Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

# ПОБУДОВА СИСТЕМИ ПЕРЕВІРКИ ВІДПОВІДНОСТІ ЗОБРАЖЕНЬ ТА ТЕКСТУ

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” – 121

Керівник курсової роботи
д. т. н., доц. Глибовець А. М.
_____
*(підпис)*
“____” _____ 2022 р.
Виконав студент ІПЗ-3
Сітьков І. П.
“____” _____ 2022 р.

Київ 2022

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ
Зав. кафедри мережних технологій,
проф., д.ф.-м.н.
_____ Г. І. Малашонок
*(підпис)*
“____”_____2022 р.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

на курсову роботу

студенту Сітькову Іллі Павловичу факультету інформатики 3-го курсу

ТЕМА: Побудова системи перевірки відповідності зображень та тексту

Зміст ТЧ до курсової роботи:

1. Індивідуальне завдання
2. Календарний план
3. Анотація
4. Вступ
5. Аналіз наявних рішень для розпізнавання об'єктів на зображенні та в тексті
6. Дослідження способів семантичного порівняння слів
7. Розробка рішень для визначення відповідності між зображенням та текстом
8. Тестування розроблених рішень, аналіз отриманих результатів
9. Висновки
10. Список літератури.

Дата видачі “___” _____ 2020 р.
Керівник _____
*(підпис)*
Завдання отримав _____
*(підпис)*

Тема: Побудова системи перевірки відповідності зображень та тексту

Календарний план виконання роботи:

| No п/п | Назва етапу курсової роботи | Термін виконання етапу | Примітка |
|---|---|---|---|
| 1. | Отримання завдання на курсову роботу. | 12.10.2021 | |
| 2. | Аналіз доступних алгоритмів для розпізнавання об'єктів на зображенні та в тексті, семантичного порівняння слів. | 18.01.2022 | |
| 3. | Розробка програмних рішень для визначення відповідності між зображенням та текстом. | 03.03.2022 | |
| 4. | Вибір метрик для тестування, створення тестової колекції. | 25.03.2022 | |
| 5. | Тестування розроблених алгоритмів, аналіз результатів, визначення переваг та обмежень розроблених рішень | 10.04.2022 | |
| 6. | Написання курсової роботи. | 15.05.2022 | |
| 7. | Коригування виконаної роботи. | 20.05.2022 | |
| 8. | Остаточне оформлення роботи. | 06.06.2022 | |
| 9. | Захист курсової роботи. | 15.06.2022 | |

Студент Сітьков І.П.
Керівник Глибовець А. М.
“____”_____ 2022 р.

ЗМІСТ

# АНОТАЦІЯ

У пропонованій курсовій роботі досліджено алгоритми, призначені для розпізнавання об'єктів на зображенні та в тексті, векторного представлення слів, порівняння семантики лексичних одиниць. На підставі результатів аналізу здійснено розробку підходів до визначення відповідності між зображенням та текстом. Ефективність запропонованих способів перевірено під час тестування. Обґрунтовано доречність застосування кожного методу для роботи з різними видами описів до зображень, що дозволило зробити висновки про найефективніший алгоритм, окреслити подальші шляхи його вдосконалення з урахуванням наведених у роботі обмежень та переваг.

Ключові слова: алгоритм, подібність зображень і тексту, семантична подібність, порівняння на основі об'єктів, порівняння з урахуванням площі, порівняння без урахування площі, порівняння на основі векторів речень, синусна міра схожості, векторне представлення, машинне навчання, іменована сутність, середня точність, середньоквадратичне відхилення.

# ВСТУП

Найважливішим ресурсом у сучасному світі є інформація, яка представлена в різних форматах: візуальному, текстовому, звуковому. Найчастіше використовуються перші два, що пов’язано з легкістю та зручністю їхнього сприйняття. Саме тому текстову та візуальну інформацію використовують в усіх сферах життя. Однак щоб задовольнити свої потреби, автори контенту часом вдаються до маніпуляцій. Це й породжує потребу перевірки відповідності зображень текстовому опису до них. Небезпеку для споживача інформації можуть становити такі випадки:

1) намагання видати один об’єкт за інший з метою збільшення показників відвідуваності сайту через використання підпису, який не відповідає зображенню

2) спроба уникнути відповідальності за поширення забороненого контенту шляхом підробки текстового підпису до зображення;

3) формування викривленої громадської думки з метою розпалювання ворожнечі через використання зображень, які не відповідають подіям, місцям, об’єктам, описаним у тексті.

Актуальність обраної теми полягає у важливості пошуку рішень для розв’язання вищенаведених проблем, новому застосуванні алгоритмів із різних галузей машинного навчання, яке активно розвивається.

У пропонованій нижче роботі буде приділено увагу розробці системи, що дозволить розв’язати перші дві проблеми із зазначених раніше. Остання ж є досить широкою, тому не може вміститися в межах короткого дослідження й потребує зусиль не лише розробників, а й представників інших спеціальностей (журналістів, політиків тощо).

Наші програмні рішення дозволять визначати, чи певний текст відповідає зображенню, а також давати кількісну характеристику подібності між картинкою та її описом.

Робота містить три розділи.

У першому розділі проаналізовано готові рішення для розпізнавання об'єктів на зображенні та в тексті, а також способи порівняння семантики визначених об'єктів; розглянуто можливості використання алгоритмів для роботи з текстами, написаними українською мовою.

У другому розділі запропоновано конкретні алгоритми для розпізнавання об'єктів на зображенні та в тексті, способи векторного представлення слів та шляхи застосування в нашій розробці; наведено міркування щодо можливих методів визначення подібності між зображенням та текстом, додано реалізацію описаних підходів.

Третій розділ присвячено тестуванню розглянутих способів для визначення відповідності між зображенням та текстом з метою оцінювання точності програмних рішень. У ході перевірки алгоритмів визначено найефективніші підходи, обґрунтовано доречність їхніх застосувань для роботи з різними видами текстових описів. Окрім цього, проаналізовано переваги та недоліки розроблених рішень.

Об’єктом дослідження є способи визначення подібності між зображенням та текстом.

Предметом дослідження є ефективність способу обчислення подібності через порівняння семантики об’єктів, які містяться на зображенні та в тексті.

Метою курсової роботи є розробка системи для перевірки відповідності зображень тексту.

Для цього мають бути розв’язані такі завдання:

1) проаналізувати готові рішення для розпізнавання об’єктів на зображенні та в тексті, дослідити способи порівняння семантики об’єктів у тексті та на зображенні;

2) визначити конкретні реалізації готових алгоритмів для застосування під час написання власного;

3) побудувати алгоритм для встановлення відповідності між текстом та зображенням;

4) перевірити роботу алгоритму, обґрунтувати результати тестування, на їхній основі зробити висновок про найефективніший алгоритм, визначити переваги та недоліки розробленого рішення.

Для досягнення мети необхідними виявилися такі методи наукового дослідження: аналіз та вивчення літератури, порівняння, вимірювання, експеримент, узагальнення результатів.

Джерелами для дослідження стали наукові роботи та статті на тему машинного навчання, де було представлено інформацію щодо ефективності, переваг та недоліків алгоритмів, інтернет-статті з доступними та ілюстрованими поясненнями принципів роботи алгоритмів, офіційна документація бібліотек, яка застосовувалися під час аналізу наявних рішень та в ході розробки власної програми.

# РОЗДІЛ 1. АНАЛІЗ НАЯВНИХ РІШЕНЬ ДЛЯ РОЗПІЗНАВАННЯ ОБ'ЄКТІВ НА ЗОБРАЖЕННІ ТА В ТЕКСТІ, ВИЗНАЧЕННЯ СЕМАНТИЧНОЇ ПОДІБНОСТІ СЛІВ

Проблема перевірки відповідності тексту та зображень постала практично одночасно з появою цифрового контенту. І хоча на сьогодні не існує загальнодоступних алгоритмів для її розв’язання, деякі розробки з заслуговують на увагу.

## 1.1 Алгоритми для розпізнавання об'єктів на зображенні

Одним із перших запропонованих підходів до розпізнавання об’єктів був алгоритм із застосуванням рухомого вікна (Sliding Windows object detection) [1]. Процес обробки зображення полягав в багаторазовому застосуванні класифікатора до частин вхідного зображення, які виокремлювалися рухомим «вікном» певного розміру. Процедура повторювалася ітеративно зі збільшенням розміру вікна. Очевидно, недоліком такого рішення була надмірна кількість та складність обчислень, яка зростала зі збільшенням розмірів зображення, що впливало на час, потрібний для опрацювання картинки. Крім того, такий спосіб не гарантував точного визначення обмежувальної рамки. Аби позбутися згаданих недоліків, модель було вдосконалено. Використання згорткової нейронної мережі дозволило покращити продуктивність алгоритму. Перевагою алгоритму OverFeat [2], розробленого Pierre Sermanet та ін., стало наскрізне навчання моделі, яка за повним вхідним зображенням могла за один прохід обробити всі можливі «вікна», визначити класи об’єктів у них, не виконуючи окремих обрахунків для кожного регіону. Недоліком моделі була необхідність у великому обсязі даних для навчання,

до того ж алгоритм не враховував контексту об’єктів, тому вимагав додаткової обробки результатів [7].

Більш раціональне рішення було запропоновано в роботі Ross Girshick та ін. «Rich feature hierarchies for accurate object detection and semantic segmentation» [3], де описується алгоритм R-CNN (Regions with CNN features), який розпізнавав об’єкт у межах певного регіону. Система складалася з трьох модулів:

1) визначення регіонів;
2) обчислення векторів властивостей;
3) класифікація.

Алгоритм виділяв на вхідному зображенні близько двох тисяч регіонів-пропозицій (region proposals), не залежних від категорії (category-independent), використовуючи як алгоритм для сегментації вибірковий пошук (selective search); для кожного з таких регіонів за допомогою згорткової нейронної мережі обчислювалися вектори властивостей фіксованої довжини, які були використані декількома SVM для класифікації регіонів зображення. Результатом обробки кожного регіону був певний клас та обмежувальна рамка. Незважаючи на зменшення обчислювальної складності, обраховування класів для двох тисяч регіонів було витратним, тому таке рішення не могло застосовуватися для розпізнавання об’єктів у реальному часі (швидкість складала 47 с/зображ.[5]). Іншим недоліком моделі був статичний алгоритм для пошуку регіонів (selective search) [4], який не брав участі в навчанні моделі, тому міг призводити до похибок.

Згодом Ross Girshick розробив покращену модель R-CNN – Fast R-CNN [5]. У ній було замінено всі SVM на простішу функцію softmax, можливостей якої, як з’ясувалося під час тестування, було достатньо для класифікації об’єктів. Також важливо зазначити, що змінилася послідовність кроків алгоритму: ціле зображення спочатку оброблювалося згортковою нейронною мережею (принцип, схожий на той, що було

використано в OverFeat [2]), що повертала мапу властивостей, а потім відбувався поділ на регіони та їхня класифікація. Однак регіони досі визначалися за алгоритмом вибіркового пошуку (selective search). Здобутком цієї роботи стали покращена точність, збільшена швидкість обчислень, удосконалений процес навчання моделі, проте регіони-пропозиції все ж залишалися вузьким місцем алгоритму, оскільки вони помітно вповільнювали процес навчання.

Зазначену проблему усунули Shaoqing Ren та ін. в удосконаленій моделі Faster R-CNN [6]. Вона базувалася на попередньому рішенні Fast R-CNN [5], але для визначення регіонів замість згаданого алгоритму вибіркового пошуку використовувалася окрема глибока згорткова нейронна мережа Region Proposal Network, яка застосовувалася до мапи властивостей з попереднього кроку й, використовуючи якірні рамки, виокремлювала близько трьохсот можливих регіонів. У підсумку модель показала кращі за Fast R-CNN результати під час тестування та наблизилася до розпізнавання об’єктів у реальному часі, оскільки могла оброблювати відео зі швидкістю п’ять кадрів на секунду.

Новий підхід до розв’язання задачі розпізнавання об’єктів було втілено в моделі Джозефа Редмона та ін. YOLO (You Only Look Once) [7]. Попередні моделі складалися з кількох модулів (окремі мережі або алгоритми для виокремлення регіонів та їх класифікації), які навчалися незалежно один від одного; також моделі вимагали багаторазового застосування класифікатора до виокремлених регіонів. Для алгоритму YOLO ж достатньо «переглянути» зображення лише один раз, щоб вказати, які об’єкти є на картинці та де вони розташовані. Така ефективність забезпечується однією згортковою нейронною мережею, яка одночасно передбачає кілька обмежувальних рамок на зображенні та ймовірні класи для об’єктів, що можуть в них знаходитися. Перевагою моделі є висока швидкість обробки зображення (45 та 150 кадрів на секунду з використанням базової та швидкої версій алгоритму відповідно

(base YOLO model, Fast YOLO (використовується менше фільтрів та дев'ять згорткових шарів замість двадцяти чотирьох))); удвічі кращі показники середньої точності (mAP) у порівнянні з іншими моделями розпізнавання об'єктів у реальному часі; урахування контексту об'єкта (оскільки зображення не розділяється на частини, а аналізується повністю), чого не було в моделях [2], [3]; високий ступінь узагальненості представлення об'єктів у моделі.

Процес обробки зображення схожий на той, що реалізований у моделях R-CNN: комірки є тими регіонами, для яких визначаються обмежувальні рамки та класи об'єктів у них. У ході аналізу зображення поділяється на клітини сіткою розміру $S \times S$. Кожна комірка, у якій розміщується центр певного об'єкта, розпізнає $B$ (зазвичай дві) обмежувальних рамок з різними відношеннями розмірів сторін (4:3, 16:9 тощо). До такого прийому вдаються через те, що в одній комірці можуть розміщуватися центри одразу кількох об'єктів різної форми. Хоча за достатньо дрібної сітки ймовірність виникнення таких ситуацій мала, проблемними однак залишаються випадки, коли одразу кілька об'єктів належать до однієї якірної рамки або кількість об'єктів більша за кількість якірних рамок.

У результаті обробки для кожної такої рамки вказуються:

- $p_c$ – імовірність того, що рамка містить певний об'єкт; обчислюється за формулою:

$$p_c = P(Object) * IoU_{pred}^{truth}, \tag{1.1}$$

де *P(Object)* – ймовірність появи об'єкта в рамці;

$IoU_{pred}^{truth}$ – Intersection over Union, метрика для визначення точності детекції об'єктів, яка обчислюється як

$$IoU = \frac{pred \cap truth}{pred \cup truth}, \quad (1.2)$$

де *pred* – площа розпізнаної рамки;

*truth* – площа дійсної рамки [25]);

- $(b_x, b_y)$ – координати центра рамки відносно комірки;
- $b_h, b_w$ – висота та ширина обмежувальної рамки відносно розмірів зображення;
- для кожної комірки також визначається $C$ – значення ймовірностей для кожного з класів ($Class_i$), до яких може належати об'єкт ($Object$): $P(Class_i|Object)$.

Як і модель OverFeat [2], цей алгоритм застосовується не окремо до кожної з комірок, а до цілого зображення одразу. Результатом обробки зображення є тривимірна матриця $S \times S \times (5 * B + \#classes)$, де число 5 відповідає кількості елементів, що вказують ймовірність об'єкта в рамці, позицію та розмір рамки (тобто значення $p_c, b_x, b_y, b_h, b_w$), $\#classes$ – кількість класів, $B$ – кількість обмежувальних рамок. Кожна комірка описується вектором $1 \times 1 \times (5 * B + \#classes)$ [7].

Незважаючи на свої переваги, модель має обмеження: маленькі об'єкти на зображенні не завжди розпізнаються коректно; функція втрат не чутлива до розмірів обмежувальної рамки, тому однаково сприймає похибки у визначенні великих і малих прямокутників; у порівнянні з Fast R-CNN модель має меншу точність локалізації об'єктів [8].

У наступних роботах "YOLO9000: Better, Faster, Stronger" [8], "YOLOv3: An Incremental Improvement" [16], "YOLOv4: Optimal Speed and Accuracy of Object Detection" [17], "PP-YOLO: An Effective and Efficient Implementation of Object Detector" [18] модель було удосконалено за рахунок багатьох чинників: доповнення даних, нормалізації, ієрархічної класифікації за допомогою WordTree та ін., що дозволило збільшити точність та швидкість обробки зображення.

Оскільки алгоритм YOLO показує найкращі результати під час тестування, саме його можна використати для визначення класів об’єктів на вхідному зображенні.

Актуальним є питання розробки алгоритмів для українськомовного контенту. Складність розв’язання проблеми полягає в тому, що навчання представлених вище моделей проводилося на основі колекцій ImageNet, MS COCO, PASCAL VOC 2007, PASCAL VOC 2012, де класи представлено англійською мовою. Отже, одним з першочергових завдань на майбутнє є створення схожих колекцій українською мовою з метою задоволення потреб вітчизняного користувача.

## 1.2 Алгоритми для розпізнавання іменованих сутностей у тексті

Для розв’язання проблеми перевірки відповідності тексту та зображень варто з’ясувати способи виокремлення сутностей із вхідного тексту. Це завдання належить до сфери обробки природної мови та відоме як NER, або Named Entity Recognition (розпізнавання іменованих сутностей). Іменована сутність – слово або фраза, які дозволяють чітко відрізнити один об’єкт від решти інших, що мають схожі атрибути. Задача NER полягає у визначенні одного слова або послідовності слів, які іменують певну сутність, та класифікації такого імені [9].

### 1.2.1 Підготовчий етап для визначення іменованих сутностей

Важливим попереднім кроком є POS-tagging (part of speech tagging) – процес, який також належить до галузі NLP і полягає у визначенні частини мови для кожного слова тексту. Оскільки POS-теги описують характерну структуру лексичних термінів у тексті, теги використовують, щоб зробити припущення щодо семантики й розпізнати іменовані сутності. Для виконання задачі існує кілька підходів: заснований на правилах,

імовірнісний та з використанням нейронних мереж (наприклад, Recurrent Neural Network, Conditional Random Fields).

POS-тегери на основі правил використовують словник, де для кожного слова вказано потенційні теги. Для розв’язання неоднозначностей застосовують контекстні правила, складені вручну, та регулярні вирази, що є обмеженням такого рішення.

Прихована модель Маркова реалізує імовірнісний підхід і є стохастичною моделлю, яка дозволяє представляти системи, що випадково змінюються і де наступний стан (а точніше – імовірність переходу в наступний стан) залежить лише від поточного. Модель зручно уявити як скінченний автомат або орієнтований граф, у якому станами (вершинами) є теги, що позначають різні частини мови (іменники, дієслова, прикметники тощо), а на ребрах вказані ймовірності того, що з певного стану машина може перейти в наступний [23]. Матричне представлення прихованої моделі Маркова є основою для алгоритму Вітербі, що визначає найбільш відповідний список станів (шлях Вітербі), який у контексті ланцюгів Маркова отримує найімовірнішу послідовність подій, що відбулися [22]. Алгоритм здійснює POS-tagging, але для роботи вимагає анотованого корпусу текстів. Модель не може передбачити тег для слова, якого не було в навчальній колекції, що є недоліком.

### 1.2.2 Named Entity Recognition

Для вирішення задачі NER використовують алгоритми на основі заданих наперед правил, на основі словників, алгоритми машинного навчання, а також гібридні моделі [10].

У першому випадку системи для розпізнавання сутностей спираються на синтактико-лексичні моделі, словники синонімів, семантичні та синтаксичні правила мови [9]. Недоліком такого підходу є довгий ручний процес визначення правил таким чином, аби вони

описували всі можливі випадки, а також неефективність під час аналізу термів, які система бачить уперше.

Для розв’язання задачі NER також використовують моделі машинного навчання, зокрема згорткові нейронні мережі, Long Short-Term Memory із застосуванням Conditional Random Fields (CRF), оскільки нейронні мережі дозволяють моделювати нелінійні дані та самостійно визначати закономірності в тексті.

Однак найкращі результати показують системи, які поєднують різні способи, перелічені раніше: наприклад, моделі машинного навчання використовуються разом зі словниками або набором певних семантичних та синтаксичних правил.

Для обробки природної мови (та визначення іменованих сутностей зокрема) розроблено кілька бібліотек з відкритим кодом: SpaCy, NLTK, Stanford NER. Розгляньмо детальніше принцип роботи алгоритму для NER у бібліотеці SpaCy.

У програмі використовується модель на основі переходів (Transition-Based Chunking Model), запропонована в дослідженні Guillaume Lample та ін. «Neural Architectures for Named Entity Recognition» [11], принцип роботи якої подібний до машини зі станами, наступний перехід (SHIFT, REDUCE, OUT) і конфігурація якої залежить від слова, що знаходиться на вершині вхідного стека-буфера. Нейронна мережа, що розв’язує проблему NER у бібліотеці, спирається на принцип Embed (представлення слів у векторному просторі (word vectors) за допомогою, наприклад, Word2Vec), Encode (удосконалення векторного представлення слів із урахуванням їхнього контексту та позицій з використанням CNN або BiLSTM; водночас вихідний вектор для слова формується не лише за допомогою нейронної мережі, але береться до уваги й початковий вектор слова), Attend (зведення матриці документа, що оброблюється, до єдиного вектора), Predict (використання нейронної мережі для класифікації) [26].

Можливість визначати частини мови та іменовані сутності реалізовано в бібліотеках NLTK та SpaCy. Перша з них дозволяє розпізнавати лише російську та англійську мови. Друга надає доступ до натренованих моделей, що дозволяють обробляти тексти, написані англійською, китайською, французькою, німецькою та ін. Хоча готового рішення для української мови в SpaCy немає, можна використати натреновану модель для багатомовних текстів (multi-language) [19]. Для розмітки українськомовного тексту за частинами мови існує UDPipe API [24].

Якщо якість результатів роботи алгоритму не задовольнятиме, бібліотека SpaCy вимагатиме поповнення моделями, які натреновані розпізнавати сутності в українськомовних текстах. Цей пункт потребуватиме додаткових досліджень, а також значних ресурсів та часу.

## 1.3 Алгоритми для семантичного порівняння слів

Після визначення назв класів для об’єктів на зображенні та у тексті потрібно виконати їхнє порівняння, аби з’ясувати ступінь подібності зображення та тексту. Для цього існує кілька опцій.

### 1.3.1 Порівняння слів з використанням векторного представлення

Для порівняння значення слів можна представити їх у вигляді векторів.

One-hot encoding – векторне представлення слів, де вектор $V_i$ слова $W_i$ має розмірність $N$, де $N$ – кількість слів у словнику, і складається з нулів на всіх позиціях, окрім тієї, що відповідає порядковому номеру слова $W_i$ у словнику: $V_i = [0,0,\dots 0,1,0,\dots 0]$. Такий спосіб представлення не є ефективним, оскільки вектор дуже розріджений і не відображає семантичного значення слова.

Word2Vec – векторне представлення слів таким чином, що близькі за значенням слова розташовуються поруч у векторному просторі і навпаки. Значення кожного елемента вектора можна інтерпретувати як міру належності слова до певної неявної категорії (наприклад, до теми садівництва, спорту чи медицини). Вектори для слів визначаються в процесі навчання нейронних мереж: згорткової нейронної мережі, LSTM (навчання з наглядом, є потреба в анотованих текстах), CBOW, Skip-Gram (навчання без нагляду, немає потреби в анотованих текстах) [12] [13]. Для порівняння слів можна скористатися косинусною мірою схожості між векторами, що відповідають цим словам. Незважаючи на широке застосування моделі Word2Vec, вона має кілька недоліків: доступні вектори лише для тих слів, які були в текстах під час навчання моделі; векторне представлення форм того самого слова може значно відрізнятися, оскільки немає можливості перевикористання вже відомих параметрів.

FastText – удосконалений метод побудови векторного представлення слів, який враховує морфологічну будову слова, яка не береться до уваги в інших методах векторного представлення, зокрема в Word2Vec, де кожне слово вважається унікальним, а тому має власне неповторне представлення (хоча насправді спільнокореневі слова, різні форми одного слова, слова з однаковими суфіксами та префіксами зазвичай мають дуже схоже, якщо не ідентичне, значення). Завдяки цій особливості модель FastText дозволяє обчислювати вектори навіть для слів, які рідко зустрічалися або взагалі були відсутні в колекції, за умови, що хоча б 1 *n-грама* цих слів була оброблена під час навчання. Алгоритм розбиває слова на *n-грами* – підслова довжини *n,* – хешує кожну *n-граму*, ставлячи їй у відповідність число від *1* до *B* (*B* – задана кількість різних хешів), після чого визначає векторне представлення не для кожної *n-грами*, а для деякої їх підмножини, що мають однаковий хеш. У результаті вектором для слова є сума векторів *n-грам*, з яких воно складається [14].

Хоч для навчання моделі потрібно у 1.5 рази більше часу, який витрачається на обробку n-грам, алгоритм показує кращі результати під час визначення подібності слів. Перевагою також є здатність алгоритму визначати вектори для слів, яких не було в навчальній колекції [13].

Описаний метод реалізовано в бібліотеці FastText, розробленій фахівцями Facebook AI Research. Вона вирішує задачу класифікації та представлення текстів. Для визначення векторів слів у готових моделях використовується нейронна мережа CBOW, довжина n-грам складає 5 символів. Бібліотека дозволяє отримати вектори для слів 157 мов, зокрема й української [20].

### 1.3.2 Порівняння слів з використанням семантичного словника

Для порівняння значення слів також можна використати WordNet – ієрархічно організовану лексичну базу даних для англійської мови. У ній зібрано слова чотирьох частин мови: іменники, прикметники, дієслова та прислівники, які поєднано в групи синонімів – *синсети*. Іменники пов’язані зв’язками: *гіперонім* – відношення між словом і більш загальним поняттям, *гіпонім* – відношення між словом і більш специфічним поняттям, *меронім* – відношення між складовою та цілим, *голонім* – відношення між цілим та складовою, *антонім* – протилежне за значенням слово до даного.

Хоча робота зі створення WordNet-подібного семантичного словника для української мови розпочалася ще в 2009 році, поки готовим є лише його фрагмент, розроблений фахівцями Львівського національного університету «Львівська політехніка». Лексико-семантична база містить 194 синсети (у той час, як WordNet налічує 117659 синсетів), між якими встановлено відношення *гіпо/гіперонімії, антонімії, меронімії/голонімії* [21]. Складання такого словника викликає труднощі й потребує

подальшого ґрунтовного дослідження, адже поки що доступна система для української мови не в змозі замінити WordNet.

Для визначення подібності слів використаємо алгоритм Wu-Palmer, реалізований у бібліотеці NLTK. Згідно з алгоритмом, значення подібності визначається за формулою:

$$Sim_{WP}(C_1, C_2) = \frac{2*N}{N_1+N_1}, \quad (1.3)$$

де $C_1$, $C_2$ – вузли слів;

$N$ – відстань від найближчого спільного предка $CS$ вузлів $C_1$, $C_2$ до кореневого вузла $R$;

$N_1$, $N_2$ – відстані від вузлів $C_1$, $C_2$ до кореневого вузла $R$ (рисунок 1.1) [15].

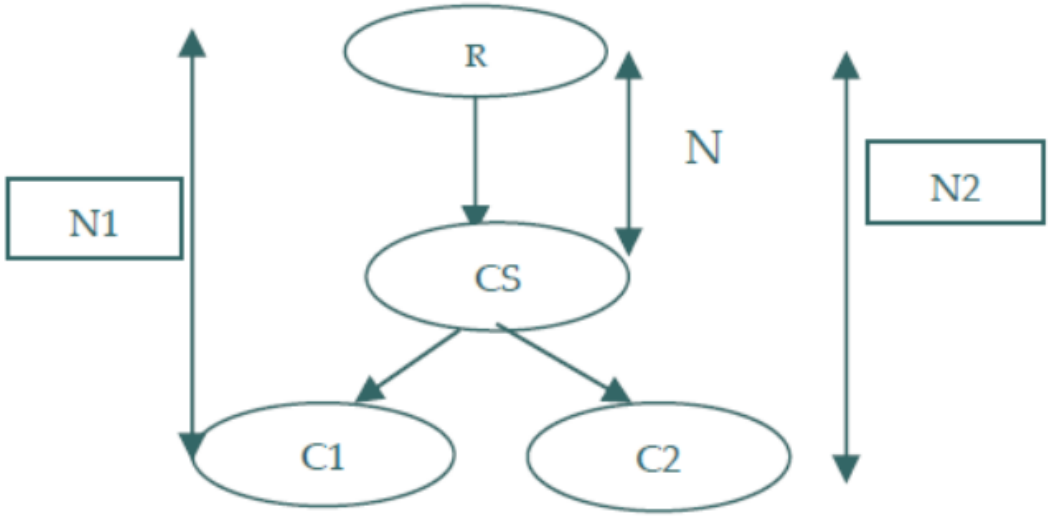



*Рисунок 1.1 – Алгоритм Wu-Palmer [15]*

Підсумовуючи, зазначимо, що хоча на сьогодні не існує комплексного вирішення проблеми, розглянуті вище алгоритми для розпізнавання об’єктів у тексті та на зображенні, побудови векторного представлення слів, порівняння семантики об’єктів, можуть служити

надійним підґрунтям для розробки системи для перевірки відповідності зображення та тексту.

# РОЗДІЛ 2. ПОШУКИ РІШЕНЬ ДЛЯ ВИЗНАЧЕННЯ ПОДІБНОСТІ МІЖ ЗОБРАЖЕННЯМ ТА ТЕКСТОМ

Якщо взяти до уваги різноманіття способів представлення зображення у вигляді тексту, визначення семантики зображень та тексту, порівняння інформації, то для побудови системи перевірки відповідності зображень та тексту не може існувати єдиного рішення.

Найбільш ефективним, на наш погляд, є спосіб перевірки відповідності зображення та тексту шляхом визначення схожості об’єктів, які містяться в них. Це пов’язано з тим, що саме об’єкти несуть найбільше семантичне навантаження.

Для реалізації такого рішення необхідно:

1) визначити класи об’єктів на зображенні;
2) визначити об’єкти в тексті;
3) порівняти визначені класи.

## 2.1 Визначення класів об'єктів на зображенні

Для визначення класів об’єктів на зображенні буде використано модель YOLOv5, розроблену Glenn Jocher з використанням фреймворку PyTorch на основі алгоритму YOLOv4, а саме версію YOLOv5m з уже натренованими вагами. Ця модель має 290 шари, 22.3 млн. параметрів, середня точність (mAP[0.5]) на даних колекції COCO val2017 складає 68.9, може розпізнати 365 класів.

Структура детектора показана на рисунку 2.1 і включає:

1) input (зображення);
2) backbone;
3) neck (SPP модуль для збільшення рецептивного поля, PANet path-aggregation neck), ;

4) head (YOLOv3).

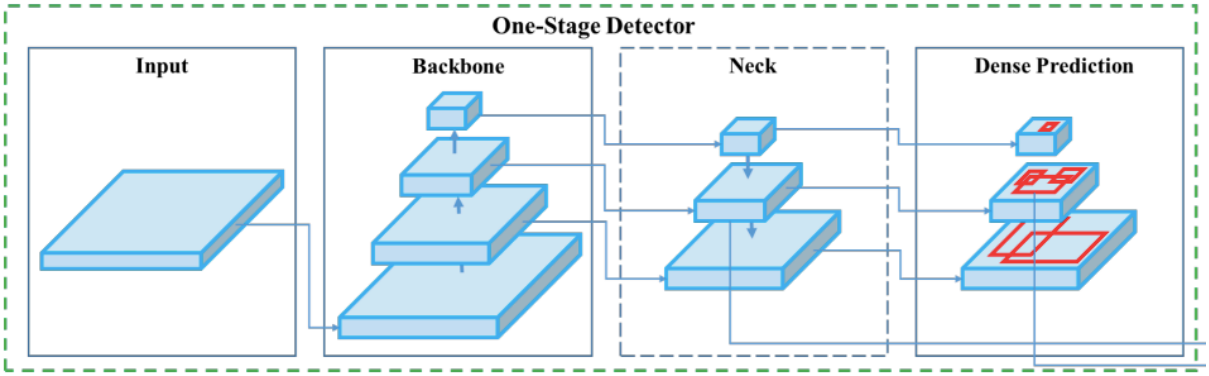



*Рисунок 2.1 – Структура детектора [17]*

У роботі дослідника Alexey Bochkovskiy та ін. «YOLOv4: Optimal Speed and Accuracy of Object Detection» [17] використовується поняття Bag of freebies – методи, які покращують якість розпізнавання об'єктів без збільшення вартості обробки зображення, а за рахунок ускладнення процесу навчання. Основною складовою цього набору методів є доповнення даних – збільшення варіативності вхідних зображень. Для цього виконують фотометричні та геометричні спотворення: змінюють яскравість, контраст, насиченість, обтинають, перевертають картинку, змінюють її масштаб або закривають її частини (*cutout data augmentation*) (рисунок 2.2, a, c, e), об'єднують кілька зображень в одне (*mosaic data augmentation*) (рисунок 2.2, d), накладають кілька зображень одне на одне (*mixup data augmentation*) (рисунок 2.2, b). До Bag of freebies також відносять використання негативних навчальних прикладів, згладжування класів (встановлення меншої вірогідності для класу, ніж є насправді: 0.9 замість 1.0), Self-Adversarial Training (метод тренування, за якого модель спочатку деформує вхідне зображення, спираючись на свої знання, а потім уже змінене зображення використовує для свого навчання), що дозволяє уникнути перенавчання моделі.

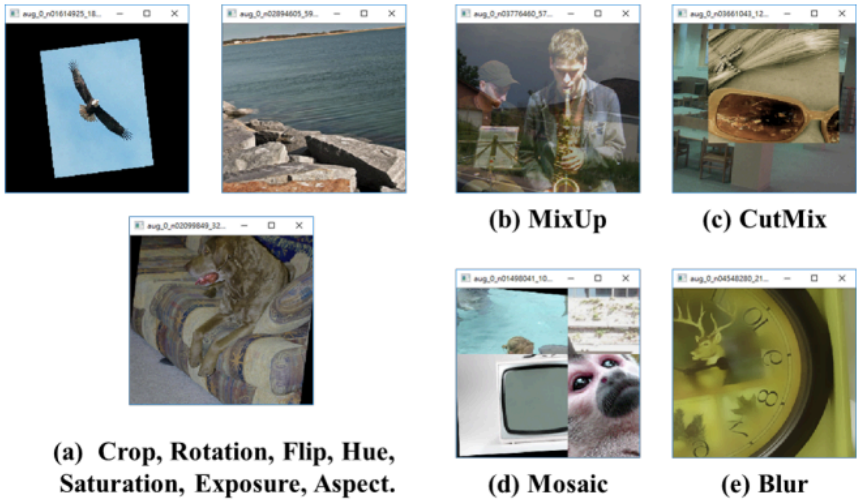



*Рисунок 2.2 – Різні способи доповнення даних: a – відтинання, поворот, відтінок, насиченість, експозиція; b – накладання зображень; c – поєднання частин зображень; d – мозаїчне об'єднання зображень; e – розмиття зображень [17]*

Іншим важливим поняттям є Bag of specials – методи, які несуттєво збільшують вартість обробки, але значно покращують якість розпізнавання. Зокрема сюди відносять кращі функції активації (наприклад,

$$\text{ReLU: } f(x) = max(x, 0), \tag{2.1}$$

$$\text{Swish: } f(x) = x * sigmoid(ßx)), \tag{2.2}$$

залишкові мережі (residual networks, наприклад, BiFPN; такі мережі беруть до уваги не лише дані з попереднього шару, але й з нижчих), методи для фільтрування результатів виконання алгоритму.

Якість роботи алгоритму було протестовано з використанням колекції ILSVRC 2012 val для перевірки правильності класифікації та колекції MS COCO (test-dev 2017) з метою оцінки правильності розпізнавання об'єктів.

Завантажити модель можна за допомогою PyTorchHub, указавши потрібну версію програми. Після цього отриманій моделі треба передати на вхід список шляхів до файлів із зображеннями. У результаті обробки буде повернуто об’єкт, який міститиме для кожного зображення інформацію про координати (*xmin, ymin, xmax, ymax*) обмежувальної рамки, імовірність класу (*confidence*), номер (*class*) та назву (*name*) класу для кожного розпізнаного об’єкта (лістинг 2.1).

```
1. import torch
2. # Loading the model
3. model = torch.hub.load('ultralytics/yolov5', 'custom',
   path='yolov5m_Objects365.pt')
4. # Specifying the paths to the images
5. imgs = ['zidane.jpeg', 'people.jpeg']
6. # Processing the images
7. results = model(imgs)
8. # Showing the results
9. for r in results.pandas().xyxy:
10.     print(r)
11.     print('\n')
```

*Лістинг 2.1 – Розпізнавання об'єктів на зображенні за допомогою алгоритму YOLO*

*Output* з лістингу 2.1 (рисунок 2.3).

```
         xmin        ymin         xmax        ymax  confidence  class    name
0  746.929810   43.906097  1152.728516  716.236328    0.907331      0  Person
1  431.014099  433.541504   525.119202  720.000000    0.805805     43     Tie
2   57.333435  200.546906  1149.097412  715.865601    0.779514      0  Person
3  983.769409  321.179230  1136.907593  704.514404    0.354390     43     Tie
4  977.071899  317.970215  1140.722778  697.951172    0.337996     23  Flower


         xmin        ymin         xmax         ymax  confidence  class    name
0  706.812866  190.054642  1457.999634  1365.000000    0.895168      0  Person
1  535.604248  371.482513  1177.084595  1363.167236    0.397575      0  Person
```

*Рисунок 2.3 – Результат розпізнавання об'єктів на зображенні*

Гадаємо, після визначення об'єктів на зображенні доречно виконати фільтрування результатів. Насамперед, варто позбутися об'єктів, які мають маленьке значення ймовірності (*confidence*) (встановимо поріг 0.5), оскільки, швидше за все, вони не важливі або розпізнані неправильно, що може негативно вплинути на точність роботи нашої програми в майбутньому. Іншим обмеженням, яке можна накласти на визначені об’єкти, є частка їхньої площі від площі всього зображення (встановимо поріг 0.01). Ми припускаємо, що дуже маленькі за розміром об’єкти також не несуть значного семантичного навантаження, але можуть негативно вплинути на якість визначення подібності тексту та зображення (погіршення якості може відбутись через те, що для дрібного об’єкта, визначеного на зображенні, не знайдеться відповідника в тексті, оскільки в описі найчастіше вказуються основні об’єкти картинки, а незначні деталі пропускаються). З урахуванням вищенаведеного функція для визначення об’єктів на зображенні матиме вигляд, як у лістингу 2.2.

```
1.  # Function to detect objects in the image
2.  # and filter them with certain confidence and area quotient
3.  def get_objects_from_image(
4.          image_path,
5.          confidence_threshold=0.5,
6.          min_area_threshold=0.01):
7.      results = model(image_path)
8.      width, height, _ = results.imgs[0].shape
9.      full_area = width*height
10.     for r in results.pandas().xyxy:
11.         # calculating areas of objects
12.         area_arr = (r.xmax - r.xmin) * (r.ymax - r.ymin)
13.         # getting names of classes
14.         objects_arr = r.name.array
15.         objects_areas_arr = area_arr.array
16.         # getting confidence values
17.         confidence = r.confidence.array
18.         zipped = list(zip(confidence, objects_arr, objects_areas_arr))
19.         # filtering classes by confidence and area
20.         confident_objects = [e for e in zipped if e[0] >=
    confidence_threshold and e[2]/full_area >= min_area_threshold]
21.         objects = []
22.         areas = []
23.         if len(confident_objects) > 0:
24.             _, objects, areas = zip(*confident_objects)
```

```
25.         return objects, areas
```

*Лістинг 2.2 – Функція для розпізнавання об'єктів на зображенні, фільтрування їх за площею та ймовірністю*

## 2.2 Визначення об'єктів у тексті

Наступним етапом є визначення об’єктів у тексті. Текст для комп’ютера є лише набором символів, позбавлених будь-якого сенсу, тому для розпізнавання семантики документа машиною варто певні послідовності символів віднести до відповідних категорій. Такими категоріями в досліджуваній предметній області є частини мови. Далі необхідно взяти до уваги: мова влаштована таким чином, що зазвичай для позначення сутностей використовують іменники, отже, визначення об’єктів зводиться до пошуку в тексті всіх слів названої частини мови. Для розв’язання цього завдання було розроблено розглянуті вище POS-tagging алгоритми. Однак доволі часто в текстах на позначення об’єктів уживають складні власні назви або іменникові словосполучення, іншими словами, конструкції з кількох слів, які не обов’язково є іменниками, тому одного лише визначення частин мови виявляється недостатньо для аналізу семантики й важливо мати засоби розпізнавання та класифікації конструкцій з кількох слів. Такими є моделі, що здійснюють визначення іменованих сутностей. Рішення для розв’язання завдань тегування та NER імплементовано в бібліотеках NLTK та SpaCy, перша з яких буде використана для виокремлення всіх іменників, друга – для класифікації іменованих сутностей.

Для пошуку іменників потрібно розбити вхідний текст на токени та їх список передати у функцію з бібліотеки NLTK, що здійснює POS-tagging, після чого необхідно відфільтрувати лише слова, що позначені як іменники. Реалізацію функції наведено в лістингу 2.3.

```
from nltk import pos_tag
from nltk import word_tokenize
input = "Two men are walking in front of a bus"
# Function to get nouns from a string
def get_nouns(string):
    # Splitting the sentence into the words
    words = word_tokenize(string)
    # Performing POS-tagging
    words = pos_tag(words)
     # Filtering nouns
     return [word for (word, tag) in words if tag[0] == 'N']

print(get_nouns(input))
```

*Лістинг 2.3 – Функція для отримання іменників з тексту*

*Output* з лістингу 2.3*:* ['men', 'front', 'bus'].

Для визначення іменованих сутностей достатньо передати текст у відповідну функцію з бібліотеки SpaCy (лістинг 2.4). У результаті обробки отримаємо список класифікованих сутностей.

```
import spacy
# Loading the model
nlp = spacy.load("en_core_web_sm")
# Text below is taken from NLTK page on Wikipedia
input = "The Natural Language Toolkit, or more commonly NLTK,
is a suite of libraries and programs for symbolic and
statistical natural language processing (NLP) for English
written in the Python programming language. It was developed by
Steven Bird and Edward Loper in the Department of Computer and
Information Science at the University of Pennsylvania. NLTK
includes graphical demonstrations and sample data. It is
accompanied by a book that explains the underlying concepts
behind the language processing tasks supported by the toolkit,
plus a cookbook."
# Processing the string
entities = nlp(input)
for ent in entities.ents:
    print(ent.text, ent.label_)
```

*Лістинг 2.4 – Розпізнавання іменованих сутностей у тексті*

*Output* з лістингу 2.4:

The Natural Language Toolkit LAW

NLTK ORG

NLP ORG

English LANGUAGE

Steven Bird PERSON

Edward Loper PERSON

the Department of Computer and Information Science ORG

the University of Pennsylvania ORG

Оскільки наше завдання – знайти об’єкти, які мають візуальне представлення (тобто можуть дійсно траплятися на зображеннях), то важливими для нас будуть тільки сутності, позначені тегами "PERSON" ('People, including fictional'), "FAC" ('Buildings, airports, highways, bridges, etc.'), "MONEY" ('Monetary values, including unit'), "ORG" ('Companies, agencies, institutions, etc.'), "LOC" ('Non-GPE locations, mountain ranges, bodies of water'), "PRODUCT" ('Objects, vehicles, foods, etc. (not services)'), ('QUANTITY', 'Measurements, as of weight or distance'), "WORK_OF_ART" ('Titles of books, songs, etc.'). Крім того, отриманим тегам необхідно поставити у відповідність клас (лістинг 2.5).

```
# Filtering entities
needed_entity_labels = ["PERSON", "FAC", "MONEY", "ORG", "LOC", "PRODUCT", "WORK_OF_ART"]
filtered_entities = [e for e in entities.ents if e.label_ in needed_entity_labels]
# Dictionary to convert labels to classes
label_to_class_dict = {
    "ORG": "organisation",
    "LOC": "location",
    "FAC": "facility",
    "WORK_OF_ART": "art"
}
# Function to convert labels to classes
def label_to_class(label):
    return label_to_class_dict.get(label, label.lower())

# Converting labels to classes
```

```
16. classes = [label_to_class(e.label_) for e in
    filtered_entities]
17. print(classes)
```

*Лістинг 2.5 – Фільтрування тегів іменованих сутностей, перетворення тегів на назви класів*

*Output* з лістингу 2.5: ['organisation', 'organisation', 'person', 'person', 'organisation', 'organisation'].

Наступним кроком має бути об’єднання об’єктів-іменників з класами іменованих сутностей. Для цього до списку класів треба додати іменники, крім тих, що стали іменованою сутністю або її частиною (лістинг 2.6).

```
1. # Concatenated recognised named entities
2. all_named_entities = " ".join([e.text for e in entities.ents])
3. # Classes + nouns
4. all_objects = classes + [n for n in get_nouns(input) if n not
   in all_named_entities]
5. print(all_objects)
```

*Лістинг 2.6 – Об’єднання множини іменників та назв класів іменованих сутностей*

*Output* з лістингу 2.6: ['organisation', 'organisation', 'person', 'person', 'organisation', 'organisation', 'suite', 'libraries', 'programs', 'language', 'processing', 'Python', 'programming', 'language', 'demonstrations', 'data', 'book', 'concepts', 'language', 'processing', 'tasks', 'toolkit', 'cookbook'].

## 2.3 Порівняння об'єктів тексту та зображення

### 2.3.1 Порівняння значення двох слів

Отримавши два списки об’єктів із зображення та тексту, виконаємо їх порівняння. Оскільки немає відповідності між об’єктами з першого та

другого списків, треба кожен об’єкт з першого списку порівняти з кожним об’єктом другого списку. Обираючи найбільші значення з отриманої матриці, ми знайдемо подібність для пар об’єктів. Важливо також узяти до уваги, що між об’єктами існує взаємно однозначна відповідність (одному об’єкту з першого списку відповідає один об’єкт з другого списку і навпаки), тому шукані максимальні значення мають знаходитися в різних рядках та колонках матриці (інакше якщо кілька максимальних значень знаходиться в одному рядку, то слову з першого списку відповідатиме кілька слів з другого; якщо кілька максимальних значень знаходиться в одній колонці, то одному слову з другого списку відповідатиме кілька слів з першого). Для визначення схожості між словами можна використати або *WordNet* та розглянутий вище алгоритм *Wu-Palmer* (див. формулу 1.3) (лістинг 2.7), або косинусну міру схожості (лістинг 2.8):

$$cos\_similarity(U,V) = \frac{U*V}{\|U\|*\|V\|}, \tag{2.3}$$

де $U$, $V$ – попередньо обчислені вектори слів.

```
1. from nltk.corpus import wordnet
2. def wordnet_word_similarity(w1, w2):
3.     # Getting synsets for the words
4.     syn1 = wordnet.synsets(w1)[0]
5.     syn2 = wordnet.synsets(w2)[0]
6.     # Calculating similarity between two words
7.     # using Wu-Palmer algorithm
8.     return syn1.wup_similarity(syn2)
9.
10. print(wordnet_word_similarity('vehicle', 'car')
```

*Лістинг 2.7 – Функція для обчислення подібності між словами за допомогою семантичного словника та алгоритма Wu-Palmer*

*Output* з лістингу 2.7: 0.8.

```
# Function to calculate cosine similarity
def cosine_similarity(v1, v2):
    if len(v1) == 0 or len(v2) == 0:
        return 0
    a = 0
    for i in range(len(v1)):
        a += v1[i] * v2[i]
    n1 = vector_norm(v1)
    n2 = vector_norm(v2)
     if n1*n2 == 0:
         return 0
     return a / (n1 * n2)
```

*Лістинг 2.8 – Функція для обчислення косинусної міри схожості між двома векторами*

Для обчислення подібності двох слів за косинусною мірою схожості, насамперед, необхідно подати кожне з них у вигляді вектора. Таке представлення можна отримати за допомогою бібліотеки *FastText* (лістинг 2.9). Для роботи буде використано готову модель для англійської мови “cc.en.300.bin”. Реалізацію функції для обчислення подібності між словами наведено в лістингу 2.10.

```
import fasttext
# Loading the model
model = fasttext.load_model('cc.en.300.bin')
# Getting the word vector
word_vector = model.get_word_vector("motorbike")
```

*Лістинг 2.9 – Обчислення векторного представлення слова за допомогою бібліотеки FastTest*

```
def fasttext_word_similarity(w1, w2):
    v1 = model.get_word_vector(w1)
    v2 = model.get_word_vector(w2)
    return cosine_similarity(v1, v2)
```

*Лістинг 2.10 – Функція для обчислення подібності між словами за допомогою векторного представлення*

### 2.3.2 Порівняння списків об’єктів

Нижче наведемо способи порівняти списки об’єктів.

$A = \{a_1, \dots, a_n\}$ – список класів об’єктів на зображенні;

$B = \{b_1, \dots, b_m\}$ – список об’єктів у тексті;

$s(c1, c2)$– функція для визначення схожості між класом $c_1$ та $c_2$

$S = \begin{pmatrix} s(a_1, b_1) & s(a_1, b_2) & s(a_1, b_m) \\ s(a_2, b_1) & s(a_2, b_2) & s(a_2, b_m) \\ & \vdots & \\ s(a_n, b_1) & s(a_n, b_2) & s(a_n, b_m) \end{pmatrix}$ – матриця схожості між усіма парами об’єктів зі списків A та B;

$V = \{v_1, \dots, v_k\},\ k = min(n, m)$ – список найбільших значень схожості між парами об’єктів з обох списків;

$v_i = t \mid t = max(S), row(t) \notin \{row(v_1), \dots, row(v_{i-1})\},\ col(t) \notin \{col(v_1), \dots, col(v_{i-1})\};$

$row(t)$ – функція, яка повертає номер рядка, з якого значення *t* було взяте;

$col(t)$ – функція, яка повертає номер колонки, з якої значення *t* було взяте;

$min(a, b)$ – функція, яка повертає мінімальне з двох чисел*;*

$max(S)$ – функція, яка повертає максимальне значення в матриці *S;*

$P = \{p(a_1), \dots, p(a_n)\}$ – список значень площі об’єктів на зображенні;

$p(x)$ – функція обчислення площі обмежувальної рамки об’єкта *x* на зображенні.

Якщо вважати, що всі об’єкти на зображенні однаково важливі (звідси множник $\frac{1}{k}$, який кожному об’єкту надає однакову вагу), то подібність можна порахувати за формулою (реалізацію наведено в лістингу 2.11):

$$Smilarity = \sum_{i=1}^{k} \frac{1}{k} v_i. \quad (2.4)$$

Якщо ж вважати, що важливість об’єкта залежить від частки його площі від суми площ усіх об’єктів на зображенні, тоді під час пошуку максимального значення в матриці подібності варто запам’ятовувати номер рядка, де це значення знаходиться. Номер рядка – позиція класу об’єкта із зображення в списку усіх класів. Номер рядка допоможе

зберегти відповідність між класом об’єкта та його площею. Отже, елемент $v_i$ списку $V$ максимальних значень подібності тоді матиме вигляд:

$$v_i = (t, row(t)) \mid t = max(S), row(t) \notin \{row(v_1), \dots, row(v_{i-1})\},\ col(t) \notin \{col(v_1), \dots, col(v_{i-1})\}.$$

Тоді для обчислення подібності двох списків класів з урахуванням площі об’єктів на зображенні формула матиме вигляд (реалізацію наведено в лістингу 2.11):

$$Similarity = \sum_{i=1}^{k} \frac{p_{v_i[1]}}{\sum_{j=1}^{n} p_j} v_i[0]. \quad (2.5)$$

```
# Function to build similarity matrix for all pairs of classes in lists
# using the similarity_func to calculate similarity between words
def build_matrix(word_list1, word_list2, similarity_func):
    matrix = []
    for w1 in word_list1:
        vector = []
        for w2 in word_list2:
            vector.append(similarity_func(w1, w2))
        matrix.append(vector)
    return matrix

# Function to find a single max value in the matrix along with its indices
def get_max_value(matrix):
    cols = len(matrix[0])
    k = numpy.argmax(matrix)
    i = int(k / cols)
    j = k - i * cols
    return matrix[i][j], i, j

# Function to set -1 to cells which can not be
# used to get max value
def update_matrix(matrix, ii, jj):
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            if i == ii or j == jj:
                matrix[i][j] = -1

# Function to get max similarity values as a vector
def get_max_values(matrix):
    k = min(len(matrix), len(matrix[0]))
    vector = [0] * k
    for i in range(k):
```

```
33.         vi = get_max_value(matrix)
34.         vector[i] = (vi[0],vi[1]) # max_value and its row; row
    - index of the corresponding image object
35.         update_matrix(matrix, vi[1], vi[2])
36.     return vector
37.
38. # Function to calculate similarity between two lists of
    classes
39. # supposing importance depends on object area
40. def calculate_similarity_area(image_objects, text_objects,
    image_objects_areas, similarity_func):
41.     if len(image_objects) == 0 or len(text_objects) == 0:
42.         return 0
43.     matrix = build_matrix(image_objects, text_objects,
    similarity_func)
44.     max_values = get_max_values(matrix)
45.     full_area = 0
46.     for a in image_objects_areas:
47.         full_area += a
48.     result = 0
49.     for i in range(len(max_values)):
50.         max_object_area =
    image_objects_areas[max_values[i][1]]
51.         result += max_object_area / full_area *
    max_values[i][0]
52.     return result
53.
54. # Function to calculate similarity between two lists of
    classes
55. # supposing all objects are equally important
56. def calculate_similarity_n(word_list1, word_list2,
    similarity_func):
57.     if len(word_list1) == 0 or len(word_list2) == 0:
58.         return 0
59.     matrix = build_matrix(word_list1, word_list2,
    similarity_func)
60.     max_values = get_max_values(matrix)
61.     result = 0
62.     for i in range(len(max_values)):
63.         result += 1 / len(word_list1) * max_values[i][0]
64.     return result
65.
66. print(calculate_similarity_area(["dog", "car", "bus"], ["dog",
    "automobile"], [0.13, 0.59, 0.75],
67.                                 fasttext_word_similarity))
68. print(calculate_similarity_area(["dog", "car", "bus"], ["dog",
    "automobile"], [0.13, 0.59, 0.75],
69.                                 wordnet_word_similarity))
```

*Лістинг 2.11 – Функції для обчислення подібності між двома списками об'єктів з урахуванням площі (функція calculate_similarity_area) (див. формулу 2.5), без урахування площі (функція calculate_similarity_n) (див. формулу 2.4)*

*Output* з лістингу 2.11:

0.37026498575955724

0.4834791059280855

Іншим варіантом порівняння класів є об’єднання слів списків у два рядки та визначення схожості вже між реченнями. Однак таке рішення не можна реалізувати з використанням WordNet (оскільки його засоби дозволяють порівняти лише окремі слова) і можливе тільки у випадку використання векторного представлення слів. У бібліотеці FastText обчислення вектора речення відбувається шляхом пошуку середнього арифметичного для векторів усіх слів речення та спеціального токена кінця речення (для *supervised* моделей). У разі застосування *unsupervised* моделей вектор для токена кінця речення не враховується, а кожен вектор слова спочатку ділиться на модуль цього вектора, після чого відбувається обчислення середнього значення отриманих векторів. Подібність векторів речень також обчислюється за косинусною мірою схожості. Реалізацію цього способу порівняння наведено в лістингу 2.12.

$$Similarity = cos\_similarity(sentence\_vector(A), sentence\_vector(B)), \quad (2.6)$$

де *cos_similarity* – функція для обчислення косинусної міри схожості;

*sentence_vector(X)* – функція для знаходження векторного представлення для речення, утвореного конкатенацією слів списку *X*.

```
1. def calculate_similarity_with_sentences(word_list1,word_list2):
2.     return cosine_similarity(
```

```
        model.get_sentence_vector(
            " ".join(word_list1)),
        model.get_sentence_vector(
            " ".join(word_list2)))
print(calculate_similarity_with_sentences(["dog", "car",
"bus"], ["dog", "automobile"]))
```

*Лістинг 2.12 – Функція для обчислення подібності між двома списками слів з використанням косинусної міри схожості між векторами речень*

*Output* з лістингу 2.12: 0.7771885924519614

Отже, ми визначили інструменти для пошуку об’єктів у тексті та на зображенні, дослідили можливості для векторного представлення слів та порівняння їхнього значення, вказали способи для обчислення подібності двох списків об’єктів. Наступним кроком шляхом тестування, визначення обмежень, переваг та недоліків кожного з підходів перевіримо, яке з рішень є найбільш вдалим.

# РОЗДІЛ 3. ТЕСТУВАННЯ РОЗРОБЛЕНИХ АЛГОРИТМІВ

Тестування – невід'ємна складова будь-якого процесу розробки, адже яким би надійним, обґрунтованим та лаконічним не був алгоритм в теорії, лише практичне застосування дозволяє зробити справжні висновки про правильність та точність його роботи.

## 3.1 Визначення метрик

Для оцінки зазначених показників у галузі машинного навчання було розроблено багато універсальних метрик, які не лише дозволяють об'єктивно проаналізувати якість алгоритму, але й порівняти його з подібними алгоритмами для того, аби визначити найкращий.

Вибір таких метрик напряму залежить від того, що саме є найважливішим для користувача під час роботи програми і яку найголовнішу проблему вона має розв'язувати. Призначення нашої розробки – перевіряти відповідність між зображенням та його описом. Отже, найсуттєвішим показником якості є точність результатів роботи алгоритму.

### 3.1.1 Average Precision

Для оцінки алгоритму можна використати середню точність (*Average Precision, AP*) – метрику, яка застосовується в машинному навчанні для вимірювання точності визначення об'єктів на зображенні [27]. Для обчислення цього значення потрібні додаткові показники: повнота (*Recall*), точність (*Precision*), а також *IoU (Intersection over union)*. Однак оскільки зараз нас цікавить точність визначення подібності тексту та зображення, то замість останньої метрики *IoU* ми використаємо метрику *Similarity* (див.

формули 2.4, 2.5, 2.6), яку було описано в попередньому розділі і якою позначатимемо схожість між зображенням та його описом. Значенням середньої точності (*AP*) є площа під кривою точність-повнота (*Precision-Recall Curve*) [27]. Відповідно до нашого застосування поняття точності та повноти можна визначити, як наведено нижче.

Точність (*Precision*) – величина, яка вказує на частку правильно передбачених позитивних результатів серед усіх передбачених позитивних результатів (скільки з передбачених подібних описів та зображень насправді є подібними).

$$Precision = \frac{TP}{TP+FP}, \tag{3.1}$$

де *TP (True Positive)* – кількість правильно передбачених позитивних результатів (опис та зображення були насправді подібними і були передбачені як подібні);

*FP (False Positive)* – кількість хибно передбачених позитивних результатів (опис та зображення не були насправді подібними, але були передбачені як подібні).

Повнота (*Recall*) – величина, яка вказує на частку правильно передбачених позитивних результатів серед усіх можливих позитивних результатів (скільки з наявних подібних описів і зображень було передбачено як подібні).

$$Recall = \frac{TP}{TP+FN}, \tag{3.2}$$

де *FN (False Negative)* – кількість хибно не передбачених результатів (опис та зображення були насправді подібними, але не були передбачені як подібні) [27].

Будемо вважати, що опис та зображення *насправді подібні*, якщо вказане в тестовій колекції значення подібності між ними більше або рівне 0.5.

Будемо вважати, що опис і зображення *були передбачені як подібні*, якщо передбачене значення подібності між ними більше певного порогу або рівне йому.

Для побудови графіку кривої *Precision-Recall* обчислюють значення повноти та точності для деякої множини значень *Similarity (S)* (рисунок 3.1).

*Таблиця 3.1 – Значення для побудови графіку кривої Precision-Recall*

| | *S = 0.3* | *S = 0.4* | *S = 0.5* | *S = 0.6* | *S = 0.7* | *S = 0.8* | *S = 0.9* |
|---|---|---|---|---|---|---|---|
| *Precisions* | 0.35 | 0.29 | 0.32 | 0.5 | 0.5 | 0.43 | 0.75 |
| *Recalls* | 0.5 | 0.33 | 0.33 | 0.33 | 0.28 | 0.17 | 0.17 |

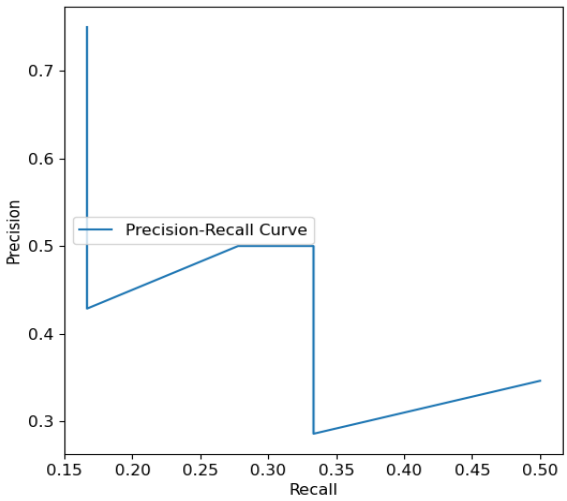



*Рисунок 3.1 – Precision-Recall Curve*

Після цього значення *AP* можна обчислити за формулою:

$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k), \quad (3.3)$$

де *Recalls(n)* = 0;

*Precisions(n)* = 1;

*n* – кількість різних значень порогу [28].

Оскільки ми використали значення *Similarity* від 0.3 до 0.9 з кроком 0.1, то отримана середня точність буде позначатися як *AP@[.30:.10:.90]* = 0.27. Наведемо функцію, яка обчислює значення метрики, в лістингу 3.1.

```
1. def calculate_ap(precisions, recalls, number_of_thresholds):
2.     sum_val = 0
3.     for k in range(0, number_of_thresholds):
4.         recalls_k1 = 0 if k + 1 == number_of_thresholds else
   recalls[k + 1]
5.         sum_val += (recalls[k] - recalls_k1) * precisions[k]
6.     return sum_val
```

*Лістинг 3.1 – Функція для обчислення середньої точності*

Отже, метрику *AP@[.30:.01:.90]* (зі значенням кроку 0.01) буде використано для оцінки точності алгоритму.

### 3.1.2 Root-Mean-Squared Error

Результатом роботи описаного у попередньому розділі алгоритму є чисельне значення подібності, а не бінарне (до бінарних значень «текст відповідає зображенню»/«текст не відповідає зображенню» можна перетворити отримане значення за певним порогом), тому іншим важливим показником під час оцінки якості програми може бути відхилення передбаченого значення подібності від реального: чим така похибка менша, тим вищою є точність програми.

У галузі машинного навчання широко використовується метрика *RMSE (Root-Mean-Squared Error)* для визначення середньоквадратичного відхилення отриманого числа від очікуваного [29]. Її можна застосувати й у нашому випадку.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(P_i - A_i)^2}{n}}, \quad (3.4)$$

де $n$ – кількість тестових прикладів;

$A_i$ – очікуване значення;

$P_i$ – передбачене значення.

Наведемо функцію для обчислення значень метрики в лістингу 3.2.

```
1. def get_rmse(truth_predicted_tuple_arr):
2.     squared_error_arr = np.array(list(map(lambda v: (v[0] -
   v[1])**2, truth_predicted_tuple_arr)))
3.     rmse_value = squared_error_arr.mean()
4.     return np.sqrt(rmse_value)
```

*Лістинг 3.2 – Функція для обчислення середньоквадратичного відхилення*

## 3.2 Структура тестової колекції

Для аналізу роботи алгоритму за згаданими раніше метриками необхідна тестова колекція, з якою й будуть порівнюватися передбачені алгоритмом значення. Визначмо структуру її об’єктів (лістинг 3.3).

```
{
    «img_path»:
«https://farm9.staticflickr.com/8204/8158895181_4ec7b51961_z.j
pg»,
    «short»: {
        «text»: «A row of benches»,
        «sim»: 0.93
    },
    «middle»: {
        «text»: «A row of old and dusted benches in the park
with cars behind it»,
        «sim»: 0.94
    },
    «long»: {
        «text»: «The mayor of the city declared that all the
old benches in the Green St. Will soon be cleaned and
painted»,
        «sim»: 0.75
```

```
    },
}
```

*Лістинг 3.3 – Приклад об’єкта з тестової колекції*

Кожен тестовий приклад включатиме ключі:

1) ***image_path*** – шлях до зображення, яке алгоритм аналізуватиме; усі зображення випадковим чином обрано з датасету COCO 2017 train/val – великої колекції зображень, які використовуються під час розробки алгоритмів для сегментації, опису картинок, пошуку об’єктів на них [30];

2) ***short*** – короткий описовий текст, який містить назви найголовніших об’єктів, зображених на картинці (одне із майбутніх застосувань програми – перевірка відповідності між картинкою та альтернативним підписом до неї, тому важливо визначити точність алгоритму при такому використанні; короткий влучний опис імітуватиме текст, який вказують в HTML-тегу *alt* при додаванні зображення на сторінку);

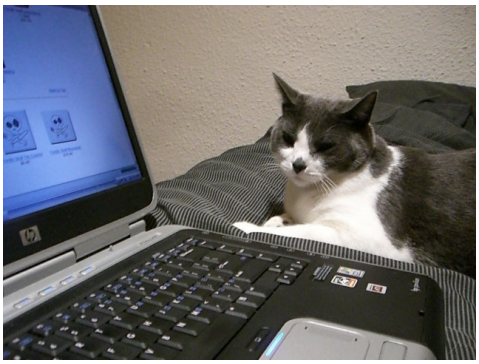
3) ***middle*** – речення середньої довжини, яке більш детально й повно описує зображення, містить синонімічні назви об’єктів з картинки (таким чином ми зімітуємо текст, який вказують в HTML-тегу *figcaption* для підпису картинок на сторінці);

4) ***long*** – досить довге речення або кілька речень, які майже не містять назв об’єктів, безпосередньо не стосуються зображення, але логічно з ним поєднуються; зазвичай для цього обиралися заголовки або речення зі статей, які мали з картинкою спільну тематику і до яких дане зображення було б доречною ілюстрацією (таким чином ми можемо перевірити якість програми, якщо їй необхідно визначити відповідність між зображенням та довгим уривком тексту, зрозуміти загальний сенс статті та картинки й порівняти їх);

5) ***sim*** – очікувані значення подібності між зображенням та кожним з трьох описових текстів.

Для тестування було складено 50 прикладів. Значення подібності опису та зображення визначено шляхом суб'єктивної оцінки схожості з урахуванням кількості спільних об'єктів та смислових зв'язків між ними.

Значення подібності між картинкою та текстом є високим, якщо текст містить усі основні деталі зображення та правильно описує дії об'єктів (рисунок 3.2).



```
«short»: {
    «text»: «A cat near a laptop on
a bed»,
    «sim»: 0.95
}
```

*Рисунок 3.2 – Приклад зображення з високою подібністю до тексту*

Значення подібності є середнім, якщо текст загалом відповідає зображенню, але в ньому бракує певної інформації або вона спотворена (рисунок 3.3).

```
«middle»: {
    «text»: «Woman feeds giraffe
from her hotel balcony in
Nairobi.»,
    «sim»: 0.6
}
```

*Рисунок 3.3 – Приклад зображення із середньою подібністю до тексту*

Значення подібності є малим, якщо текст дуже віддалено стосується зображення, не має з ним нічого спільного або суперечить йому (рисунок 3.4).

```
«long»: {
    «text»: «Elephantidae is a
family of large, herbivorous
proboscidean mammals
collectively called elephants
and mammoths.»,
    «sim»: 0.01
}
```

*Рисунок 3.4 – Приклад зображення із низькою подібністю до тексту*

## 3.3 Результати тестування

Описавши метрики та структуру тестової колекції, можна перейти до перевірки алгоритму. У попередньому розділі було розглянуто кілька різних способів для порівняння значення слів, визначення подібності між зображенням та описом, тому ми маємо протестувати кожен сценарій окремо, аби зробити висновки про переваги та недоліки підходів.

Наведемо пояснення скорочень, які будуть використані для зручності.

Способи порівняння значень двох слів:

1) ***fasttext*** – визначити векторне представлення обох слів за допомогою моделі *FastText,* обчислити подібність між векторами за косинусною мірою схожості;

2) ***wordnet*** – обчислити подібність слів за алгоритмом *Wu-Palme*r з використанням семантичного словника *WordNet*.

Способи порівняння двох списків об’єктів:

1) ***sim_area*** – порівняння списків об’єктів з урахуванням їхньої площі; важливість об’єкта пропорційна частці його площі від загальної площі зображення (див. формулу 2.5);

2) ***sim_n*** – порівняння списків об’єктів без урахування їхньої площі; усі об’єкти вважаються рівноважливими (див. формулу 2.4);

3) ***sim_sentence*** – порівняння списків об’єктів за допомогою косинусної міри схожості між векторами, що визначаються моделлю FastText для речень (див. формулу 2.6).

Вид опису до зображення:

1) ***short*** – короткий опис;

2) ***middle*** – опис середньої довжини;

3) ***long*** – довгий опис.

Метрики:

1) ***RMSE*** – Root-Mean-Squared Error (див. формулу 3.4);

2) ***AP*** – Average Precision (див. формулу 3.3).

Обчислені значення метрик наведемо в таблиці 3.2.

*Таблиця 3.2 – Обчислені значення метрик*

| **Спосіб порівняння значень двох слів** | **Спосіб порівняння двох списків об’єктів** | **Вид опису до зображення** | **Метрика** | **Обчислене значення** |
|---|---|---|---|---|
| – | *sim_sentence* | *short* | *RMSE* | *0.317* |
| | | | *AP* | *0.894* |
| | | *middle* | *RMSE* | *0.347* |
| | | | *AP* | *0.732* |
| | | *long* | *RMSE* | *0.236* |
| | | | *AP* | *0.345* |
| *fasttext* | *sim_area* | *short* | *RMSE* | *0.408* |
| | | | *AP* | *0.706* |
| | | *middle* | *RMSE* | *0.448* |

| | | | | |
|---|---|---|---|---|
| | | | *AP* | *0.505* |
| | | *long* | *RMSE* | *0.309* |
| | | | *AP* | *0.266* |
| | *sim_n* | *short* | *RMSE* | *0.439* |
| | | | *AP* | *0.622* |
| | | *middle* | *RMSE* | *0.441* |
| | | | *AP* | *0.565* |
| | | *long* | *RMSE* | *0.261* |
| | | | *AP* | *0.266* |
| *wordnet* | *sim_area* | *short* | *RMSE* | *0.374* |
| | | | *AP* | *0.751* |
| | | *middle* | *RMSE* | *0.431* |
| | | | *AP* | *0.596* |
| | | *long* | *RMSE* | *0.368* |
| | | | *AP* | *0.214* |
| | *sim_n* | *short* | *RMSE* | *0.413* |
| | | | *AP* | *0.667* |
| | | *middle* | *RMSE* | *0.421* |
| | | | *AP* | *0.652* |
| | | *long* | *RMSE* | *0.323* |
| | | | *AP* | *0.248* |

## 3.4 Аналіз результатів

Обчисливши значення метрик, можемо проаналізувати, який зі способів краще використовувати для визначення відповідності між зображенням та текстом.

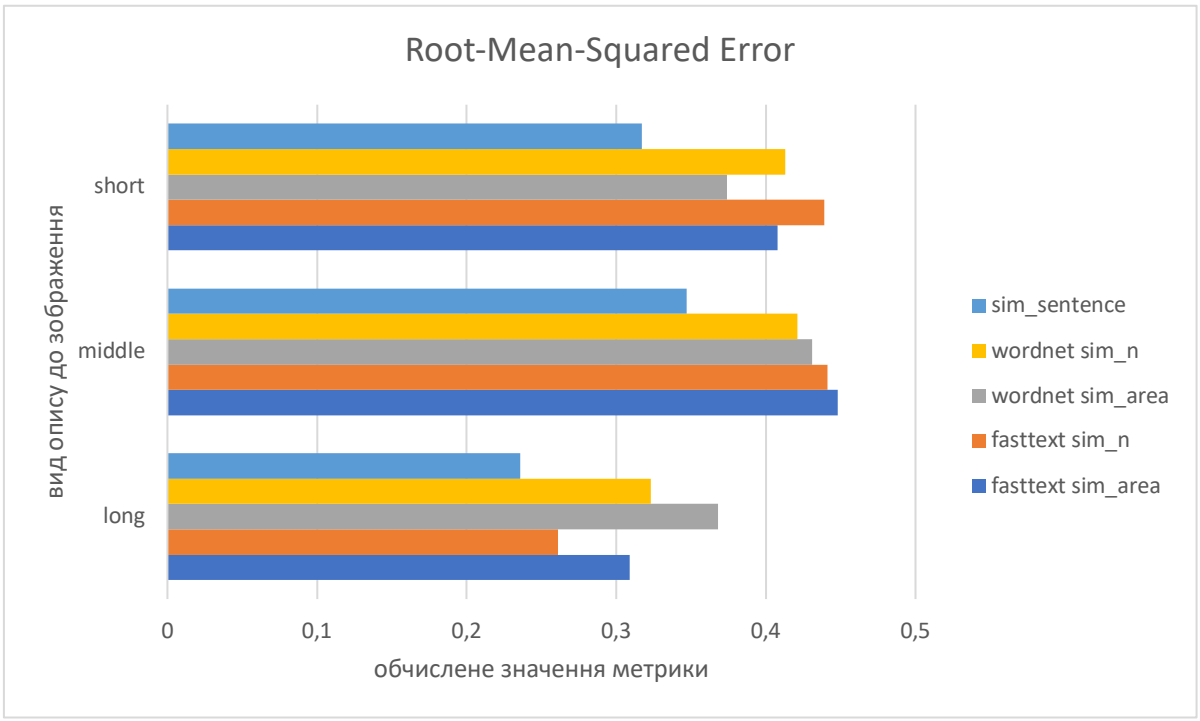



*Рисунок 3.5 – Діаграма з результатами обчислення середньоквадратичного відхилення*

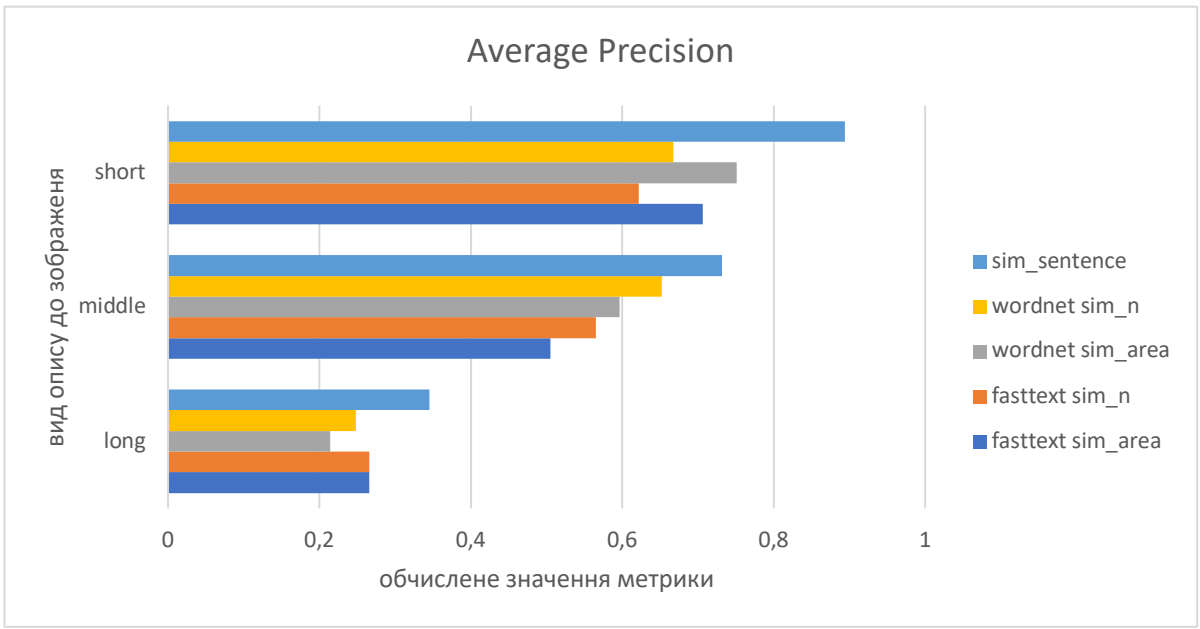



*Рисунок 3.6 – Діаграма з результатами обчислення середньої точності*

Порівнюючи значення метрик (рисунки 3.5, 3.6), можна помітити залежність між середньоквадратичним відхиленням та середньою точністю: їхні значення є обернено пропорційними в межах кожного виду опису (чим меншою є похибка, тим вищою є точність і навпаки). Також при впорядкуванні способів визначення подібності за зменшенням точності *AP* (відповідно, за збільшенням похибки *RMSE*) зберігається їхня послідовність. Це можна простежити в наведених нижче таблицях 3.3, 3.4, 3.5.

Така логічна кореляція між значеннями метрик свідчить про правильність проведених обчислень та їхню достовірність.

*Таблиця 3.3 – Упорядковані значення метрик для короткого опису*

| | *AP* | *RMSE* |
|---|---|---|
| *sim_sentence* | 0.894 | 0.317 |
| *wordnet sim_area* | 0.751 | 0.374 |
| *fasttext sim_area* | 0.706 | 0.408 |
| *wordnet sim_n* | 0.667 | 0.413 |
| *fasttext sim_n* | 0.622 | 0.439 |

*Таблиця 3.4 – Упорядковані значення метрик для середнього опису*

| | *AP* | *RMSE* |
|---|---|---|
| *sim_sentence* | 0.732 | 0.347 |
| *wordnet sim_n* | 0.652 | 0.421 |
| *wordnet sim_area* | 0.596 | 0.431 |
| *fasttext sim_n* | 0.565 | 0.441 |
| *fasttext sim_area* | 0.505 | 0.448 |

*Таблиця 3.5 – Упорядковані значення метрик для довгого опису*

| | *AP* | *RMSE* |
|---|---|---|
| *sim_sentence* | 0.345 | 0.236 |
| *fasttext sim_n* | 0.266 | 0.261 |
| *fasttext sim_area* | 0.266 | 0.309 |
| *wordnet sim_n* | 0.248 | 0.323 |
| *wordnet sim_area* | 0.214 | 0.368 |

### 3.4.1 Найефективніший алгоритм

Найвищою середньою точністю та найменшим середньоквадратичним відхиленням під час обчислення подібності між зображенням та ***всіма трьома видами описів*** до нього характеризується спосіб із застосуванням векторів речень (*sim_sentence*):

*Таблиця 3.6 – Значення метрик для алгоритму з використанням векторів речень*

| | *AP* | *RMSE* |
|---|---|---|
| *short* | 0.894 | 0.317 |
| *middle* | 0.732 | 0.347 |
| *long* | 0.345 | 0.236 |

Меншу точність продемонстрували інші способи визначення подібності. Розглянемо їх окремо для кожного виду опису, аби визначити альтернативні рішення.

### 3.4.2 Альтернативне рішення для короткого опису

Використання семантичного словника (*wordnet*) виявилось більш ефективним за векторне представлення (*fasttext*), що видно з даних, наведених у таблицях 3.7, 3.8.

*Таблиця 3.7 – Порівняння wordnet та fasttext за значенням AP для короткого тексту*

| | *AP* | | | *AP* |
|---|---|---|---|---|
| ***wordnet*** *sim_area* | 0.751 | > | ***fasttext*** *sim_area* | 0.706 |
| ***wordnet*** *sim_n* | 0.667 | > | ***fasttext*** *sim_n* | 0.622 |

*Таблиця 3.8 – Порівняння wordnet та fasttext за значенням RMSE для короткого тексту*

| | *RMSE* | | | *RMSE* |
|---|---|---|---|---|
| ***wordnet*** *sim_area* | 0.374 | < | ***fasttext*** *sim_area* | 0.408 |
| ***wordnet*** *sim_n* | 0.413 | < | ***fasttext*** *sim_n* | 0.439 |

Вищу ефективність методу з використанням семантичного словника можна пояснити тим, що в короткому тексті зазвичай вказуються точні назви об’єктів, зображених на картинці, що дозволяє ефективно визначати подібність об’єктів на зображенні та в тексті саме через ланцюги синонімів *WordNet*.

Нижча точність алгоритму з використанням векторного представлення для порівняння слів може обґрунтовуватись тим, що, на відміну від *WordNet*, який був детально розроблений науковцями вручну, *FastText* – нейронна мережа, якість якої залежить від навчальної колекції, кількості даних тощо. Отже, модель штучного інтелекту не може гарантувати такої ж точності, як і складений людьми словник.

Спосіб визначення подібності з урахуванням частки площі об’єкта на зображенні (*sim_area*) виявився більш ефективним за спосіб, де площа не мала значення (*sim_n*), що видно з даних, наведених у таблицях 3.9, 3.10.

*Таблиця 3.9 – Порівняння sim_area та sim_n за значенням AP для короткого тексту*

| | *AP* | | | *AP* |
|---|---|---|---|---|
| *wordnet* ***sim_area*** | 0.751 | > | *wordnet* ***sim_n*** | 0.667 |
| *fasttext* ***sim_area*** | 0.706 | > | *fasttext* ***sim_n*** | 0.622 |

*Таблиця 3.10 – Порівняння sim_area та sim_n за значенням RMSE для короткого тексту*

| | *RMSE* | | | *RMSE* |
|---|---|---|---|---|
| *wordnet* ***sim_area*** | 0.374 | < | *wordnet* ***sim_n*** | 0.413 |
| *fasttext* ***sim_area*** | 0.408 | < | *fasttext* ***sim_n*** | 0.439 |

Урахування площі об’єкта виявилося виправданим через те, що в короткому описі зазвичай вказуються назви найголовніших об’єктів на зображенні (таких, які займають найбільшу площу). Незначні ж деталі (відповідно об’єкти, що займають невелику площу) пропускаються. Отже, під час обчислення подібності саме наявність у тексті назв основних (великих) об’єктів з картинки має найбільшу вагу.

У випадку використання способу *sim_n* однакову вагу мають і дрібні об’єкти, про які часто не згадується в тексті, і основні об’єкти на зображенні, що збільшує відхилення та зменшує точність.

### 3.4.3 Альтернативне рішення для середнього опису

Використання семантичного словника (*wordnet*) виявилось більш ефективним за векторне представлення (*fasttext*), що видно з даних, наведених у таблицях 3.11, 3.12.

*Таблиця 3.11 – Порівняння wordnet та fasttext за значенням AP для середнього опису*

| | *AP* | | | *AP* |
|---|---|---|---|---|
| ***wordnet*** *sim_area* | 0.596 | > | ***fasttext*** *sim_area* | 0.505 |
| ***wordnet*** *sim_n* | 0.652 | > | ***fasttext*** *sim_n* | 0.565 |

*Таблиця 3.12 – Порівняння wordnet та fasttext за значенням RMSE для середнього опису*

| | *RMSE* | | | *RMSE* |
|---|---|---|---|---|
| ***wordnet*** *sim_area* | 0.431 | < | ***fasttext*** *sim_area* | 0.448 |
| ***wordnet*** *sim_n* | 0.421 | < | ***fasttext*** *sim_n* | 0.441 |

Результати можна обґрунтувати тим, що в тексті часто наводилися точні або синонімічні назви об’єктів з картинки, тому схожість між двома наборами об’єктів ефективно визначалась за допомогою *WordNet*.

Причина більшого відхилення результатів алгоритму з використанням векторного представлення слів аналогічна до наведеної вище.

Спосіб визначення подібності без урахування частки площі об’єкта на зображенні (*sim_n*) виявився більш ефективним за спосіб, де площа мала значення (*sim_area*), що видно з даних, наведених у таблицях 3.13, 3.14.

*Таблиця 3.13 – Порівняння sim_area та sim_n за значенням AP для середнього опису*

| | *AP* | | | *AP* |
|---|---|---|---|---|
| *fasttext* ***sim_n*** | 0.565 | > | *fasttext* ***sim_area*** | 0.505 |
| *wordnet* ***sim_n*** | 0.652 | > | *wordnet* ***sim_area*** | 0.596 |

*Таблиця 3.14 – Порівняння sim_area та sim_n за значенням RMSE для середнього опису*

| | *RMSE* | | | *RMSE* |
|---|---|---|---|---|
| *fasttext* ***sim_n*** | 0.441 | < | *fasttext* ***sim_area*** | 0.448 |
| *wordnet* ***sim_n*** | 0.421 | < | *wordnet* ***sim_area*** | 0.431 |

Оскільки в середньому описі вказуються не лише основні об’єкти, але й менші деталі, які також необхідно враховувати під час визначення подібності, то спосіб, який спирався на припущення про рівноважливість усіх об’єктів на зображенні (*sim_n*) продемонстрував кращі результати, а метод, який надавав значущості об’єктам пропорційно до частки їхньої площі (*sim_area*), – гірші.

### 3.4.2 Альтернативне рішення для довгого опису

Порівняння значень слів за допомогою їхнього векторного представлення (*fasttext*) виявилося більш ефективним, ніж використання семантичного словника (*wordnet*), що видно з даних, наведених у таблицях 3.15, 3.16.

*Таблиця 3.15 – Порівняння wordnet та fasttext за значенням AP для довгого опису*

| | *AP* | | | *AP* |
|---|---|---|---|---|
| ***fasttext*** *sim_area* | 0.266 | > | ***wordnet*** *sim_area* | 0.214 |
| ***fasttext*** *sim_n* | 0.266 | > | ***wordnet*** *sim_n* | 0.248 |

*Таблиця 3.16 – Порівняння wordnet та fasttext за значенням RMSE для довгого опису*

| | *RMSE* | | | *RMSE* |
|---|---|---|---|---|
| ***fasttext*** *sim_area* | 0.309 | < | ***wordnet*** *sim_area* | 0.368 |
| ***fasttext*** *sim_n* | 0.261 | < | ***wordnet*** *sim_n* | 0.323 |

Довгий текст безпосередньо не пов’язаний із зображенням, тому описує об’єкти непрямо та не містить точних назв. Відповідно, використання семантичного словника вже не є настільки ж ефективним, як у попередніх випадках.

Натомість більшу точність дає метод з використанням векторного представлення слів. Як було зазначено вище, модель машинного навчання *FastText* націлена розрізняти приховані смисли слів, тому дозволяє знайти подібність навіть у далеких за значенням або написанням словах. Через цю особливість метод є більш ефективним.

У випадку довгого опису спосіб визначення подібності без урахування частки площі об’єкта (*sim_n*) на зображенні також виявився точнішим за спосіб, де площа мала значення (*sim_area*), що видно з даних, наведених в таблицях 3.17, 3.18.

*Таблиця 3.17 – Порівняння sim_area та sim_n за значенням AP для довгого опису*

| | *AP* | | | *AP* |
|---|---|---|---|---|
| *wordnet* ***sim_n*** | 0.248 | > | *wordnet* ***sim_area*** | 0.214 |
| *fasttext* ***sim_n*** | 0.2661 | > | *fasttext* ***sim_area*** | 0.2660 |

*Таблиця 3.18 – Порівняння sim_area та sim_n за значенням RMSE для довгого опису*

| | *RMSE* | | | *RMSE* |
|---|---|---|---|---|
| *wordnet* ***sim_n*** | 0.323 | < | *wordnet* ***sim_area*** | 0.368 |
| *fasttext* ***sim_n*** | 0.261 | < | *fasttext* ***sim_area*** | 0.309 |

Менше відхилення при використанні цього способу можна пояснити тим, що текст опосередковано стосувався зображення, тому однаково важливим було враховувати подібність і малих, і великих об’єктів на зображенні до об’єктів з тексту.

## 3.5 Узагальнення результатів

Результати аналізу значень метрик узагальнимо в таблиці 3.19.

*Таблиця 3.19 – Узагальнена таблиця за результатами тестування*

| **Вид опису до зображення** | **Найкращий спосіб для визначення подібності між текстом та зображенням** | | | **Альтернативний спосіб для визначення подібності між текстом та зображенням** | | | |
|---|---|---|---|---|---|---|---|
| | **Спосіб** | **AP** | **RMSE** | **Спосіб порівняння значень двох слів** | **Спосіб порівняння двох списків об’єктів** | **AP** | **RMSE** |
| *short* | *sim_sentence* | 0.894 | 0.317 | *wordnet* | *sim_area* | 0.751 | 0.374 |
| *middle* | *sim_sentence* | 0.732 | 0.347 | *wordnet* | *sim_n* | 0.652 | 0.421 |
| *long* | *sim_sentence* | 0.345 | 0.236 | *fasttext* | *sim_n* | 0.266 | 0.261 |

Наведемо остаточні реалізації алгоритмів в лістингах 3.4, 3.5, 3.6.

```
# Function to calculate similarity value and determine
# if the description corresponds to the image
# using sentence vectors
def sim_sentence(image_path, image_description, sim_threshold=
config.THRESHOLD_TRUE):
    text_objects = ner.get_objects_from_text(image_description)
    [image_objects, _] = yolo.get_objects_from_image(image_path)
    similarity_value = similarity_utils.calculate_similarity_with_sentences(
        ner.process_string_list(image_objects), # lowercase, split objects' classes
        ner.process_string_list(text_objects)) # lowercase text objects
    return similarity_value, similarity_value >= sim_threshold
```

*Лістинг 3.4 – Повна функція для визначення відповідності між зображенням та текстом за допомогою векторів речень (див. формулу 2.6)*

```
# Function to calculate similarity value and determine
# if the description corresponds to the image
# using word similarity function
# and TAKING into consideration the AREA of the objects
# word_similarity_func can be either fasttext word similarity or
# wordnet word similarity described in the text above
def sim_area(image_path, image_description, word_similarity_func, sim_threshold=
config.THRESHOLD_TRUE):
    text_objects = ner.get_objects_from_text(image_description)
    [image_objects, image_objects_areas] = yolo.get_objects_from_image(image_path)
    similarity_value = similarity_utils.calculate_similarity_area(
        ner.process_string_list(image_objects),
        ner.process_string_list(text_objects),
        image_objects_areas,
        word_similarity_func)
    return similarity_value, similarity_value >= sim_threshold
```

*Лістинг 3.5 – Повна функція для визначення відповідності між зображенням та текстом з урахуванням площі (див. формулу 2.5)*

```
# Function to calculate similarity value and determine
# if the description corresponds to the image
# using word similarity function
# and NOT TAKING into consideration the AREA of the objects
# word_similarity_func can be either fasttext word similarity or
# wordnet word similarity described in the text above
def sim_n(image_path, image_description, word_similarity_func, sim_threshold=
config.THRESHOLD_TRUE):
    text_objects = ner.get_objects_from_text(image_description)
    [image_objects, _] = yolo.get_objects_from_image(image_path) # ignore areas
    similarity_value = similarity_utils.calculate_similarity_n(
        ner.process_string_list(image_objects),
        ner.process_string_list(text_objects),
        word_similarity_func)
    return similarity_value, similarity_value >= sim_threshold
```

*Лістинг 3.6 – Повна функція для визначення відповідності між зображенням та текстом без урахування площі (див. формулу 2.4)*

Отже, у результаті тестування ми з’ясували, що для визначення подібності між зображенням та текстом найкраще використовувати метод, у якому для порівняння об'єктів на картинці та в описі застосовується косинусна міра схожості між векторами речень.

Алгоритм можна достатньо ефективно використовувати для визначення відповідності лише між зображенням та коротким або середнім описом до нього, адже значення точності при таких застосуваннях є достатньо високими. Однак низькою є ефективність методу для аналізу подібності довгого тексту та зображення, оскільки в такому випадку недостатнім є аналіз подібності між окремими об'єктами, а важливості набуває також контекст, у якому вони знаходяться, та інші чинники.

Альтернативним рішенням може бути використання семантичного словника та нейронної мережі для порівняння об'єктів на зображенні та в тексті, хоча точність результатів буде меншою. Ці способи також не є ефективними для визначення відповідності між картинкою та довгим описом.

## 3.6 Переваги та недоліки розроблених алгоритмів

Переваги підходу визначати відповідність між зображенням та текстом на основі подібності об’єктів:

1) нескладний у реалізації, проте може бути ефективним для визначення відповідності між зображенням з простими об’єктами та невеликим за обсягом описом;

2) для реалізації може бути поєднано найефективніші алгоритми з галузей обробки зображень та природної мови.

Обмеження підходу:

1) враховується подібність між назвами об'єктів, але не беруться до уваги зв’язки, які існують між об’єктами. Це призводить

до неправильних результатів у випадку, якщо текст і зображення містять ті самі об’єкти, проте описують протилежні відношення між ними;

2) труднощі викликають абстрактні поняття, які можна визначити в тексті, але складно виокремити на зображенні, оскільки на сьогодні зусилля розробників спрямовані на пошук ефективних методів визначення саме фізичних об'єктів на картинці;

3) іншим недоліком є те, що група з кількох об'єктів на зображенні сприймається як множина розрізнених об'єктів (наприклад,{“Птах”, Птах”, “Птах”, “Птах”}, {“Людина”, Людина”, Людина”} тощо), у тексті ж зазвичай використовуються іменники на позначення сукупності (“зграя”, “гурт” тощо). Через зазначену невідповідність між об’єктами у таких випадках великим є значення похибки під час визначення подібності.

# ВИСНОВОК

У ході роботи було проаналізовано різні алгоритми для визначення об'єктів на зображенні. Розуміння переваг та недоліків кожного з них дозволило обрати найефективніший для використання у власній розробці. Окрім цього, розглянуто методи тегування частин мови, розпізнавання іменованих сутностей у тексті та з’ясовано обмеження, пов’язані з ефективним розв’язанням перелічених задач. З метою порівняти семантику об'єктів визначено способи векторного представлення слів з використанням моделей машинного навчання, а також запропоновано застосувати алгоритм, розроблений для лексичної бази даних.

Досліджено можливість застосовувати алгоритми з розпізнавання об'єктів на зображенні та в тексті, визначення іменованих сутностей, порівняння значень слів для роботи з українською мовою. На жаль, на сьогодні бракує готових ефективних розробок, що мають вищеокреслену специфіку. Це відкриває широкі можливості для вітчизняних програмістів, що задіяні у сфері машинного навчання.

На основі обраних під час дослідження засобів складено 5 варіантів алгоритмів для визначення відповідності між зображенням та текстом. Аби перевірити точність їхньої роботи було обрано метрики, які широко застосовуються в машинному навчанні, та складено невелику за обсягом тестову колекцію, яка дозволила оцінити якість методів при обчисленні подібності між картинкою та трьома видами описів до неї.

У ході аналізу значень метрик, отриманих у результаті тестування, було з’ясовано, що найточнішим з усіх виявився метод визначення подібності між об’єктами тексту та зображення із застосуванням векторів речень. Також визначено альтернативні способи для кожного з видів описів з обґрунтуванням можливих причин, які могли призвести до похибок певного алгоритму.

Головним результатом роботи стали функції для перевірки відповідності між зображенням та описом, які було складено на основі алгоритмів, що під час тестування виявилися найефективнішими з точки зору визначення подібності.

Подальшими шляхами розвитку може бути покращення алгоритму з урахуванням згаданих у роботі обмежень розробленого рішення.

# СПИСОК ЛІТЕРАТУРИ


1. Rrohan Arrora. Convolutional implementation of the sliding window algorithm [Електронний ресурс]. – Режим доступу: https://medium.com/ai-quest/convolutional-implementation-of-the-sliding-window-algorithm-db93a49f99a0
2. P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/1312.6229.pdf
3. R. Girshick, J. Donahue, T. Darrell, J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/1311.2524.pdf
4. Rohith Gandhi. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
5. Ross Girshick. Fast R-CNN [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/1504.08083.pdf
6. S. Ren, K. He, R. Girshick, J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/1506.01497.pdf
7. J. Redmon, S. Divvala, R. Girshick, A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/1506.02640.pdf
8. J. Redmon, A. Farhadi. YOLO9000: Better, Faster, Stronger [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/1612.08242.pdf

9. J. Li, A. Sun, J. Han, C. Li. A Survey on Deep Learning for Named Entity Recognition [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/1812.09449.pdf
10. N. Perera, M. Dehmer, F. Emmert-Streib. Named Entity Recognition and Relation Detection for Biomedical Information Extraction [Електронний ресурс]. – Режим доступу: https://www.frontiersin.org/articles/10.3389/fcell.2020.00673/full
11. G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, C. Dyer. Neural Architectures for Named Entity Recognition [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/305334469_Neural_Architectures_for_Named_Entity_Recognition
12. Neural Model of Text [Електронний ресурс]. – Режим доступу: https://www.coursera.org/learn/machine-learning-duke/lecture/Y4rtA/neural-model-of-text
13. Amit Chaudhary. A Visual Guide to FastText Word Embeddings [Електронний ресурс]. – Режим доступу: https://amitness.com/2020/06/fasttext-embeddings/
14. P. Bojanowski, E. Grave, A. Joulin, T. Mikolov. Enriching Word Vectors with Subword Information [Електронний ресурс]. – Режим доступу: https://aclanthology.org/Q17-1010.pdf
15. M. Shenoy.K, Dr.K.C.Shet, Dr. U. D. Acharya. A new similarity measure for taxonomy based on edge counting [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/1211.4709.pdf
16. J. Redmon, A. Farhadi. YOLOv3: An Incremental Improvement [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/1804.02767.pdf

17. A. Bochkovskiy, C.-Y. Wang, H.-Y. M. Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/2004.10934.pdf
18. X. Long, K. Deng, G. Wang, Y. Zhang, Q. Dang, Y. Gao, H. Shen, J. Ren, S. Han, E. Ding, S. Wen. PP-YOLO: An Effective and Efficient Implementation of Object Detector [Електронний ресурс]. – Режим доступу: https://arxiv.org/pdf/2007.12099.pdf
19. SpaCy Trained Models & Pipelines [Електронний ресурс]. – Режим доступу: https://spacy.io/models
20. FastText. Library for efficient text classification and representation learning [Електронний ресурс]. – Режим доступу: https://fasttext.cc/
21. Кульчицький І.М., Романюк А.Б., Харів Х.Б. Розроблення Wordnet-подібного словника української мови [Електронний ресурс]. – Режим доступу: https://science.lpnu.ua/sisn/all-volumes-and-issues/volume-673-2010/rozroblennya-wordnet-podibnogo-slovnika-ukrayinskoyi
22. Остапчук, С. І. Метод визначення ключових слів в аудіопотоці: дис., д. т. н., проф. Терейковський І. А.; Нац. техн. ун-т України «Київський політехнічний інститут ім. І. Сікорського» — Київ, 2021. — 89 с. — Бібліогр.: с. 24. [Електронний ресурс]. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/45942/1/Ostapchuk_magistr.pdf
23. Part of Speech Tagging [Електронний ресурс]. – Режим доступу: https://www.coursera.org/learn/probabilistic-models-in-nlp/lecture/VbnrA/part-of-speech-tagging
24. UDPipe API [Електронний ресурс]. – Режим доступу: http://lindat.mff.cuni.cz/services/udpipe/api-reference.php
25. Adrian Rosebrock. Intersection over Union (IoU) for object detection [Електронний ресурс]. – Режим доступу:

https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

26. Matthew Honnibal. Embed, encode, attend, predict: The new deep learning formula for state-of-the-art NLP model [Електронний ресурс]. – Режим доступу: https://explosion.ai/blog/deep-learning-formula-nlp#entailment
27. Jonathan Hui. mAP (mean Average Precision) for Object Detection [Електронний ресурс]. – Режим доступу: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173
28. Ahmed Fawzy Gad. Evaluating Object Detection Models Using Mean Average Precision (mAP) [Електронний ресурс]. – Режим доступу: https://blog.paperspace.com/mean-average-precision/
29. James Moody. What does RMSE really mean? [Електронний ресурс]. – Режим доступу: https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e
30. COCO Common Objets in Context [Електронний ресурс]. – Режим доступу: https://cocodataset.org/#home