параметрів), адже для цього потрібні значні обчислювальні потужності.

Саме тому використовується натренована модель із адаптованим класифікатором (visual feature extractor) (Рис. 3.1), яка підганяється (finetuned) на обраному датасеті.

Існує два підходи для підгонки:

1) Підгонка всієї моделі (finetuning): всі шари моделі підганяються (дотреновуються) з низькою швидкістю навчання (learning rate). Даний підхід вимагає великої обчислювальної потужності, однак надає високу точність та досить таки швидко тренується (в порівнянні із тренуванням з нуля).

2) Підгонка класифікатора (transfer learning): відбувається тренування тільки класифікатора, фіксуючи всі інші параметри моделі. Даний підхід значно пришвидшує тренування в обмін на певну деградацію точності в порівнянні із 1-им варіантом.

В даній роботі використовується 1 варіант підгонки, так як він надає вищу точність.

### 3.5.3 Тренування MLP

Дана модель є звичайним багатошаровим персепtronом, тому її можна без проблем натренувати з нуля.

Також цю модель можна тренувати паралельно із VCNN.

### 3.5.4 Тренування LP

Дана модель призначена для обрахування вірогідностей на основі вектору  $f = [vf, tf]$ , оскільки вона складається із одного шару то її тренування також очевидне.

Також цю модель можна тренувати паралельно із LQP.

### 3.5.5 Тренування LQP

Дана модель є регресійним багатошаровим персепtronом, її тренування також є очевидним, однак варто нормалізувати вхідну к-сть лейблів, так як це пришвидшить та/або покращить збіжність моделі.

Під час тренування використовуються шари Dropout, так як дана модель дуже швидко перенавчається. В даній роботі вірогідність відкидання (dropout rate) 0.5.

Також цю модель можна тренувати паралельно із LP.

### 3.6 Процес тестування

#### 3.6.1 Тестування класифікаційних моделей (VCNN, MLP, LP)

Кожна із даних моделей обраховує вектор ймовірностей  $P$ , для тестування необхідно перевести вектор ймовірностей (наприклад:  $P = [0.9, 0.6, 0.1, 0.4, 0.6]$ ) у вигляд маркування (наприклад:  $M = [1, 1, 0, 0, 1]$ ). Дане перетворення називається індикаторною функцією.

Розглянемо два основних види індикаторної функції:

- 1) Порогове значення (threshold): для вектору ймовірностей  $P$  та порогу  $\alpha$  - вектор маркувань обраховується наступним чином: якщо  $y_i > \alpha$ , то маркуємо 1, інакше 0. Наприклад при  $\alpha = 0.5$ :  $[0.9, 0.6, 0.1, 0.4, 0.6] \rightarrow [1, 1, 0, 0, 1]$ .
- 2) Найкращі  $k$  (top  $k$ ): для вектору ймовірностей  $P$  та числа  $k$  - вектор маркувань обраховується наступним чином: маркуємо 1 найкращі  $k$  ймовірностей, інакше 0. Наприклад при  $k = 4$ :  $[0.9, 0.6, 0.1, 0.4, 0.6] \rightarrow [1, 1, 0, 1, 1]$ .

Оскільки дані моделі не передбачають передбачення кількості лейблів, то для тренування даних моделей використовується метод top  $k$ , при чому  $k = 3$ .

Тобто для зображення  $I$ , із маркуванням  $Y = [1, 0, 0, 0, 1]$ , і вектором ймовірностей  $P = [0.8, 0.9, 0.1, 0.5, 0.2]$  та результиручим вектором маркувань  $M$ :  $k = 3$ , перетворення  $P \rightarrow M \equiv [0.8, 0.9, 0.1, 0.3, 0.2] \rightarrow [1, 1, 0, 1, 0]$

#### 3.6.2 Тестування композитної моделі

Для тестування композитної моделі (Рис. 3.1) необхідно обрахувати результиуючі значення для моделей LP та LQP, і обрати top  $k$  лейблів LP, де  $k$  - це передбачення LQP.

### 3.7 Висновки до розділу

Було проведено опис архітектури композитної нейронної мережі. Результатуюча модель містить 4 компоненти (нейронні мережі), які виконують маркування зображення на основі даних із двома модальностями: зображення та тегів.

Також було наведено детальний опис процесу тренування та тестування, набору цільових функцій, які використовуються для тренування класифікаційни та регресійної моделей.

## 4 ЕКСПЕРИМЕНТИ

### 4.1 Датасет

Один із найбільш часто використовуваних датасетів для тестування моделей маркування зображень - NUS-WIDE [11], він складається із 269,655 зображень, 81 лейблу, та  $\approx 5000$  тегів в якості сторонньої текстової інформації. Для проведення тренування/тестування використовується розподіл, надведений разом із датасетом, так як він є збалансований настільки, наскільки це можливо (Рис. 4.2).

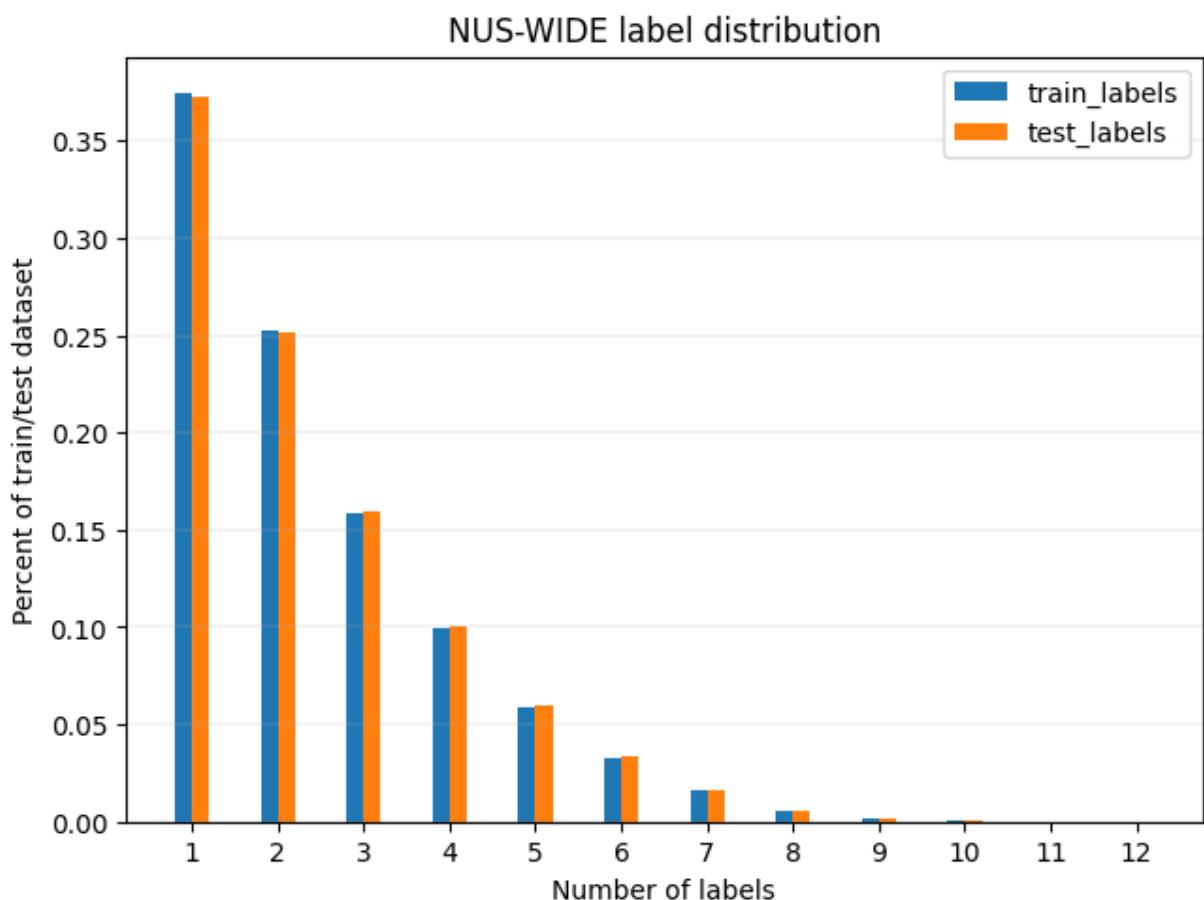


Рисунок 4.1 – Розподіл лейблів у тренувальному/тестовому датасеті

З наведеного графіку можна зробити висновок, що він містить значний дизбаланс відносно кількості класів, так як він був зібраний на основі реальних даних, якими користувались люди, із вебсайту 'Flickr'.

Важливо відмітити, що даний датасет містить посилання на зображення на ресурсі 'Flickr', і деякої частина цих зображень вже там немає.

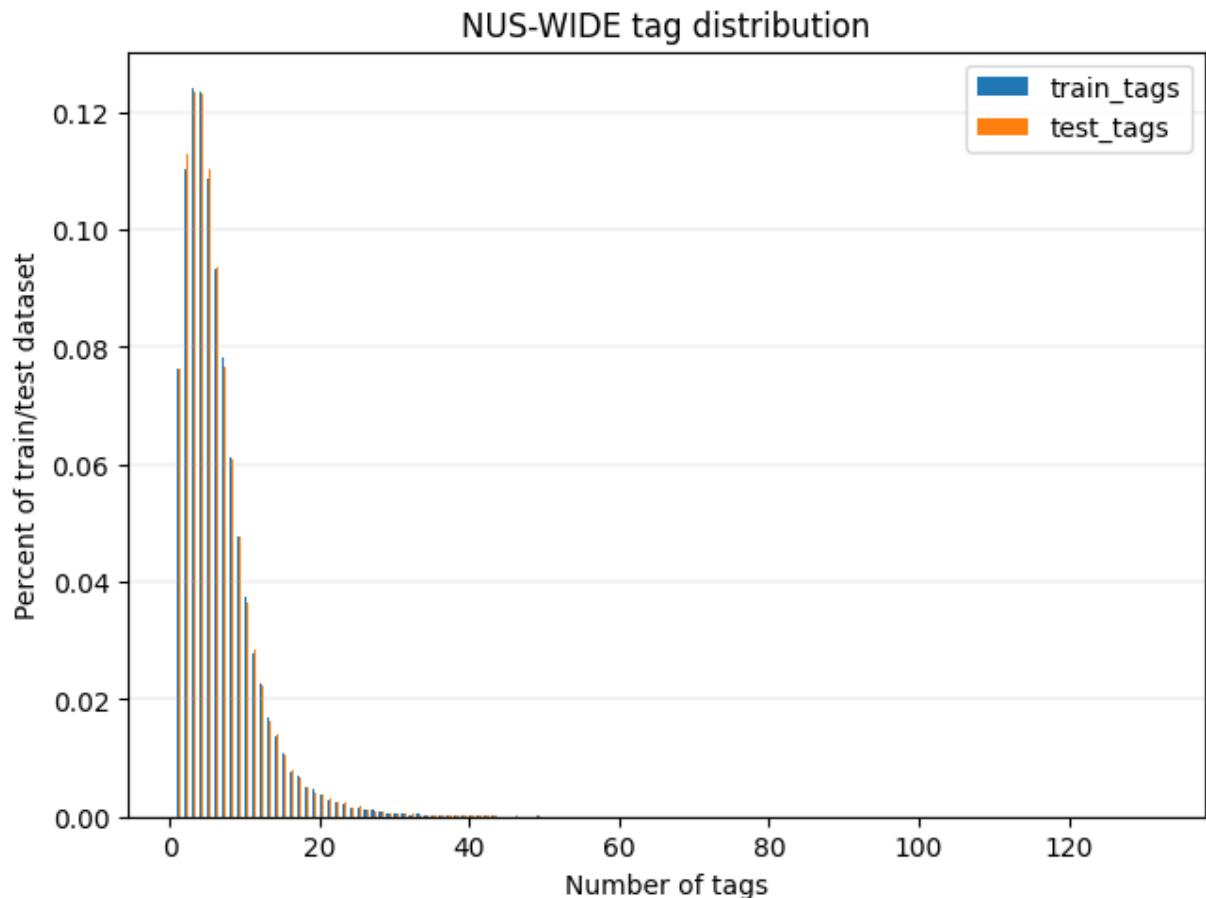


Рисунок 4.2 – Розподіл тегів у тренувальному/тестовому датасеті

Також буде використано тільки 1000 найбільш частих тегів з  $\approx 5000$ , при чому зображення, які не містять жодного тега відфільтровано.

#### 4.1.1 Множина цільових класів

Датасет NUS-WIDE [11] містить 81 цільовий клас.



Рисунок 4.3 – Візуалізація цільових класів у датасеті. Класи із більшою частотою мають більший шрифт тексту та темніший колір.

Множина цільових класів містить значний дизбаланс відносно частоти.

#### 4.1.2 Числові характеристики датасету

	Тренування	Тестування
Кількість зобаржень	121962	81636
Середня к-сть лейблів	2.42	2.43
Медіана к-сть лейблів	2	2
Мінімальна к-сть лейблів	1	1
Максимальна к-сть лейблів	12	13
Середня к-сть тегів	6.27	6.26
Медіана к-сть тегів	5	5
Мінімальна к-сть тегів	1	1
Максимальна к-сть тегів	131	125

Таблиця 4.1 – Характеристики тренувальної/тестової вибірок

З наведеної таблиці варто відзначити характеристики, що стосуються тегів. Так як теги - це стороння інформація, враховуючи що медіана = 5, то варто розглянути який вплив несе менша кількість тегів для маркування, так як зазвичай при завантаженні люди додають власноруч приблизно 3 теги. Тому в пода'льшому буде розглянуто вплив тегів, при виборі фіксованої максимальної кількості тегів.

## 4.2 Тестові метрики

Для оцінки точності будуть використовуватись метрики, які є загально вживаними для оцінки задачі маркування зображень (multi-label image annotation).

$$\begin{aligned}
 \text{C-P} &= \frac{1}{C} \sum_{j=1}^C \frac{NI_j^c}{NI_j^p} & \text{O-P} &= \frac{\sum_{i=1}^N NL_i^c}{\sum_{i=1}^N NL_i^p} \\
 \text{C-R} &= \frac{1}{C} \sum_{j=1}^C \frac{NI_j^c}{NI_j^g} & \text{O-R} &= \frac{\sum_{i=1}^N NL_i^c}{\sum_{i=1}^N NL_i^g} \\
 \text{C-F1} &= \frac{2 \cdot \text{C-P} \cdot \text{C-R}}{\text{C-P} + \text{C-R}} & \text{O-F1} &= \frac{2 \cdot \text{O-P} \cdot \text{O-R}}{\text{O-P} + \text{O-R}}
 \end{aligned} \tag{4.1}$$

,де

- \*  $C$  - к-сть класів
- \*  $N$  - к-сть тестових зображень
- \*  $NI_j^c$  - к-сть зображень які коректно промарковано як клас  $j$
- \*  $NI_j^g$  - к-сть зображень які мають клас  $j$
- \*  $NI_j^p$  - к-сть зображень які промарковано як клас  $j$
- \*  $NL_i^c$  - к-сть коректно промаркованих лейблів для зображення  $i$
- \*  $NL_i^g$  - к-сть лейблів які має зображення  $i$
- \*  $NL_i^p$  - к-сть промаркованих лейблів для зображення  $i$

Варто відзначити, що дані метрики є зміщеними (biased), при чому по-класові метрики (C) зміщені в сторону рідкісних класів, а загальні метрики (O) - в сторону частих класів [22].

Для того щоб отримати унформене представлення про ефективність моделі буде використовуватись наступна метрика, яка бере до уваги як C-F1 так і O-F1, що полегшує інтерпретацію результатів:

$$H-F1 = \frac{2 \cdot C-F1 \cdot O-F1}{C-F1 + O-F1} \quad (4.2)$$

## 4.3 Тренування

Для програмної реалізації запропонованої моделі було використано PyTorch.

Оскільки в даній роботі, використовується модель ResNext [15] натренована на датасеті ImageNet [9], то для вхідних зображень потрібно застосувати певне перетворення:

- 1) Зміна розміру (Resize)  $232 \times 232$ , використовуючи білінійну інтерполяцію
- 2) Центральний кроп (Central crop)  $224 \times 224$
- 3) Зміна масштабу (Rescale)  $[0,1]$
- 4) Нормалізація на основі статистичних величин ImageNet [9]. А саме: mean =  $[0.485, 0.456, 0.406]$  та std =  $[0.229, 0.224, 0.225]$

Дане перетворення доступно у бібліотеці PyTorch.

### 4.3.1 Параметри навчання

Класифікаційні моделі VCNN та MLP навчалися із швидкістю навання (learning rate) 0.001, а LP - зі швидкістю 0.01. Також для начання цих трьох моделей використовувався контроллер швидкості навчання (learning rate scheduler), який множив швидкість навчання на 0.5, досягаючи 5-ої та 10-ої епохи.

Регресійна модель LQP навчалась із сталою швидкістю навчання (learning rate) 0.0005.

Для всіх навчання всіх вище згаданих моделей використовувався оптимізатор AdamW, із параметром 11 регуляризації (weight decay) 0.0003.

Розмір групи (batch size) 32.

Також варто відзначити, що в даній роботі епоха - це 20% від усіх даних, при чому після кожної епохи дані перемішуються (shuffle), отримаючи нові 20% даних.

### 4.3.2 Процес навчання

Для тренування було використано графічний процесор 'Nvidia L4'. Для тренування всіх елементів моделі знадобилось  $\approx 3$  години.

Для оптимізації процесу тренування було застосовано техніку mixed precision, яка використовує f16 замість f32, під час певних етапів тренування [23].

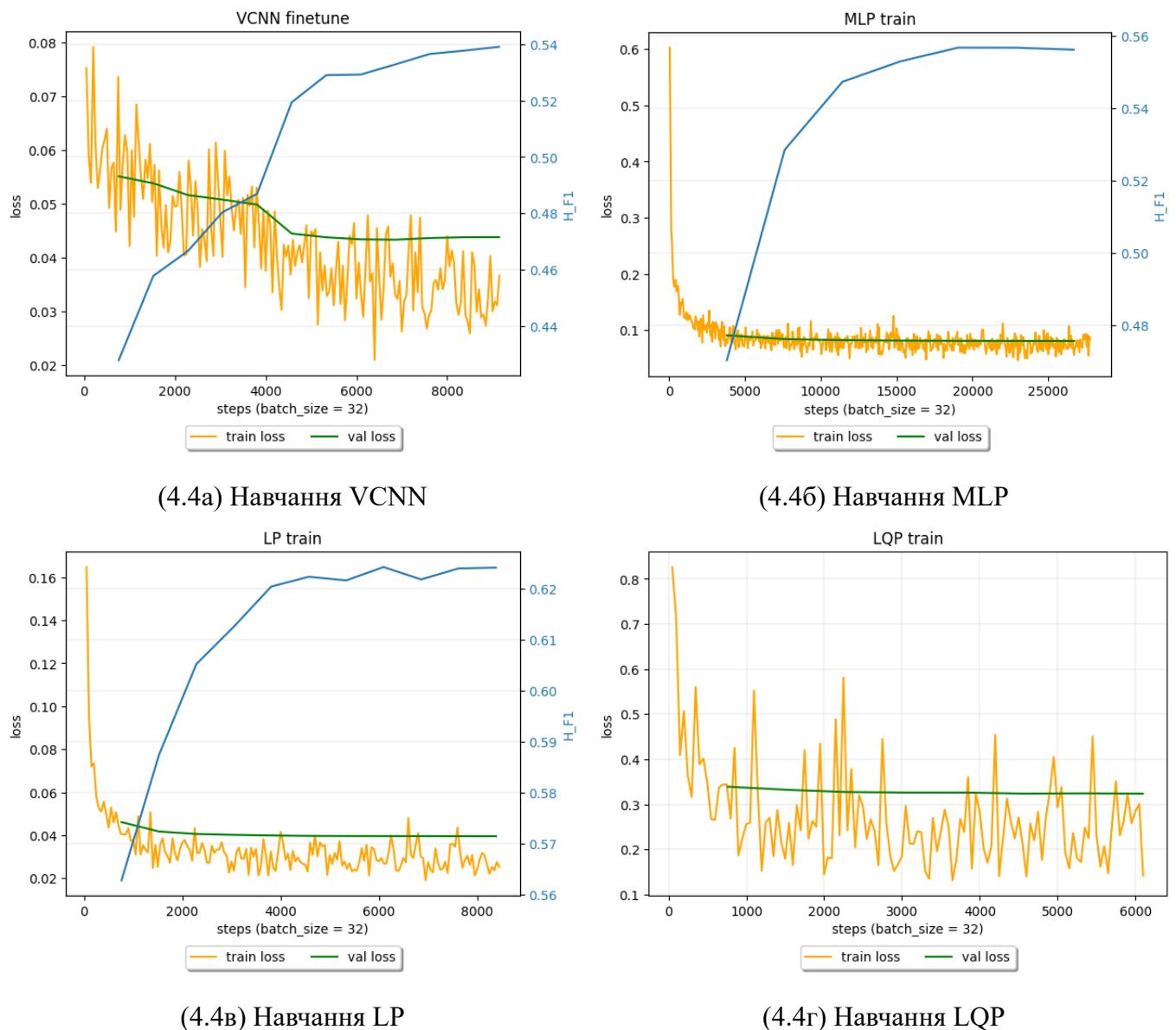


Рисунок 4.4 – Процес навчання моделей

З наведених вище графіків можна прийти до висновку, що кожна із 4-ох

компонентів моделі успішно завершила процес навчання (збіглась).

#### 4.4 Висновки до розділу

Було проведено детальний огляд обраного датасету:

- \* Тренувальна/тестова вибірки містять 121962 та 81636 зображень відповідно
- \* Всі зображення, які не містять жодного тегу було відфільтровано
- \* Множина лейблів (цільових класів) містить 81 унікальне значення
- \* В якості додаткової інформації використовуються теги, які містять 1000 унікальних значень

- \* цільові класи містять значний дизбаланс відносно частот

Детально наведено тестові метрики, дані метрики використовуються в абсолютній більшості робіт, що дає можливість проводити детальний порівняльний аналіз.

Наведено деталі для перетворення даних, для роботи мережі. Підібрано оптимальні значення для проведення навчання та графіки, які описують сам процес навчання.

## 5 АНАЛІЗ РЕЗУЛЬТАТІВ

### 5.1 Аналіз компонентів моделі

Модель	Індикаторна ф-ція (3.6.1)	Модальність	C-P	C-R	C-F1	O-P	O-R	O-F1	H-F1
Композитна	top k	Зображення+теги	72.49	59.51	65.36	76.53	74.41	75.46	70.04
VCNN+MLP+LP	top 3	Зображення+теги	60.51	61.28	60.89	67.87	71.52	63.98	62.40
VCNN+LQP	top k	Зображення	64.62	37.61	47.54	77.07	59.01	66.84	55.56
VCNN	top 3	Зображення	44.48	53.32	48.50	55.44	68.52	61.29	54.15

Таблиця 5.1 – Порівняння компонентів моделі

Результатуюча композитна модель показала значно кращий результат ніж базове рішення (VCNN).

### Додаткова модальність (MLP+LP)

Додавання додаткової модальності у вигляді тегів внесло значний вклад у підвищення якості маркування. Порівнюючи відповідні метрики H-F1 для моделей VCNN та VCNN+MLP+LP, можна побачити приріст на 8.25%. При чому варто відзначити, що цей ріст в основному забезпечений приростом метрики C-F1, яка є зміщеною в сторону більш рідкісних класів. Варто відзначити, що це працює завдяки тому що зазвичай теги надані користувачами відмічають досить рідкісні поняття, які на відміну від частих тегів (небо, сонце, людина, вода і тд.) складно розпізнати маючи одне лише зображення.

### Передбачення кількості лейблів (LQP)

При використанні компоненту LQP кількість лейблів обирається за принципом 'top k', а не 'top 3' (3.6.1). Це очевидним чином підвищує точність фінального маркування, адже деякі зображення можуть мати більше трьох лейблів, інші - менше трьох. Порівнюючи вплив компоненти LQP для базового рішення (VCNN) та композитної

моделі (VCNN+MLP+LP+LQP) можна зробити припущення, що VCNN аналізує загальні поняття на зображенні, а враховуючи дизбаланс класів у датасеті, використання принципу 'top k' збільшує точність (precision), сильно жертвуючи по-класовим охопленням (C-R), і, як наслідок, не сильно збільшує величину головної метрики H-F1. На практиці це виливалось у те, абсолютна більшість зображень маркувалась частими лейблами (людина, вода, небо і тд.), а рідкісні теги - ігнорувались. Натомість у композитній моделі вищезгаданий принцип чудово проявив себе. Згідно із тестовими метриками покращення становить 7.64%.

## 5.2 Аналіз впливу додаткової інформації

При завантаженні зображення на певний ресурс люди можуть вказати додаткову тегову інформацію для покращення роботи автоматичного маркування, однак зазвичай вони вказують до 4-5 тегів, а оскільки медіана кількості тегів у датасеті становить 5, то варто розглянути вплив кількості тегів на процес тегування більш детально.

Максимальна кількість тегів	C-P	C-R	C-F1	O-P	O-R	O-F1	H-F1
0	44.48	53.32	48.50	55.44	68.52	61.29	54.15
1	66.42	37.28	47.75	74.80	66.78	70.56	56.96
2	70.20	43.00	53.33	75.50	68.75	71.97	61.26
3	71.06	46.97	56.55	75.70	69.85	72.66	63.60
4	71.11	49.80	58.57	75.95	71.00	73.39	65.15
5	71.99	52.62	60.80	76.19	71.95	74.01	66.76
6	71.92	53.37	61.49	76.25	72.44	74.30	67.29
7	71.86	54.65	62.08	76.35	72.85	74.56	67.75
8	72.25	56.03	63.12	76.49	73.26	74.84	68.48
9	72.07	56.55	63.37	76.57	73.62	75.07	68.73
10	72.33	57.13	63.84	76.59	73.83	75.18	69.05
-	72.49	59.51	65.36	76.53	74.41	75.46	70.04

Таблиця 5.2 – Порівняння впливу кількості тегів на маркування

З наведеної таблиці можна зробити висновок, що між якістю маркування та кількістю тегів є пряма залежність. В середньому приріст точності відносно метрики H-F1 становить 1 – 2%. Також варто відзначити, що після проходження медіані (5) приріст спадає до менше ніж 0.5%.

Таким чином для того щоб досягати оптимальної якості із найменшою кількістю додаткових тегів потрібно розглядати від 2 до 5 тегів.

### 5.3 Порівняння з існуючими рішеннями

Модель	Індикаторна ф-ція (3.6.1)	Модальність	C-P	C-R	C-F1	O-P	O-R	O-F1	H-F1
Композитна	top k	Зображення+теги	72.49	59.51	65.36	76.53	74.41	75.56	70.04
Query2Label [4]	threshold $\alpha$	Зображення (+аналіз класів)	-	-	67.60	-	-	76.3	71.69
SR-CNN-RNN [1]	top 3	Зображення+теги	71.73	61.73	66.36	77.41	76.88	77.15	71.35
Resnet-CPSD [5]	threshold $\alpha$	Зображення (+аналіз класів)	-	-	64.00	-	-	75.30	69.19
MS-CMA [3]	threshold $\alpha$	Зображення (+аналіз класів)	-	-	60.50	-	-	73.80	66.49
Resnet-SRN [2]	threshold 0.5	Зображення (+аналіз класів)	65.20	55.80	58.50	75.50	71.50	73.40	65.10
SINN [17]	top 3	Зображення+теги	58.30	60.63	59.44	57.05	79.12	66.29	62.68
TagNeighbour [18]	top 3	Зображення+метадані	54.74	57.30	55.99	53.46	75.10	62.46	59.05
CNN+Logistic [17]	top 3	Зображення	45.60	45.03	45.31	51.32	70.77	59.50	51.44
CNN-RNN [20]	top 3	Зображення	40.50	30.40	34.70	49.9	61.70	55.20	42.61
CNN+WARP [22]	top 3	Зображення	31.65	35.60	33.51	48.59	60.49	53.89	41.32
CNN+Softmax [22]	top 3	Зображення	31.68	31.22	31.45	47.82	59.52	53.03	39.48

Таблиця 5.3 – Порівняння результатуючих метрик для різних моделей на датасеті  
NUS-WIDE

**Точність запропонованого рішення** Композитна модель (VCNN+MLP+LP+LQP) продемонструвала високу якість маркування на тестових метриках у порівнянні із розглянутими альтернативними рішеннями. Згідно із метрикою H-F1 запропоноване рішення є третім.

**Модальність даних** Задача маркування зображень розглядає зображення як основну модальність, однак додавання модальності, очікувано, покращує результати маркування. Це підтверджують метрики наведені в Табл. 5.3. Серед розглянутих рішень є 3 варіанти модальності даних з якимим працюють нейронні мережі. Найменш ефективним, як і очікувалось, виявилися моделі які аналізують виключно

зображення. Введення інших двох видів додаткових модальностей: теги, аналіз класів, - надають значно кращі результати. Варто відзначити, - аналіз класів (найкраще імплементовано в: Query2Label [4], Resnet-CPSC [5] та MS-CMA [3]) не потребує ніяких додаткових даних окрім зображення, що є вагомою конкурентною перевагою, враховуючи незначну відміність в точності моделей.

**Індикаторна функція** Для оцінки ефективності запропонованої індикаторної функції 'top k', варто ізолювати вплив саме цієї функції. Для цього розглянемо існуючі моделі, які працюють із тією ж модальністю даних. Найкращою із таких моделей є SINN [17]. Використання моделі LQP, яка передбачає роботу із динамічною кількістю лейблів при маркуванні (top k) значено підвищує якість. Згідно із наведеними метриками покращення складає 7.36% (Табл. 5.3), що є вагомим приrostом.

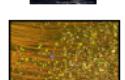
## 5.4 Демонстративні приклади

З тестового датасету випадковим чином обрано декілька зображень, для демонстрації роботи моделі:

Image	Truth	Model pred	Image	Truth	Model pred
	clouds sky	clouds sky		snow water	glacier lake water
	flowers plants	flowers plants		clouds mountain rocks sky	mountain rocks sky
	cityscape clouds sky	buildings clouds nighttime sky		buildings clouds	flowers
	clouds	clouds sky		clouds person sky	buildings person sky

(5.1a)

(5.1б)

Image	Truth	Model pred	Image	Truth	Model pred
	clouds ocean water	clouds military sky		person wedding	person wedding
	person	person sky		bear	animal
	clouds moon ocean sky water	lake moon ocean sky water		clouds nighttime sky window flowers	buildings clouds nighttime sky
	flowers grass water	grass plants		garden grass plants water	flowers garden grass plants

(5.1в)

(5.1г)

Рисунок 5.1 – Демонстративні приклади

Задамо умовне позначення "a::1", що означає демонстративний приклад "a", перше зображення зверху, "в::2" - приклад "в", друге зображення зверху і тд.

Серед наведених прикладів можна розглянути кілька цікавих моментів, які не

відображають тестові метрики:

- \* Іноді модель передбачає маркування, якого немає у датасеті, однак присутнє на зображені. Наприклад: [a::3,a::4,б::1,б::3,б::4,в::1,в::2,г::2,г::3].

- \* Існують випадки коли модель відмічає поняття, які не можуть бути присутніми на одному зображені. Це є прямим наслідком того, що наша модель розглядає цільові класи як незалежі сутності. Наприклад: lake та ocean як-от в прикладі 'в::3'.

- \* Іноді передбачення моделі відсікають неіснуючі поняття та маркують зображення краще ніж це було зроблено в датасеті. Так для зображення 'б::3' на якому зображено якусь рослину датасет вказує що це: 'buildings, clouds'; а модель - 'flowers'.

Окрім наведених вище особливих випадків, іноді модель, звичайно, помиляється. Однак у наведених прикладах немає значних помилок у маркуванні.

## 5.5 Висновки до розділу

Проведено аналіз компонентів моделі та доведено ефективність кожного із них.

Розглянуто вплив максимальної кількості тегів на якість маркування, і визначено, що перші декілька максимальних значень (від 1 до 5) вносять значне покращення на якість маркування, однак після проходження медіані (5 тегів) кількості тегів, відбувається деградація цього ефекту, і приріст стає незначним.

У порівнянні із існуючими рішеннями модель показала себе чудово, однак варто зазначити, що найкращі рішення такі як Query2Label [4], Resnet-CPSD [5] та ін. досягають співставної точності навіть без додаткової інформації, що є суттєвою перевагою.

Також було розглянуто декілька демонстративних прикладів роботи моделі.

## ВИСНОВКИ

В даній роботі було розглянуто композитну модель, для маркування зображень (шпалерів робочого столу) проведено оцінювання її точності на тестових метриках.

Розглянута модель показала хороший результат у порівнянні із існуючими альтернативними рішеннями.

До переваг розглянутого рішення належать:

- + Висока точність
- + Невелика кількість параметрів ( $\approx 95\%$  параметрів має модель для аналізу зображень)
- + Висока швидкість тренування

Недоліками є:

- Неможливість тренування моделі в один етап (end-to-end)
- Необхідність використання додаткових даних (тегів) для отримання високої якості опису зображення

Також було розглянуто вплив кількості тегів у додатковій інформації на якість маркування, та визначено оптимальну кількість тегів для ефективної роботи даної моделі, що становить від 2 до 5 тегів.

В подальшому варто розглянути ефективність даної моделі на інших датасетах (наприклад MSCOCO [10]). також варто розглянути і методи для аналізу цільових класів, так як незважаючи на значне ускладнення фінальної моделі це надає досить високий приріст до точності маркування.

Результатуюча композитна модель збережена у форматі safetensors.

## ПЕРЕЛІК ПОСИЛАНЬ

- [1] F. Liu та ін. «Semantic Regularisation for Recurrent Image Annotation». B: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, лип. 2017, C. 4160—4168. doi: 10 . 1109 / CVPR . 2017 . 443. URL: <https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.443>.
- [2] F. Zhu та ін. «Learning Spatial Regularization with Image-Level Supervisions for Multi-label Image Classification». B: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, лип. 2017, C. 2027—2036. doi: 10 . 1109 / CVPR . 2017 . 219. URL: <https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.219>.
- [3] Renchun You та ін. «Cross-Modality Attention with Semantic Graph Embedding for Multi-Label Classification». B: *Proceedings of the AAAI Conference on Artificial Intelligence* 34 (квіт. 2020), C. 12709—12716. doi: 10.1609/aaai.v34i07.6964.
- [4] Shilong Liu та ін. «Query2Label: A Simple Transformer Way to Multi-Label Classification». B: (лип. 2021). arXiv: 2107.10834 [cs.CV].
- [5] Jiazhi Xu та ін. «Boosting multi-Label Image Classification with complementary Parallel Self-distillation». B: (трав. 2022). arXiv: 2205.10986 [cs.CV].
- [6] Dan Ciregan, Ueli Meier та Jürgen Schmidhuber. «Multi-column deep neural networks for image classification». B: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, C. 3642—3649. doi: 10.1109/CVPR.2012.6248110.

- [7] Karen Simonyan та Andrew Zisserman. «Very deep convolutional networks for large-scale image recognition». B: (вер. 2014). arXiv: 1409.1556 [cs.CV].
- [8] Keiron O'Shea та Ryan Nash. «An Introduction to Convolutional Neural Networks». B: (листоп. 2015). arXiv: 1511.08458 [cs.NE].
- [9] Li Deng та ін. «Imagenet: A large-scale hierarchical image database». B: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, C. 248—255.
- [10] Tsung-Yi Lin та ін. «Microsoft COCO: Common Objects in Context». B: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: <http://arxiv.org/abs/1405.0312>.
- [11] Tat-Seng Chua та ін. «NUS-WIDE: A Real-World Web Image Database from National University of Singapore». B: *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*. Santorini, Greece., July 8-10, 2009.
- [12] Kaiming He та ін. «Deep Residual Learning for Image Recognition». B: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Черв. 2016.
- [13] Krizhevsky Alex, Sutskever Ilya та Hinton Geoffrey. «ImageNet Classification with Deep Convolutional Neural Networks». B: *Advances in Neural Information Processing Systems*. За ред. F. Pereira та ін. Т. 25. Curran Associates, Inc., 2012. URL: [https://proceedings.neurips.cc/paper\\_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf).
- [14] Christian Szegedy та ін. *Going Deeper with Convolutions*. 2014. eprint: arXiv: 1409.4842.
- [15] Saining Xie та ін. «Aggregated Residual Transformations for Deep Neural Networks». B: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, C. 5987—5995. doi: 10.1109/CVPR.2017.634.

- [16] Alexey Dosovitskiy та ін. «An image is worth 16x16 words: Transformers for image recognition at scale». B: (жовт. 2020). arXiv: 2010.11929 [cs.CV].
- [17] Hexiang Hu та ін. «Learning Structured Inference Neural Networks with Label Relations». B: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, C. 2960—2968. doi: 10.1109/CVPR.2016.323.
- [18] J. Johnson, L. Ballan та L. Fei-Fei. «Love Thy Neighbors: Image Annotation by Exploiting Image Metadata». B: *2015 IEEE International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, груд. 2015, C. 4624—4632. doi: 10 . 1109 / ICCV . 2015 . 525. URL: <https://doi.ieeecomputersociety.org/10.1109/ICCV.2015.525>.
- [19] K. Tang та ін. «Improving Image Classification with Location Context». B: *2015 IEEE International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, груд. 2015, C. 1008—1016. doi: 10.1109/ICCV.2015.121. URL: <https://doi.ieeecomputersociety.org/10.1109/ICCV.2015.121>.
- [20] J. Wang та ін. «CNN-RNN: A Unified Framework for Multi-label Image Classification». B: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, черв. 2016, C. 2285—2294. doi: 10 . 1109 / CVPR . 2016 . 251. URL: <https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.251>.
- [21] Nitish Srivastava та ін. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». B: *Journal of Machine Learning Research* 15.56 (2014), C. 1929—1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [22] Yunchao Gong та ін. *Deep Convolutional Ranking for Multilabel Image Annotation*. 2013. eprint: arXiv:1312.4894.

- [23] Paulius Micikevicius та ін. «Mixed Precision Training». В: (жовт. 2017). arXiv: 1710.03740 [cs.AI].

## Додаток А

### Лістинг програм

Лістинг файлу 1: data.py - Класи для підготовки та паралеїзації завантаження даних

---

```

import os
import torch
import random
import pandas as pd
import lightning as lg
import torchvision
from pathlib import Path
from models.utils import TagTransform
from torchvision.io import read_image, ImageReadMode
from torchvision.transforms import v2 as transforms
from torch.utils.data import Dataset, DataLoader

class CustomDataset(Dataset):

    def __init__(self, files: list[str] | str, root: str, load_images: bool = True, transform=None, tag_transform=None, label_transform=None):
        self.root = root
        self.load_images = load_images
        self.transform = transform
        self.tag_transform = tag_transform
        self.label_transform = label_transform
        self.images = pd.read_json(files[0], lines=True)
        if len(files) > 1:
            for f in files[1:]:
                tmp = pd.read_json(f, lines=True)
                self.images = pd.concat([self.images, tmp], ignore_index=True)
        if imgs := os.environ.get("SPECIFIC_IMGS"):
            imgs = imgs.split(",")
            df = self.images[self.images["file_name"] == imgs[0]]
            for img in imgs[1:]:
                tmp = self.images[self.images["file_name"] == img]
                df = pd.concat([df, tmp], ignore_index=True)
            self.images = df
        if n_tags := os.environ.get("RANDOM_NTAGS"):
            n_tags = int(n_tags)
            self.images["tags"] = self.images["tags"].apply(lambda x: random.sample(x, n_tags) if len(x) > n_tags else x)

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

    def __getitem__(self, idx):
        row = self.images.iloc[idx]

```

```

image_path = Path(
    self.root,
    row["image_root"],
    row["file_name"],
)
if self.load_images:
    image = read_image(str(image_path), mode = ImageReadMode.RGB)
else:
    image = 0
labels = row["labels"]
tags = row["tags"]
if self.load_images and self.transform:
    image = self.transform(image)
if self.label_transform:
    labels = self.label_transform(labels)
if self.tag_transform:
    tags = self.tag_transform(tags)
return image, tags, labels

```

```
class DataModule(lg.LightningDataModule):
```

```

def __init__(self, root: str = "", load_images: bool = True, batch_size: int = 32, prefetch_factor: int = None, num_workers: int = 0):
    super().__init__()
    self.load_images = load_images
    self.batch_size = batch_size
    self.num_workers = num_workers
    self.prefetch_factor = prefetch_factor
    self.prepare_data_per_node = False
    self.transform = transforms.Compose([
        torchvision.models.ResNeXt101_32X8D_Weights.IMAGENET1K_V2.transforms()
    ])
    self.root = Path(root)
    self.dataroot = self.root / "assets" / "preprocessed"
    self.tag_transform = transforms.Compose([
        TagTransform(self.dataroot / "Tags_nus-wide.ndjson"),
        transforms.ToDtype(torch.float32)
    ])
    self.label_transform = transforms.Compose([
        TagTransform(self.dataroot / "Labels_nus-wide.ndjson"),
        transforms.ToDtype(torch.float32)
    ])
    self.train_data = [
        str(self.dataroot / "Train_nus-wide.ndjson"),
    ]
    self.test_data = [
        str(self.dataroot / "Test_nus-wide.ndjson"),
    ]

```

```

def setup(self, stage: str):
    if stage == "fit":
        self.train = CustomDataset(
            self.train_data,
            root = self.root,
            load_images = self.load_images,
            transform = self.transform,
            label_transform = self.label_transform,
            tag_transform = self.tag_transform,
        )
        self.val = CustomDataset(
            self.test_data,
            root = self.root,
            load_images = self.load_images,
            transform = self.transform,
            label_transform = self.label_transform,
            tag_transform = self.tag_transform,
        )

    if stage == "test":
        self.test = CustomDataset(
            self.test_data,
            root = self.root,
            load_images = self.load_images,
            transform = self.transform,
            label_transform = self.label_transform,
            tag_transform = self.tag_transform,
        )

    if stage == "predict":
        self.predict = CustomDataset(
            self.test_data,
            root = self.root,
            load_images = self.load_images,
            transform = self.transform,
            label_transform = self.label_transform,
            tag_transform = self.tag_transform,
        )

def train_dataloader(self):
    return DataLoader(
        self.train, batch_size=self.batch_size, shuffle=True,
        num_workers=self.num_workers, pin_memory=True,
        prefetch_factor=self.prefetch_factor
    )

def val_dataloader(self):
    return DataLoader(
        self.val, batch_size=self.batch_size,

```

```

    num_workers=self.num_workers, pin_memory=True,
    prefetch_factor=self.prefetch_factor
)

def test_dataloader(self):
    return DataLoader(
        self.test, batch_size=self.batch_size,
        num_workers=self.num_workers, pin_memory=True,
        prefetch_factor=self.prefetch_factor
    )

def predict_dataloader(self):
    return DataLoader(
        self.predict, batch_size=self.batch_size,
        num_workers=self.num_workers, pin_memory=True,
        prefetch_factor=self.prefetch_factor
    )
)

```

Лістинг файлу 2: models/utils.py - утиліти для вирішення певних проміжних задач

```

import torch
import numpy as np
import polars as pl
import torch.nn.functional as F
from torch import Tensor
from pathlib import Path
from typing import Iterable
from collections import defaultdict
from torchvision.transforms import v2 as transforms
from torchtext.data.utils import get_tokenizer
from torchtext.vocab import build_vocab_from_iterator

nlabels_f32 = transforms.Compose([
    transforms.Lambda(lambda x: x.sum(axis=1)),
    transforms.Lambda(lambda x: x.unsqueeze(1)),
])
# Calculated in eda.ipynb
LAB_MEAN = 2.4186057952477
LAB_STD = 1.5880828167376047

nlabels_normalize = transforms.Compose([
    transforms.Lambda(lambda x: (x - LAB_MEAN) / LAB_STD)
])
nlabels_denormalize = transforms.Compose([
    transforms.Lambda(lambda x: (x * LAB_STD) + LAB_MEAN)
])

```

])

**class Metrics:**

```

def __init__(self, num_classes: int):
    self.num_classes = num_classes
    self.reset()

def update(self, pred: Tensor, target: Tensor):
    cls_tp = ((pred == 1) & (target == 1)).sum(dim=0).cpu().numpy()
    cls_tn = ((pred == 0) & (target == 0)).sum(dim=0).cpu().numpy()
    cls_fp = ((pred == 1) & (target == 0)).sum(dim=0).cpu().numpy()
    cls_fn = ((pred == 0) & (target == 1)).sum(dim=0).cpu().numpy()
    im_tp = ((pred == 1) & (target == 1)).sum(dim=1).cpu().numpy()
    im_tn = ((pred == 0) & (target == 0)).sum(dim=1).cpu().numpy()
    im_fp = ((pred == 1) & (target == 0)).sum(dim=1).cpu().numpy()
    im_fn = ((pred == 0) & (target == 1)).sum(dim=1).cpu().numpy()
    self._cls_tp = np.add(self._cls_tp, cls_tp)
    self._cls_tn = np.add(self._cls_tn, cls_tn)
    self._cls_fp = np.add(self._cls_fp, cls_fp)
    self._cls_fn = np.add(self._cls_fn, cls_fn)
    self._im_tp = np.append(self._im_tp, im_tp)
    self._im_tn = np.append(self._im_tn, im_tn)
    self._im_fp = np.append(self._im_fp, im_fp)
    self._im_fn = np.append(self._im_fn, im_fn)

def reset(self):
    self._cls_tp = np.zeros(self.num_classes)
    self._cls_tn = np.zeros(self.num_classes)
    self._cls_fp = np.zeros(self.num_classes)
    self._cls_fn = np.zeros(self.num_classes)
    self._im_tp = np.array([])
    self._im_tn = np.array([])
    self._im_fp = np.array([])
    self._im_fn = np.array([])

def CP(self):
    scls_tp_fp = self._cls_tp + self._cls_fp
    val = np.divide(
        self._cls_tp,
        scls_tp_fp,
        out = np.zeros_like(self._cls_tp),
        where = (scls_tp_fp != 0),
    )
    return val.mean()

def CR(self):
    scls_tp_fn = self._cls_tp + self._cls_fn

```

```

val = np.divide(
    self._cls_tp,
    scls_tp_fn,
    out = np.zeros_like(self._cls_tp),
    where = (scls_tp_fn != 0),
)
return val.mean()

def CF1(self):
    cp = self.CP()
    cr = self.CR()
    return 2 * (cp * cr) / (cp + cr + 1e-12)

def IP(self):
    sim_tp_fp = self._im_tp + self._im_fp
    return self._im_tp.sum() / (sim_tp_fp.sum() + 1e-12)

def IR(self):
    sim_tp_fn = self._im_tp + self._im_fn
    return self._im_tp.sum() / (sim_tp_fn.sum() + 1e-12)

def IF1(self):
    ip = self.IP()
    ir = self.IR()
    return 2 * (ip * ir) / (ip + ir + 1e-12)

def HF1(self):
    cf1 = self.CF1()
    if1 = self.IF1()
    return 2 * (cf1 * if1) / (cf1 + if1 + 1e-12)

class TagTransform:

    def __init__(self, file: str | Path):
        self._tags = pl.read_ndjson(file)
        tags = set(self._tags["name"].to_list())
        tokenizer = get_tokenizer("basic_english")
        self.voc = build_vocab_from_iterator(
            [tokenizer(tag) for tag in tags],
        )
        self.tags = set(self.voc.get_stoi().keys())
        self.voclen = len(self.voc)

    def __call__(self, sample: Iterable) -> Tensor:
        return self.encode(sample)

    def encode(self, sample: Iterable) -> Tensor:
        ftags = [t for t in sample if t in self.tags]

```

```

if ftags == []:
    return torch.zeros(self.voclen)
val = F.one_hot(
    torch.tensor(self.voc.forward(ftags)),
    num_classes=self.voclen
).amax(dim=0)
return val

def decode(self, cls: Tensor) -> list[list[str]]:
    bins = torch.argwhere(cls == 1)
    d = defaultdict(list)
    for k, v in bins:
        d[k.item()].append(v.item())
    ltokens = list(d.values())
    out = []
    for tokens in ltokens:
        out.append(self.voc.lookup_tokens(tokens))
    return out

def decode_topn(self, cls: Tensor, clen: Tensor) -> Tensor:
    sort_cls = cls.argsort(dim=1, descending=True)
    for i, (_class, _len) in enumerate(zip(sort_cls, clen)):
        iones = _class[:_len]
        cls[i] = torch.zeros_like(cls[i])
        cls[i] = cls[i].put(iones, torch.ones(len(iones), device=cls.device, dtype=cls.dtype))
    return cls

```

### Лістинг файлу 3: models/vcnn.py - модель VCNN

```

import torch
import torch.nn as nn
import torchvision
import lightning as lg
from torch import Tensor
from models.utils import Metrics, TagTransform

label_transform = TagTransform("./assets/preprocessed/Labels_nus-wide.ndjson")

class VCNN(lg.LightningModule):

    def __init__(self, lr: float = ..., weight_decay: float = ...):
        super().__init__()
        self.save_hyperparameters()
        base = torchvision.models.resnext101_32x8d(
            weights=torchvision.models.ResNeXt101_32X8D_Weights.IMAGENET1K_V2
        )

```

```

self.nfilters = base.fc.in_features
self.feature_extractor = nn.Sequential(
    base.conv1,
    base.bn1,
    base.relu,
    base.maxpool,
    base.layer1,
    base.layer2,
    base.layer3,
    base.layer4,
    base.avgpool,
    nn.Flatten(),
)
self.fc = nn.Sequential(
    nn.Linear(self.nfilters, 81),
)
self.loss_module = nn.BCEWithLogitsLoss()
self.activation = nn.Sigmoid()
self.metrics = Metrics(81)

def configure_optimizers(self):
    optimizer = torch.optim.AdamW(
        filter(lambda p: p.requires_grad, self.parameters()),
        lr = self.hparams.lr,
        weight_decay = self.hparams.weight_decay,
    )
    scheduler = torch.optim.lr_scheduler.MultiStepLR(
        optimizer,
        milestones=[5,10],
        gamma=0.1,
    )
    return [optimizer], [{"scheduler": scheduler, "interval": "epoch"}]

def predict(self, x: Tensor):
    x = self.forward(x)
    x = self.activation(x)
    return x

def forward(self, x: Tensor):
    x = self.feature_extractor(x)
    x = self.fc(x)
    return x

def training_step(self, batch, batch_idx):
    (image, _, labels) = batch
    pred = self.forward(image)
    loss = self.loss_module(pred, labels)
    self.log("train_loss", loss, on_step=True, prog_bar=True)
    return loss

```

```

def validation_step(self, batch, batch_idx):
    (image, _, labels) = batch
    pred = self.forward(image)
    loss = self.loss_module(pred, labels)
    labels = labels.to(torch.int64)
    pred = label_transform.decode_topn(pred, torch.tensor([3] * pred.shape[1]))
    pred = pred.to(torch.int64)
    self.metrics.update(pred, labels)
    self.log("val_loss", loss, prog_bar=True)

def test_step(self, batch, batch_idx):
    self.validation_step(batch, batch_idx)

def on_validation_epoch_end(self):
    cp, cr = self.metrics.CP(), self.metrics.CR()
    cf1 = self.metrics.CF1()
    ip, ir = self.metrics.IP(), self.metrics.IR()
    ifl = self.metrics.IF1()
    hfl = self.metrics.HF1()
    self.log("CP", cp)
    self.log("CR", cr)
    self.log("IP", ip)
    self.log("IR", ir)
    self.log("C_F1", cf1)
    self.log("I_F1", ifl)
    self.log("H_F1", hfl, prog_bar=True)
    self.metrics.reset()

def on_test_epoch_end(self):
    self.on_validation_epoch_end()

```

### Лістинг файлу 4: models/mlp.py - модель MLP

```

import torch
import lightning as lg
from torch import Tensor
from models.utils import Metrics, TagTransform

label_transform = TagTransform("./assets/preprocessed/Labels_nus-wide.ndjson")

class MLP(lg.LightningModule):

    def __init__(self, lr: float = ..., weight_decay: float = ...):
        super().__init__()
        self.save_hyperparameters()

```

```

self.fc1 = torch.nn.Sequential(
    torch.nn.Linear(1000, 2048),
    torch.nn.ReLU(),
)
self.fc2 = torch.nn.Sequential(
    torch.nn.Linear(2048, 81),
)
self.activation = torch.nn.Sigmoid()
self.loss_module = torch.nn.BCEWithLogitsLoss(
    pos_weight = torch.ones(81) * 2
)
self.metrics = Metrics(81)

def configure_optimizers(self):
    optimizer = torch.optim.AdamW(
        filter(lambda p: p.requires_grad, self.parameters()),
        lr = self.hparams.lr,
        weight_decay = self.hparams.weight_decay,
    )
    scheduler = torch.optim.lr_scheduler.MultiStepLR(
        optimizer,
        milestones=[5,10],
        gamma=0.1
    )
    return [optimizer], [{"scheduler": scheduler, "interval": "epoch"}]

def predict(self, x: Tensor):
    x = self(x)
    x = self.activation(x)
    return x

def forward(self, x: Tensor):
    x = self.fc1(x)
    x = self.fc2(x)
    return x

def training_step(self, batch, batch_idx):
    _, tags, labels = batch
    pred = self.forward(tags)
    loss = self.loss_module(pred, labels)
    self.log("train_loss", loss, on_step=True, prog_bar=True)
    return loss

def validation_step(self, batch, batch_idx):
    _, tags, labels = batch
    pred = self.forward(tags)
    loss = self.loss_module(pred, labels)
    labels = labels.to(torch.int64)
    pred = label_transform.decode_topn(pred, torch.tensor([3] * pred.shape[1]))

```

```

pred = pred.to(torch.int64)
self.metrics.update(pred, labels)
self.log("val_loss", loss, prog_bar=True)

def on_validation_epoch_end(self):
    cp, cr = self.metrics.CP(), self.metrics.CR()
    cf1 = self.metrics.CF1()
    ip, ir = self.metrics.IP(), self.metrics.IR()
    if1 = self.metrics.IF1()
    hf1 = self.metrics.HF1()
    self.log("CP", cp)
    self.log("CR", cr)
    self.log("IP", ip)
    self.log("IR", ir)
    self.log("C_F1", cf1)
    self.log("I_F1", if1)
    self.log("H_F1", hf1, prog_bar=True)
    self.metrics.reset()

```

### Лістинг файлу 5: models/lp.py - модель LP

---

```

import torch
import lightning as lg
from .venn import VCNN
from .mlp import MLP
from models.utils import Metrics, TagTransform

label_transform = TagTransform("./assets/preprocessed/Labels_nus-wide.ndjson")

class LP(lg.LightningModule):

    def __init__(self, lr: float = ..., weight_decay: float = ..., models: dict = None):
        super().__init__()
        self.save_hyperparameters(ignore=["models"])
        if models:
            self.vcnn = models.get("vcnn", VCNN())
            self.mlp = models.get("mlp", MLP())
            self.fc = torch.nn.Sequential(
                torch.nn.Linear(81 * 2, 81),
            )
            self.activation = torch.nn.Sigmoid()
            self.loss_module = torch.nn.BCEWithLogitsLoss()
            self.metrics = Metrics(81)

    def configure_optimizers(self):

```

```

optimizer = torch.optim.AdamW(
    filter(lambda p: p.requires_grad, self.parameters()),
    lr = self.hparams.lr,
    weight_decay = self.hparams.weight_decay,
)
scheduler = torch.optim.lr_scheduler.MultiStepLR(
    optimizer,
    milestones=[5,10],
    gamma=0.1
)
return [optimizer], [{"scheduler": scheduler, "interval": "epoch"}]

def predict(self, x):
    x = self.fc(x)
    x = self.activation(x)
    return x

def forward(self, x):
    image, tags = x
    f_vis = self.vcnn.predict(image)
    f_text = self.mlp.predict(tags)
    f = torch.cat((f_vis, f_text), 1)
    x = self.fc(f)
    return x

def training_step(self, batch, batch_idx):
    (image, tags, labels) = batch
    pred = self.forward((image, tags))
    loss = self.loss_module(pred, labels)
    self.log("train_loss", loss, on_step=True, prog_bar=True)
    return loss

def validation_step(self, batch, batch_idx):
    (image, tags, labels) = batch
    pred = self.forward((image, tags))
    loss = self.loss_module(pred, labels)
    labels = labels.to(torch.int64)
    pred = label_transform.decode_topn(pred, torch.tensor([3] * pred.shape[1]))
    pred = pred.to(torch.int64)
    self.metrics.update(pred, labels)
    self.log("val_loss", loss, prog_bar=True)

def on_validation_epoch_end(self):
    cp, cr = self.metrics.CP(), self.metrics.CR()
    cf1 = self.metrics.CF1()
    ip, ir = self.metrics.IP(), self.metrics.IR()
    if1 = self.metrics.IF1()
    hf1 = self.metrics.HF1()
    self.log("CP", cp)

```

```

self.log("CR", cr)
self.log("IP", ip)
self.log("IR", ir)
self.log("C_F1", cf1)
self.log("I_F1", if1)
self.log("H_F1", hfl, prog_bar=True)
self.metrics.reset()

```

## Лістинг файлу 6: models/lqp.py - модель LQP

```

import torch
import lightning as lg
from .vnn import VCNN
from .mlp import MLP
from .utils import nlabels_f32, nlabels_normalize, nlabels_denormalize

class LQP(lg.LightningModule):

    def __init__(self, lr = 0.01, weight_decay = 0.01, models: dict = None):
        super().__init__()
        self.save_hyperparameters(ignore=["models"])
        if models:
            self.vcnn = models.get("vcnn", VCNN())
            self.mlp = models.get("mlp", MLP())
            self.fc = torch.nn.Sequential(
                torch.nn.Linear(81 * 2, 512),
                torch.nn.ReLU(inplace=True),
                torch.nn.Dropout(),
                torch.nn.Linear(512, 256),
                torch.nn.ReLU(inplace=True),
                torch.nn.Dropout(),
                torch.nn.Linear(256, 1),
            )
            self.loss_module = torch.nn.MSELoss()

    def configure_optimizers(self):
        optimizer = torch.optim.AdamW(
            filter(lambda p: p.requires_grad, self.parameters()),
            lr = self.hparams.lr,
            weight_decay = self.hparams.weight_decay,
        )
        scheduler = torch.optim.lr_scheduler.MultiStepLR(
            optimizer,
            milestones=[5,10],
            gamma=0.1
        )

```

```

)
return [optimizer], [{"scheduler": scheduler, "interval": "epoch"}]

def predict(self, x):
    x = self.fc(x)
    x = nlabels_denormalize(x)
    return x

def forward(self, x):
    image, tags = x
    f_vis = self.vcnn.predict(image)
    f_text = self.mlp.predict(tags)
    f = torch.cat((f_vis, f_text), 1)
    x = self.fc(f)
    return x

def training_step(self, batch, batch_idx):
    (image, tags, labels) = batch
    pred = self.forward((image, tags))
    f_labels = nlabels_f32(labels)
    f_labels = nlabels_normalize(f_labels)
    loss = self.loss_module(pred, f_labels)
    self.log("train_loss", loss, on_step=True, prog_bar=True)
    return loss

def validation_step(self, batch, batch_idx):
    (image, tags, labels) = batch
    pred = self.forward((image, tags))
    f_labels = nlabels_f32(labels)
    f_labels = nlabels_normalize(f_labels)
    loss = self.loss_module(pred, f_labels)
    self.log("val_loss", loss, prog_bar=True)

```

Лістинг файлу 7: models/compose.py - композитна модель

```

import torch
import lightning as lg
from torch import Tensor
from models.lp import LP
from models.mlp import MLP
from models.lqp import LQP
from models.vcnn import VCNN
from .utils import TagTransform, Metrics

```

```
label_transform = TagTransform("./assets/preprocessed/Labels_nus-wide.ndjson")
```

```

class Model(lg.LightningModule):

    def __init__(self, mode: str = "vcnn+mlp+lp+lqp", topk: int = 3):
        super().__init__()
        self.mode = mode
        self.topk = topk
        self.vcnn = VCNN()
        self.mlp = MLP()
        self.lp = LP()
        self.lqp = LQP()
        self.metrics = Metrics(81)

    def test_vcnn(self, x: Tensor, k: int = 3):
        image, _ = x
        pred = self.vcnn.predict(image)
        pred_topn = label_transform.decode_topn(pred, torch.tensor([k] * pred.shape[1]))
        return pred_topn

    def test_vcnn_mlp_lp(self, x: Tensor, k: int = 3):
        image, tags = x
        f_vis = self.vcnn.predict(image)
        f_text = self.mlp.predict(tags)
        f = torch.cat((f_vis, f_text), 1)
        pred = self.lp.predict(f)
        pred_topn = label_transform.decode_topn(pred, torch.tensor([k] * pred.shape[1]))
        return pred_topn

    def test_vcnn_lqp(self, x: Tensor):
        image, _ = x
        f_vis = self.vcnn.predict(image)
        f_text = torch.zeros_like(f_vis)
        f = torch.cat((f_vis, f_text), 1)
        pred = self.lp.predict(f)
        number = self.lqp.predict(f)
        number = number.round().to(torch.int64)
        pred_topn = label_transform.decode_topn(pred, number)
        return pred_topn

    def test_vcnn_mlp_lp_lqp(self, x: Tensor):
        image, tags = x
        f_vis = self.vcnn.predict(image)
        f_text = self.mlp.predict(tags)
        f = torch.cat((f_vis, f_text), 1)
        pred = self.lp.predict(f)
        number = self.lqp.predict(f)
        number = number.round().to(torch.int64)
        pred_topn = label_transform.decode_topn(pred, number)
        return pred_topn

```

```

def forward(self, x):
    match self.mode:
        case "vcnn":
            return self.test_vcnn(x, k=self.topk)
        case "vcnn+lqp":
            return self.test_vcnn_lqp(x)
        case "vcnn+mlp+lp":
            return self.test_vcnn_mlp_lp(x, k=self.topk)
        case "vcnn+mlp+lp+lqp":
            return self.test_vcnn_mlp_lp_lqp(x)

def predict_step(self, batch, batch_idx):
    image, tags, _ = batch
    pred = self.forward((image, tags))
    pred = pred.to(torch.int64)
    return label_transform.decode(pred)

def test_step(self, batch, batch_idx):
    (image, tags, labels) = batch
    pred = self.forward((image, tags))
    pred = pred.to(torch.int64)
    labels = labels.to(torch.int64)
    self.metrics.update(pred, labels)

def on_test_epoch_end(self):
    cp, cr = self.metrics.CP(), self.metrics.CR()
    cf1 = self.metrics.CF1()
    ip, ir = self.metrics.IP(), self.metrics.IR()
    if1 = self.metrics.IF1()
    hf1 = self.metrics.HF1()
    self.log("CP", cp)
    self.log("CR", cr)
    self.log("IP", ip)
    self.log("IR", ir)
    self.log("C_F1", cf1)
    self.log("I_F1", if1)
    self.log("H_F1", hf1, prog_bar=True)
    self.metrics.reset()

```

Лістинг файлу 8: scripts/1ktags.py - скрипт для підготовки лейблів/тегів

---

```
import polars as pl
```

```
FILES = [
    "/home/rinkuro/Sandbox/wallpaper_app/wallpaper_tagging/assets/preprocessed/Train_nus-wide.ndjson",
]
```

```

OTAGS = "./assets/preprocessed/Tags_nus-wide.ndjson"
OLABELS = "./assets/preprocessed/Labels_nus-wide.ndjson"

df = pl.scan_ndjson(FILES)

tags = df.select(pl.col("tags").explode("tags"))
labels = df.select(pl.col("labels").explode("labels"))

tags = tags.select(pl.col("tags").value_counts())
labels = labels.select(pl.col("labels").value_counts())

tags = tags.unnest("tags").rename({"tags": "name"})
labels = labels.unnest("labels").rename({"labels": "name"})

tags = tags.sort(pl.col("count"), descending=True).head(1000)
labels = labels.sort(pl.col("count"), descending=True).head(81)

print(tags.collect())
print(labels.collect())
tags.collect().write_ndjson(OTAGS)
labels.collect().write_ndjson(OLABELS)

```

Лістинг файлу 9: scripts/nuswide2ndjson.py - скрипт для підготовки датасету

```

import glob
import polars as pl
from tqdm import tqdm
from pathlib import Path
from torchvision.io import read_image

# Change these lines
IMAGE_ROOT = "./assets/nus-wide/images"
OUTPUT = "./assets/preprocessed/Test_nus-wide.ndjson"
IMAGES = "./assets/nus-wide/TestImagelist.txt"
TAGS_1K = "./assets/nus-wide/Test_Tags1k.dat"

# Change this line
LABEL_FILES = glob.glob(r"./assets/nus-wide/src/Groundtruth/*_Test.txt")

IMAGE_FILES = glob.glob(r"*.*", root_dir=IMAGE_ROOT)

LABEL_FILES.sort()

header_1k = pl.read_csv("./assets/nus-wide/TagList1k.txt", has_header=False)[["column_1"]].to_list()
header_81 = pl.read_csv("./assets/nus-wide/Concepts81.txt", has_header=False)[["column_1"]].to_list()

```

```

tags_1k = pl.scan_csv(
    TAGS_1K,
    separator="\t",
    has_header=False,
    new_columns=header_1k,
)

labels_81 = pl.concat([
    pl.scan_csv(f, has_header=False, new_columns=[header_81[i]]) for i, f in enumerate(LABEL_FILES),
],
    how = "horizontal",
)

images = pl.scan_csv(
    IMAGES,
    has_header=False,
    new_columns=["file_name"],
)

tags_1k = pl.concat([images, tags_1k], how="horizontal")
tags_1k = (
    tags_1k.melt(
        id_vars="file_name",
        value_vars=header_1k,
        variable_name="tags",
    )
    .filter(pl.col("value") != 0)
    .drop("value")
)
tags_1k = tags_1k.group_by("file_name").agg(pl.col("tags"))

labels_81 = pl.concat([images, labels_81], how="horizontal")
labels_81 = (
    labels_81.melt(
        id_vars="file_name",
        value_vars=header_81,
        variable_name="labels",
    )
    .filter(pl.col("value") != 0)
    .drop("value")
)
labels_81 = labels_81.group_by("file_name").agg(pl.col("labels"))

# Conjoined table
images = labels_81.join(tags_1k, on="file_name")

```

```

# Add root path
images = images.with_columns(pl.lit(IMAGE_ROOT).alias("image_root"))

# Before file check up
print(images.collect())

# Checks image existance
images = images.filter(pl.col("file_name").is_in(IMAGE_FILES))

# Checks if images are not corrupted (by trying to read)
dflen = images.select(pl.count("file_name")).collect().item()
print("Starting check for image corruption")
dellist = []
for row in tqdm(
    images.select([pl.col("image_root"), pl.col("file_name")]).collect().to_dicts()):
    path = Path(row["image_root"], row["file_name"])
    try:
        read_image(str(path))
    except RuntimeError:
        dellist.append(row["file_name"])

images = images.filter(~pl.col("file_name").is_in(dellist))

# After file checkup
print(images.collect())

```

images.collect().write\_ndjson(OUTPUT)

## Листинг файлу 10: scripts/train/vcnn.py - скрипт для тренування VCNN

---

```

import lightning as lg
from pathlib import Path
from data import DataModule
from models.vcnn import VCNN
from lightning.pytorch.loggers import CSVLogger
from lightning.pytorch.callbacks import ModelCheckpoint, LearningRateMonitor, ModelSummary

```

```

TRAINED_MODELS = Path("./assets/trained_models")
ROOT_DIR = TRAINED_MODELS / "vcnn.train"
CKPT_PATH = TRAINED_MODELS / "vcnn.train" / "vcnn.ckpt"

data = DataModule(batch_size = 32, prefetch_factor = 16, num_workers = 6)

trainer = lg.Trainer(

```

```

devices = 1,
max_epochs = 30,
accelerator = "gpu",
precision = "bf16-mixed",
default_root_dir = ROOT_DIR,
logger = CSVLogger(ROOT_DIR, "logs", version=0),
limit_train_batches = 0.2,
limit_val_batches = 0.2,
callbacks = [
    ModelSummary(2),
    LearningRateMonitor(logging_interval = "step"),
    ModelCheckpoint(
        monitor="H_F1",
        mode="max",
        save_weights_only=True,
        save_top_k=-1,
        dirpath=ROOT_DIR / "checkpoints",
        save_on_train_epoch_end=True,
        filename="{H_F1:.5f}@{v_num}@{epoch}@{val_loss:.3f}",
    ),
],
)
)

model = VCNN(
    lr = 0.001,
    weight_decay = 0.01,
)
trainer.fit(model, datamodule=data)

```

Лістинг файлу 11: scripts/train/mlp.py - скрипт для тренування MLP

```

import lightning as lg
from data import DataModule
from pathlib import Path
from models.mlp import MLP
from lightning.pytorch.loggers import CSVLogger
from lightning.pytorch.callbacks import ModelCheckpoint, ModelSummary, LearningRateMonitor

```

```

TRAINED_MODELS = Path("./assets/trained_models")
ROOT_DIR = TRAINED_MODELS / "mlp.train"
CKPT_PATH = TRAINED_MODELS / "mlp.train" / "mlp.ckpt"

```

```

data = DataModule(load_images = False, batch_size = 32, prefetch_factor = 4, num_workers = 6)

trainer = lg.Trainer(

```

```

devices=1,
max_epochs=30,
accelerator="gpu",
precision = "bf16-mixed",
default_root_dir = ROOT_DIR,
logger=CSVLogger(ROOT_DIR, "logs", version=0),
callbacks=[

    ModelSummary(2),
    LearningRateMonitor(logging_interval = "step"),
    ModelCheckpoint(
        monitor="H_F1",
        mode="max",
        save_weights_only=True,
        save_top_k=-1,
        dirpath=ROOT_DIR / "checkpoints",
        save_on_train_epoch_end=True,
        filename="{H_F1:.5f}@{v_num}@{epoch}@{val_loss:.3f}",
    ),
],
)
)

model = MLP(
    lr = 0.0001,
    weight_decay=0.01,
)

trainer.fit(model, datamodule=data)

```

Лістинг файлу 12: scripts/train/lp.py - скрипт для тренування LP

```

import lightning as lg
from data import DataModule
from pathlib import Path
from models.lp import LP
from models.vcnn import VCNN
from models.mlp import MLP
from lightning.pytorch.loggers import CSVLogger
from lightning.pytorch.callbacks import ModelCheckpoint, ModelSummary, LearningRateMonitor

```

```

TRAINED_MODELS = Path("./assets/trained_models")
ROOT_DIR = TRAINED_MODELS / "lp.train"
CKPT_PATH = TRAINED_MODELS / "lp.train" / "lp.ckpt"

```

```

data = DataModule(batch_size = 32, prefetch_factor = 8, num_workers = 6)

trainer = lg.Trainer(

```

```

devices=1,
max_epochs=30,
accelerator="gpu",
precision = "bf16-mixed",
default_root_dir = ROOT_DIR,
logger=CSVLogger(ROOT_DIR, "logs", version=0),
limit_train_batches = 0.2,
limit_val_batches = 0.2,
callbacks=[

    ModelSummary(2),
    LearningRateMonitor(logging_interval = "step"),
    ModelCheckpoint(
        monitor="H_F1",
        mode="max",
        save_weights_only=True,
        save_top_k=-1,
        dirpath=ROOT_DIR / "checkpoints",
        save_on_train_epoch_end=True,
        filename="{H_F1:.3f}@{v_num}@{epoch}@{val_loss:.3f}",
    ),
],
)
)

vcnn = VCNN.load_from_checkpoint("./assets/trained_models/vcnn.train/vcnn.ckpt")
vcnn.freeze()
mlp = MLP.load_from_checkpoint("./assets/trained_models/mlp.train/mlp.ckpt")
mlp.freeze()

model = LP(
    lr = 0.01,
    weight_decay=0.0003,
    models = {"vcnn": vcnn, "mlp": mlp}
)
trainer.fit(model, datamodule=data)

```

Лістинг файлу 13: scripts/train/lqp.py - скрипт для тренування LQP

```

import lightning as lg
from data import DataModule
from pathlib import Path
from models.mlp import MLP
from models.vcnn import VCNN
from models.lqp import LQP
from lightning.pytorch.loggers import CSVLogger
from lightning.pytorch.callbacks import ModelCheckpoint, ModelSummary, LearningRateMonitor

```

```

TRAINED_MODELS = Path("./assets/trained_models")
ROOT_DIR = TRAINED_MODELS / "lqp.train"

data = DataModule(batch_size=32, prefetch_factor=8, num_workers=6)

trainer = lg.Trainer(
    devices=1,
    max_epochs=30,
    accelerator="gpu",
    default_root_dir=ROOT_DIR,
    logger=CSVLogger(ROOT_DIR, "logs", version=0),
    limit_train_batches=0.2,
    limit_val_batches=0.2,
    callbacks=[
        ModelSummary(2),
        LearningRateMonitor(logging_interval="step"),
        ModelCheckpoint(
            monitor="val_loss",
            mode="min",
            save_weights_only=True,
            save_top_k=-1,
            dirpath=ROOT_DIR / "checkpoints",
            save_on_train_epoch_end=True,
            filename="{v_num}@{epoch}@{val_loss:.5f}",
        ),
    ],
)
)

vcnn = VCNN.load_from_checkpoint("./assets/trained_models/vcnn.train/vcnn.ckpt")
vcnn.freeze()
mlp = MLP.load_from_checkpoint("./assets/trained_models/mlp.train/mlp.ckpt")
mlp.freeze()

model = LQP(
    lr=0.0001,
    weight_decay=0.0003,
    models={"vcnn": vcnn, "mlp": mlp}
)
)

trainer.fit(model, datamodule=data)

```

Лістинг файлу 14: scripts/compose2safe.py - скрипт для конвертації вагових коефіцієнтів у результатуючу модель

---

```

from pathlib import Path
from models.lp import LP
from models.mlp import MLP

```

```

from models.lqp import LQP
from models.vcnn import VCNN
from models.compose import Model
from safetensors.torch import save_file

MODEL_ROOT = Path("./assets/trained_models")

vcnn = VCNN.load_from_checkpoint(MODEL_ROOT / "vcnn.train" / "vcnn.ckpt")
mlp = MLP.load_from_checkpoint(MODEL_ROOT / "mlp.train" / "mlp.ckpt")
lp = LP.load_from_checkpoint(MODEL_ROOT / "lp.train" / "lp.ckpt", strict=False)
lqp = LQP.load_from_checkpoint(MODEL_ROOT / "lqp.train" / "lqp.ckpt", strict=False)

model = Model(
    models = {
        "vcnn": vcnn,
        "mlp": mlp,
        "lp": lp,
        "lqp": lqp,
    }
)

ignore = ["lp.mlp", "lp.vcnn", "lqp.mlp", "lqp.vcnn"]
state = model.state_dict()

to_del = []
for k in state.keys():
    for ik in ignore:
        if ik in k:
            to_del.append(k)

for k in to_del:
    state.pop(k)

save_file(state, MODEL_ROOT / "compose.test" / "compose.safetensors")

```

Лістинг файлу 15: scripts/test.py - скрипт для тестування роботи композитної моделі

---

```

import os
import torch
import lightning as lg
from pathlib import Path
from data import DataModule
from models.compose import Model
from safetensors.torch import load_model
from lightning.pytorch.loggers import CSVLogger

```

```

MODEL_ROOT = Path("./assets/trained_models")
ROOT_DIR = MODEL_ROOT / "compose.test"

data = DataModule(batch_size=32, num_workers=6)
trainer = lg.Trainer(
    devices=1,
    accelerator="gpu",
    limit_test_batches=0.1,
    default_root_dir=ROOT_DIR,
    logger=CSVLogger(ROOT_DIR, "logs", version=0),
)

# Choose at most N tags to use
# os.environ["RANDOM_NTAGS"] = "10"

def compose_model(**kwargs):
    model = Model(**kwargs)
    load_model(model, ROOT_DIR / "compose.safetensors", strict=False)
    model.freeze()
    return model

if __name__ == "__main__":
    model = compose_model()
    with torch.no_grad():
        trainer.test(model, datamodule=data)

```

## Додаток Б

## Додаткові приклади

Image	Truth	Model pred
	animal birds sky	animal birds sky
	person wedding	person wedding
	buildings	sky
	animal	animal birds

Рисунок Б.1

Image	Truth	Model pred
	animal clouds	animal
	animal	animal
	flowers	flowers
	animal elk snow	animal snow

Рисунок Б.2

Image	Truth	Model pred
	sky	animal clouds horses sky
	train	railroad train
	buildings clouds sky window	buildings clouds sky town window
	clouds grass sky vehicle	clouds sky vehicle window

Рисунок Б.3

Image	Truth	Model pred
	clouds sky	buildings clouds grass sky
	buildings plants	buildings grass sky
	person	person
	animal dog	animal dog

Рисунок Б.4

Image	Truth	Model pred
	person	person window
	grass road vehicle	road vehicle
	lake ocean water	beach lake ocean water
	flowers plants	flowers

Рисунок Б.5

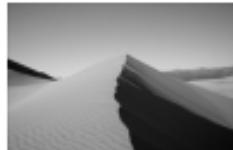
Image	Truth	Model pred
	snow	snow
	animal cat	animal cat
	animal elk lake water	animal lake water
	road sand	sand sky

Рисунок Б.6

Image	Truth	Model pred
	plane clouds	plane sky
	flowers garden grass mountain plants sky	clouds grass plants road sky
	animal beach	beach
	grass house	grass plants sky tree

Рисунок Б.7

Image	Truth	Model pred
	flowers plants	flowers plants
	animal	animal
	buildings clouds sky snow tower	buildings clouds sky
	animal bear snow	animal

Рисунок Б.8

Image	Truth	Model pred
	buildings	buildings town water
	clouds sky sunset	sky sunset
	person	person
	sky	clouds sky

Рисунок Б.9

Image	Truth	Model pred
	sky	grass sky
	road train window	sky train
	animal grass	animal grass
	water	person

Рисунок Б.10

Image	Truth	Model pred
	animal whales	animal fish
	beach clouds sky	clouds sky
	buildings	window
	grass house sky	buildings grass house

Рисунок Б.11

Image	Truth	Model pred
	beach clouds lake sky sunset beach water clouds ocean sky sunset water clouds sky sunset	clouds ocean sky sunset <del>water</del> clouds lake ocean sky sunset water clouds fire sky sunset
		
		
	flowers sky	flowers sky

Рисунок Б.12

Image	Truth	Model pred
	animal	animal
	person	person
	airport grass plane	airport clouds plane sky
	buildings clouds tower	sky tower

Рисунок Б.13

## Додаток В

## Ілюстративний матеріал

# Математичне та програмне забезпечення для автоматичного тегування зображень

•••

Виконав:  
Студент групи  
КМ-01  
Скороденко Д. О.

Науковий керівник:  
Доцент кафедри  
ПМА  
Сирота С. В.

Рисунок В.1 – Слайд 1

## Вибір теми. Актуальність.

В сучасному світі, коли людство виробляє все більше і більше даних на добу через стрімку цифровізацію, виникає необхідність структурувати ці дані для того щоб в подальшому мати змогу здійснювати якісний пошук по певним категоріям. Це, зокрема, справедливо для зображень. Для їх структуризації можна використовувати лейблі, а для автоматизації процесу присвоєння лейблів зображеню потрібно використовувати систему для маркування.

Існуючі рішення для маркування зображення поділяються на дві категорії:

- Прості нейронні мережі. Такі рішення не дуже виагливі до обчислювальних потужностей, однак не надають високої якості маркування.
- Складні нейронні мережі (композиція нейронних мереж). Такі рішення виагливі до обчислювальних потужностей, потребують значно більше часу на тренування, але значно якісніше маркують зображення.

Дана робота націлена на вирішення цих проблем, а саме проектування моделі для маркування зображення, яка має просту архітектуру, швидко тренується та якісно маркує зображення (шпалери робочого столу) із використанням тегів.

Рисунок В.2 – Слайд 2

## Постановка задачі

**Об'єктом дослідження** є маркування зображень, та методи покращення маркування зображення. Для порівняння ефективності маркування серед існуючих рішень було обрано моделі, для яких яких обраховані тестові метрики для того ж датасету, який обрано в даній роботі.

**Предметом дослідження** є множина шпалерів робочого столу в якості основної модальності даних, та додаткова інформація (надані людьми шумні теги) в якості додаткової.

**Метою роботи** є розробка ПЗ для маркування зображень (шпалерів робочого столу) для покращення системи категоріального пошуку зображень (шпалерів робочого столу).

**Кінцевим результатом** даної роботи є математичне та програмне забезпечення, архітектура моделі, вагові коефіцієнти натренованої моделі та код програмного забезпечення, в якому реалізовано дану роботу.

Рисунок В.3 – Слайд 3

## Завдання для досягнення мети

- Провести аналіз існуючих рішень
- Змоделювати композитну нейронну мережу
- Провести тренування
- Провести аналіз ефективності компонентів моделі
- Провести детальний аналіз впливу тегів на якість маркування
- Провести порівняльний аналіз якості і повноти опису розглянутої моделі відносно існуючих рішень на основі тестових метрик
- Провести аналіз декількох ілюстративних прикладів роботи системи

Рисунок В.4 – Слайд 4

## Постановка задачі маркування зображень



Рисунок В.5 – Слайд 5

## Огляд існуючих рішень

Рисунок В.6 – Слайд 6

### Базове рішення (аналіз зображення)

Ключовим аспектом задачі маркування є аналіз основної модальності даних - зображення. У абсолютній більшості робіт використовується модель CNN (Convolutional Neural Network). Щодо конкретних архітектур, то найчастіше використовується Resnet, більш рідкісним є використання ResNext, AlexNet та GoogleNet.

Найновіші підходи використовують ViT (Visual Transformer), які надають кращі результати, однак вимагають значно більше часу для тренування і потребують значного об'єму тренувальних даних.

Рисунок В.7 – Слайд 7

## Додаткова модальність даних

Більш нові роботи також розглядають додаткові джерела інформації для підвищення якості маркування. Існує два основних підходи:

- 1) **Аналіз додаткової інформації.** Даний підхід аналізує додаткову до зображення інформацію. Це може бути як текстова інформація (теги/анотації), так і метадані, геолокації, тощо. Очевидним недоліком даного методу є потреба у цій додатковій інформації, яку можуть мати не всі зображення, а відсутність даної інформації знижує точність результатуючого маркування.
- 2) **Аналіз цільових класів.** На відміну від загальної інтерпретації класів для задачі маркування (коли кожен клас - незалежна сутність), даний підхід аналізує зв'язки між цільовими класами, створюючи нову модальність на основі набору цільових класів. Для цього зазвичай використовуються embeddings (та трансформери). Перевагою даного підходу є те, що йому не потрібні ніякі додаткові дані окрім зображень. Основним недоліком таких систем є висока складність, і як наслідок - значно довший процес тренування / розпізнавання.

Рисунок В.8 – Слайд 8

## Аналіз цільових класів. Resnet-CPSD

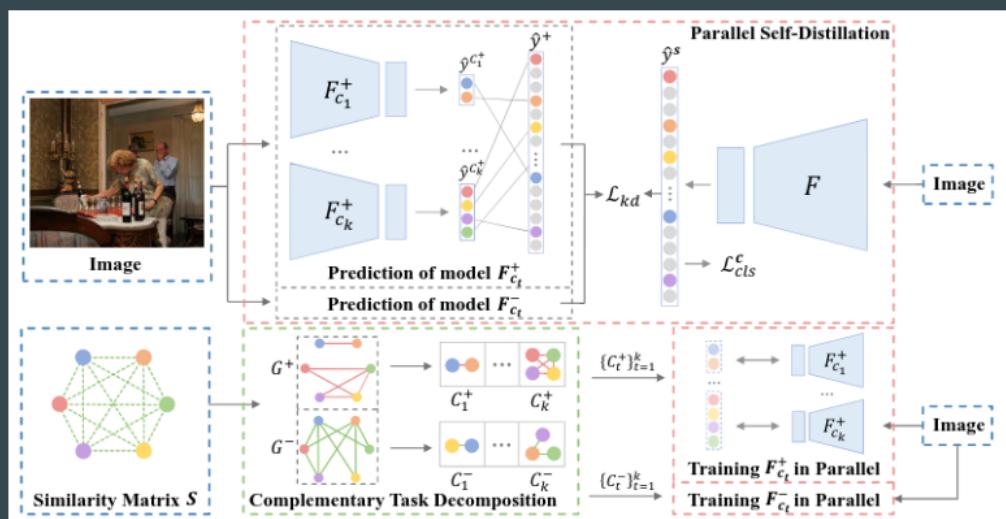


Рисунок В.9 – Слайд 9

## Кількість лейблів

Результатом роботи будь якої класифікаційної моделі є вектор ймовірностей, який репрезентує приналежність об'єкту до певних цільових класів. Для перетворення вектору в результат класифікації використовується індикаторна функція. Для задачі класифікації вибір результату на основі цього вектора очевидний - клас із найбільшою ймовірністю, однак для задачі маркування все складніше. Існує як мінімум дві основні індикаторні функції: top k та threshold a.

Рисунок В.10 – Слайд 10

## Індикаторна функція top k

Image				
Truth	clouds grass house sky	leaf plants sky	animal grass	person
Top 5 pred	clouds grass house road sky	clouds grass leaf plants sky	animal grass horses plants sky	animal clouds person road sky
Model pred	clouds grass sky	plants sky	animal grass horses	person

Top k, або “найкращих k”, обирає результат як k найбільш ймовірних класів у векторі ймовірностей.

Більшість робіт, які віднесено до множини “існуючих рішень” використовує індикаторну функцію top 3. Очевидно що така кількість лейблів не є оптимальною, так як деякі зображення містять більше 3 тегів, а деякі - менше.

Рисунок В.11 – Слайд 11

## Індикаторна функція threshold a

Threshold a, або порогове значення a, - це індикаторна функція яка маркує зображення за пороговим значенням, так для класу який має ймовірність більше ніж a маркування - позитивне, і навпаки. Перевагою даного методу є "вбудована" адаптивність до динамічної кількості лейблів, однак суттєвим недоліком є те, що вектор ймовірностей має бути досить сильно дискретизованим. Тобто для позитивних класів ймовірність має бути високою (0.6 - 1.0), а для негативних класів низькою (0.0 - 0.4).

Рисунок В.12 – Слайд 12

## Моделювання

Рисунок В.13 – Слайд 13

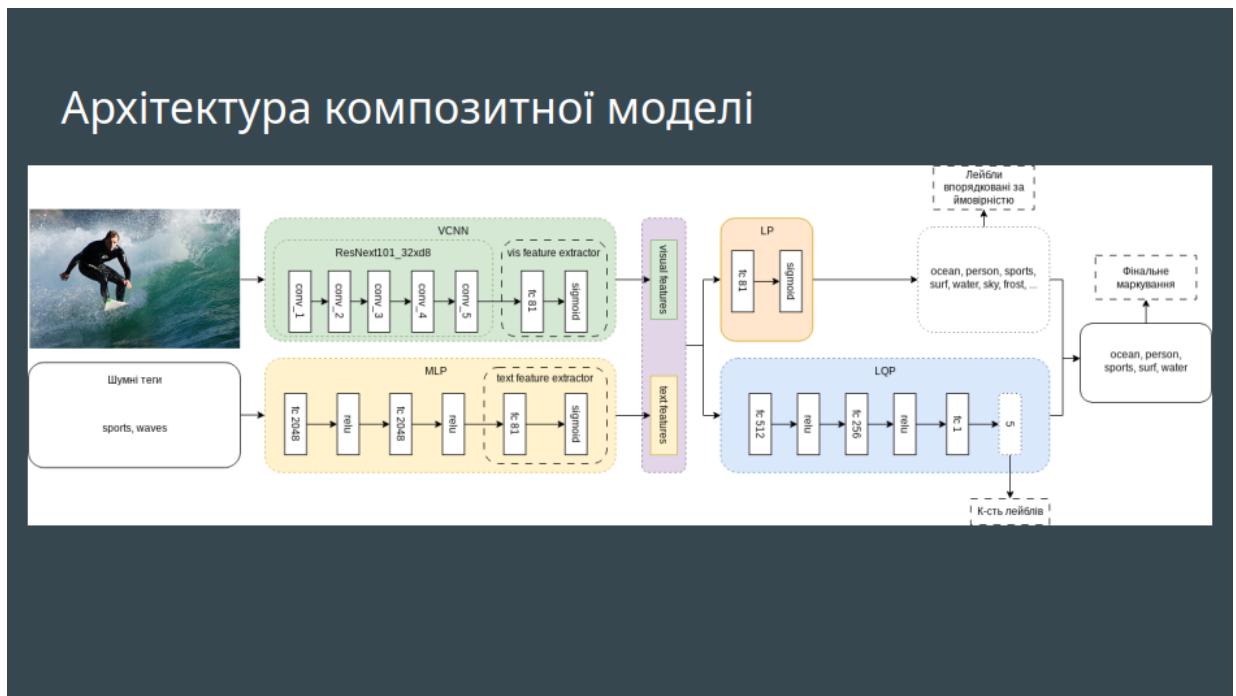


Рисунок В.14 – Слайд 14

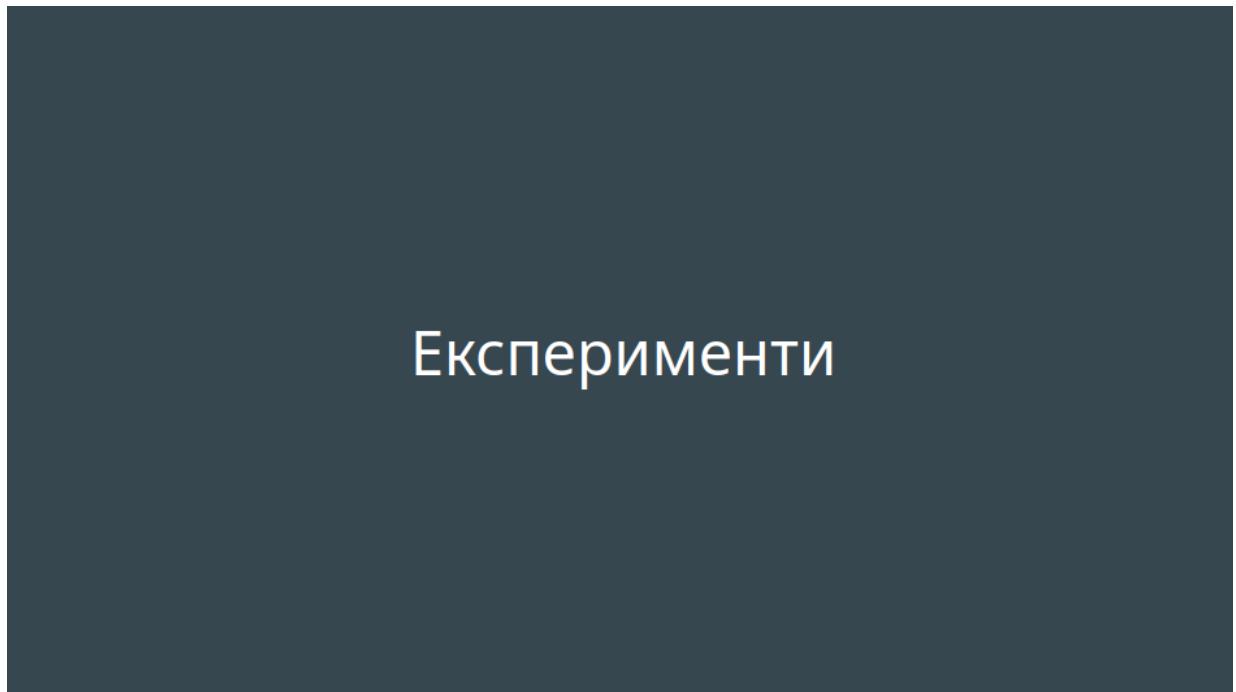


Рисунок В.15 – Слайд 15

## Датасет

Один із найбільш часто використовуваних датасетів для тестування моделей маркування зображень - NUS-WIDE, він складається із 269,655 зображень, 81 цільового класу (лейблу), та  $\geq 5000$  тегів. Для проведення тренування/тестування використовується розподіл, наведений авторами датасету, так як він є збалансованим відносно кількості лейблів.

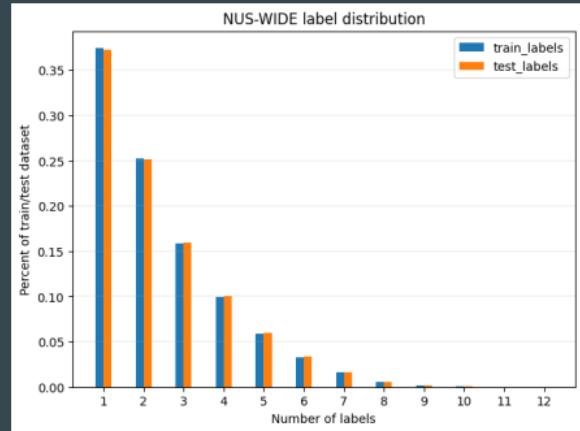


Рисунок В.16 – Слайд 16

## Датасет. Підготовка

Обраний датасет містить посилання на зображення на ресурсі Flickr, і деякої частини цих зображень вже не існує. Додатково буде використано тільки 1000 найбільш частих тегів з 5000, при чому зображення які не містять жодного тега - відфільтровано.

	Тренування	Тестування
Кількість зображень	121962	81636
Середня к-сть лейблів	2.42	2.43
Медіана к-сть лейблів	2	2
Мінімальна к-сть лейблів	1	1
Максимальна к-сть лейблів	12	13
Середня к-сть тегів	6.27	6.26
Медіана к-сть тегів	5	5
Мінімальна к-сть тегів	1	1
Максимальна к-сть тегів	131	125

Табл. 4.1 – Характеристики тренувальної/тестової вибірок

Рисунок В.17 – Слайд 17

## Датасет. Цільові класи

grass cityscape  
earthquake temple nighttime  
sports cow leaf  
**clouds** tiger waterfall  
zebra tower moon beach  
plane sunset street surf  
book statue sun birds cat  
whale elk railroad rocks  
boats police coral mountain  
flags town snow sign wedding castle  
bridge fish cars vehicle toy glacier  
**person**  
plants frost fox ocean  
soccer train house reflection harbor  
valley lake dancing protest computer  
horses sand garden tattoo  
flowers dog swimmers  
**sky**  
**buildings**  
**animal**  
**trees**

Рисунок В.18 – Слайд 18

## Тестові метрики

$$\begin{aligned} \text{C-P} &= \frac{1}{C} \sum_{j=1}^C \frac{NI_j^c}{NI_j^p} & \text{O-P} &= \frac{\sum_{i=1}^N NL_i^c}{\sum_{i=1}^N NL_i^p} \\ \text{C-R} &= \frac{1}{C} \sum_{j=1}^C \frac{NI_j^c}{NI_j^g} & \text{O-R} &= \frac{\sum_{i=1}^N NL_i^c}{\sum_{i=1}^N NL_i^g} \\ \text{C-F1} &= \frac{2 \cdot \text{C-P} \cdot \text{C-R}}{\text{C-P} + \text{C-R}} & \text{O-F1} &= \frac{2 \cdot \text{O-P} \cdot \text{O-R}}{\text{O-P} + \text{O-R}} \end{aligned}$$

Рисунок В.19 – Слайд 19

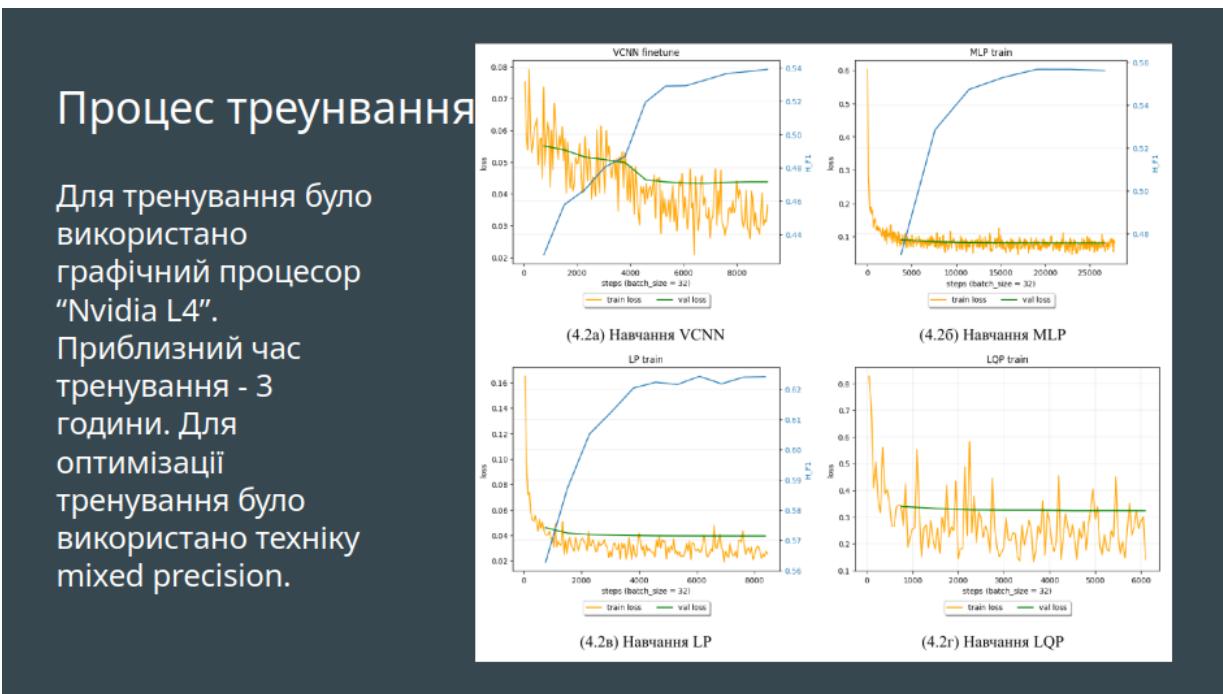


Рисунок В.20 – Слайд 20

## Аналіз результатів

Рисунок В.21 – Слайд 21

## Аналіз компонентів нейронної мережі

Модель	Індикаторна ф-ція (3.6)	Модальності	C-P	C-R	C-F1	O-P	O-R	O-F1	H-F1
Композитна	top k	Зображення+теги	72.49	59.51	65.36	76.53	74.41	75.46	70.04
VCNN+MLP+LP	top 3	Зображення+теги	60.51	61.28	60.89	67.87	71.52	63.98	62.40
VCNN+LQP	top k	Зображення	64.62	37.61	47.54	77.07	59.01	66.84	55.56
VCNN	top 3	Зображення	44.48	53.32	48.50	55.44	68.52	61.29	54.15

Табл. 5.1 – Порівняння компонентів моделі

Рисунок В.22 – Слайд 22

## Аналіз впливу максимальної кількості тегів на маркування

З наведеної таблиці можна зробити висновок, що між якістю маркування та кількістю тегів є пряма залежність. В середньому приріст точності відносно метрики H-F1 становить 1 - 2%. Також варто відзначити, що після проходження медіани (5) приріст спадає до менше ніж 0.5%.

Таким чином для того щоб досягти оптимальної якості із найменшою кількістю додаткових тегів потрібно розглядати від 2 до 5 тегів.

Максимальна кількість тегів	C-P	C-R	C-F1	O-P	O-R	O-F1	H-F1
0	44.48	53.32	48.50	55.44	68.52	61.29	54.15
1	66.42	37.28	47.75	74.80	66.78	70.56	56.96
2	70.20	43.00	53.33	75.50	68.75	71.97	61.26
3	71.06	46.97	56.55	75.70	69.85	72.66	63.60
4	71.11	49.80	58.57	75.95	71.00	73.39	65.15
5	71.99	52.62	60.80	76.19	71.95	74.01	66.76
6	71.92	53.37	61.49	76.25	72.44	74.30	67.29
7	71.86	54.65	62.08	76.35	72.85	74.56	67.75
8	72.25	56.03	63.12	76.49	73.26	74.84	68.48
9	72.07	56.55	63.37	76.57	73.62	75.07	68.73
10	72.33	57.13	63.84	76.59	73.83	75.18	69.05
-	72.49	59.51	65.36	76.53	74.41	75.46	70.04

Табл. 5.2 – Порівняння впливу кількості тегів на маркування

Рисунок В.23 – Слайд 23

## Порівняння з існуючими рішеннями

Модель	Індикаторна ф-ція (3.6)	Модальност	C-P	C-R	C-F1	O-P	O-R	O-F1	H-F1
Композитна	top k	Зображення+теги	72.49	59.51	65.36	76.53	74.41	75.56	70.04
Query2Label [11]	threshold $\alpha$	Зображення (+аналіз класів)	-	-	67.60	-	-	76.3	71.69
SR-CNN-RNN [10]	top 3	Зображення+теги	71.73	61.73	66.36	77.41	76.88	77.15	71.35
Resnet-CPSD [20]	threshold $\alpha$	Зображення (+аналіз класів)	-	-	64.00	-	-	75.30	69.19
MS-CMA [21]	threshold $\alpha$	Зображення (+аналіз класів)	-	-	60.50	-	-	73.80	66.49
Resnet-SRN [22]	threshold 0.5	Зображення (+аналіз класів)	65.20	55.80	58.50	75.50	71.50	73.40	65.10
SINN [7]	top 3	Зображення+теги	58.30	60.63	59.44	57.05	79.12	66.29	62.68
TagNeighbour [8]	top 3	Зображення+метадані	54.74	57.30	55.99	53.46	75.10	62.46	59.05
CNN+Logistic [7]	top 3	Зображення	45.60	45.03	45.31	51.32	70.77	59.50	51.44
CNN-RNN [18]	top 3	Зображення	40.50	30.40	34.70	49.9	61.70	55.20	42.61
CNN+WARP [5]	top 3	Зображення	31.65	35.60	33.51	48.59	60.49	53.89	41.32
CNN+Softmax [5]	top 3	Зображення	31.68	31.22	31.45	47.82	59.52	53.03	39.48

Табл. 5.2 – Порівняння результатуючих метрик для різних моделей на датасеті  
NUS-WIDE

Рисунок В.24 – Слайд 24

## Демонстаривний приклад №1

Image	Truth	Model pred
	animal birds sky	animal birds sky
	person wedding	person wedding
	buildings	sky
	animal	animal birds

Рисунок В.25 – Слайд 25

## Демонстаривний приклад №2

Image	Truth	Model pred
	sky	animal clouds horses sky
	train	railroad train
	buildings clouds sky window	buildings clouds sky town window
	clouds grass sky vehicle	clouds sky vehicle window

Рисунок В.26 – Слайд 26

## Демонстаривний приклад №3

Image	Truth	Model pred
	sky	grass sky
	road train window	sky train
	animal grass	animal grass
	water	person

Рисунок В.27 – Слайд 27

## Демонстаривний приклад №4

Image	Truth	Model pred
	person	person window
	grass road vehicle	road vehicle
	lake ocean water	beach lake ocean water
	flowers plants	flowers

Рисунок В.28 – Слайд 28

## Висновки

Рисунок В.29 – Слайд 29

Досягнуто мету - розроблено ПЗ для маркування зображень (шпалерів робочого столу), а саме спроектовано та натреновано композитну нейронну мережу для автоматичного тегування зображень на основі даних із двома модальностями (зображення і теги), що дозволить покращити категоріальний пошук при впровадженні даного рішення.

Після завершення процесу тренування було досягнуто точності співставної із існуючими рішеннями по якості маркування, при меншій складності результиуючої нейронної мережі, що внаслідок зменшує вимоги як до ресурсів необхідних, щоб натренувати модель, так і до ресурсів необхідних для запуску моделі.

Проведено порівняльний аналіз компонентів моделі, та доведено ефективність кожного з її компонентів. Було проаналізовано вплив кількості тегів на результиуючу якість тегування та обрано кількість тегів для оптимальної якості, а саме від 2 до 5 тегів. Також проведено порівняння із існуючими рішеннями, що дає можливість зробити висновок, що розроблена модель є досить ефективним рішенням для маркування зображень (шпалерів робочого столу).

Рисунок В.30 – Слайд 30