КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСАШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота
на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

Застосування методів машинного навчання для кластеризації та індексації відео

Виконав студент 4-го курсу

Рудь Максим Максимович _____

(підпис)

Науковий керівник:
асистент кафедри МІ
Бобиль Богдан Володимирович _____

(підпис)

Консультант:
доцент кафедри математичної інформатики, кандидат фіз.-мат. наук
Дерев'янченко Олександр Валерійович

(підпис)

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

Студент _____
(підпис)

Київ – 2022

РЕФЕРАТ

(Обсяг роботи 42 сторінка, 14 ілюстрацій, 13 джерел посилання)

КЛАСТЕРИЗАЦІЯ-ІНДЕКСАЦІЯ ВІДЕО, МАТРИЦЯ, ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, НЕЙРОННА МЕРЕЖА, ВЕКТОРНЕ ПРЕДСТАВЛЕННЯ

Об’єктом роботи є алгоритм кластеризації відео.

Метою роботи є розробка нейронної мережі для кластеризації відео та виміряння її точності.

Методи розроблення: «Життєвий цикл моделі»(DSP)[1], створення та трансформація даних для навчання(кластеризації), створення моделі «Комп’ютерного Зору», аналіз результатів кластеризації відео за допомогою відповідних метрик. Інструменти розроблення: текстовий редактор Visual Studio Code 1.56.2, web-сервіс на базі «хмарних» технологій Google Collab, мова програмування Python 3.9., відкриті програмні бібліотеки Numpy, Pandas, Tensorflow.

Результати роботи: вивчено існуючі алгоритми та підходи у вибраному напрямку; реалізовано модель для кластеризації відео; підібрано архітектуру згорткової нейронної мережі для векторних представлень відео

Розроблений та проаналізовані алгоритми можуть використовуватися для кластеризації та подальшої класифікації відеофайлів. Також може використовуватися для полегшення створення каталогів відеофайлів для користувача

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки.

Зважаючи на те, що задача кластеризації та класифікації статичних зображень є популярною та досить вивченою задачею, ці алгоритми можуть водночас використовуютися для кластеризації відео файлів за допомогою нейронних мереж.

Хоча в цій роботи ми оперуємо окремими кадрами відео, методи, які ми використовуються для кластеризації, сегментації чи індексації відео може працювати з різними об’єктами. Це може бути, як обмежений набір зображень зібраних із кадрів відео чи супровідний аудіофайл, який відповідно аналізується за семантичним змістом або використання попердніх об’єктів у тандемі із додатковими текстовими рядками, які додають семантичного змісту зображенням і аудіофайлам

Часто використовують змішані моделі із готовими «з коробки» моделями класифікації, для сегментації відео, яке потім передається алгоритму кластеризації для поділу відео на логічні частини чи каталогізації. Незважаючи на відносну схожість алгоритмів між собою, головною проблемою залишаються знаходження найкращої метрики «відстані» між ВП зображень, відео, текстів тощо для перевірки семантичної схожості між різними файлами. В цій роботі розглядається складова роботи із відео, а саме адаптація моделі для кластеризації зображень під відео.

Актуальність теми. За останніми дослідженнями до 2025 року хмарне сховище даних у всьому світі становитиме понад 200 зеттабайт[2].В зв'язку із великою популярністю хостингових, стрімінгових сервісів та сервісів зберігання даних збільшується кількість даних, яка продукується їх користувачами.

Тож організація такої великої кількості даних різного формату потребувала створення алгоритмів для роботи з нею, які б передбачали її зберігання, зокрема

каталогізацію для зручного користування. Не зважаючи на те, що відеофайли, які хоч і мають різні формати, можуть мати зручні для розуміння користувача чи комп'ютера метадані, проте серед великого об'єму даних – це рідкість. Тому проблема розмічення даних є актуальною і потребує ефективних рішень.

Ручне розмічення відео може бути надто трудомістким заняттям, що створює потребу в алгоритмах для принаймні часткового розмічення або ж розділення рухомих зображень за категоріями чи семантичним змістом.

Ці алгортми використовуються всіма популярними сервісами відеохостингами такими як: YouTube, Instagram, Twitch, Dailymotion, тощо.

Мета й завдання роботи. Мета роботи це створення алгоритму кластеризації для заданих даних, які складаються із відеофайлів із рухомими зображеннями.

Під час створення були поставлені такі завдання для вирішення:

- обрати метод кластеризації відеофайлів або адаптації алгоритму для зображень під відеофайли.
- Реалізувати метод кластеризації відеофайлів
- Селекція метрик для оцінки схожості зображень на кадрах відео
- спроектувати та навчити нейронну мережу для створення векторного представлення рухомих зображень;
- оцінити отримані результати.

Об’єкт і методи дослідження або розроблення. Алгоритм кластеризації відеофайлів та метрики схожості відеофайлів були об’єктом розроблення та дослідження відповідно.

Методами дослідження були: алгоритмізація, проектування моделі нейронної на основі принципів життєвого циклу моделей Машинного Навчання до яких входять нейронні мережі, аналіз результатів кластеризації.

Можливі сфери застосування. Розроблений алгоритм може бути використаний у сфері Великих Даних, яка потребує організації великої кількості відео. Може використовуватися, як допоміжний інструмент, для кращого користувацького досвіду на відеохостингах, стрімінгових сервісах та корпоративних цілей.

Позначення

НМ – Нейронна Мережа

МО – Мапа ознак зображення

ЗНМ - Згорткові нейронні мережі

ВП – Векторне представлення

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

Загалом можемо поділити методи індексації та кластеризації відео на декілька частина:

- класична кластеризація – позначення правдивих категорій зображень не використовуються моделлю, а тільки для перевірки фінальної кластеризації
- класифікація з напівучителем – лише частина даних розмічена відповідно до класу зображення,
- сегментація без учителя та навчання через виділення особливостей, найчастіше ці частини працють в тандемі й рідко можна знайти використання тільки одного методу без іншого.

Методи, які використовуються класичну кластеризацію з використанням автокодувальників даних, звичайних евклідових метрики та використовують алгоритм к-середнього[3, 4, 5, 6]. Вони мають спільну стріктуру, яка складається із згорткової нейронної мережі, яка виділяє низькорівневі та деякі семантичні ознаки зображень МО. Після чого перетворює отримані значення ВП. «Наївний» підхід не передбачає подальших змін у векторономy представленні, що не дає достатньої ефективності. Практичні дослідження вказують на те, що такий метод не враховує ознаки зображення високого рівня, які насправді розрізняють відмінності між зображеннями, адже згадані методи дуже залежні від початкової ініціалізації після отримання МО зображень.

Алгоритми наступного покоління, які частково наслідують структур попередніх розробок, але намагаються новітніми методами виділити семантичні особливості, тобто ознаки зображення високого рівня для того, щоб виправити недоліки методів згаданих вище. Ця робота натхненна цими останніми алгоритмами і можуть бути адаптовані для кластеризації відео. Останні

алгоритми використовують свої методи мінімізації семантичної різниці між найближчими сусідами для ВП зображення[7] або адаптація відомих метрик для цієї задачі, наприклад, метрика «взаємної інформації»[8].

Зазвичай ці алгоритми мають початковий етап, який називають «претекствою задачею», сенс її полягає в тому, щоб при створенні ВП мінімузувати відстані між модифікаціями зображення без втрати великої кількості інформації та оригінальним зображенням. Найпопулярнішим готовим рішенням для цієї початкової задачі є розробка від дослідників компанії Google під назвою «Простий фреймворк для контрастного навчання репрезентації зображення»[9]. Ця розробка складається із звичайного кодувальника зображення, який отримує на вхід пари, які складаються із оригінального та модифікованого зображення. Модифікаціями тут ми називаємо повороти, зміна колірної палітри, обрізання, виділення контурів, чорно-біле зображення тощо. Ця частина послуговується такими головними принципами.

• Композиція з кількох операцій модифікації даних має вирішальне значення у визначенні контрастних завдань прогнозування, які дають інформативні ВП даних. Крім того, контрастне навчання без учителя має переваги від більш потужного збільшення даних, ніж навчання з учителем.

• Введення нелінійного перетворення, яке навчається нейронною мережею, між представленням і контрастною втратою істотно покращує якість засвоєних представлень.

• Створення векторного представлення є найбільш ефективним із використанням перехресної ентропії, як цільової функції мінімізації

• Контрастивне навчання має переваги від більших розмірів пакетів і більш тривалого навчання в порівнянні з його аналогом із учителем. Як і

навчання з учителем, контрастне навчання має переваги від більш глибоких і широких нейронних мереж.

Розділ 2. Теоретичні відомості

Так як практична частина містить у собі велику кількість термінів, які потребують роз'яснення, ми опишемо головні треміни чи інструменти, які використовуються в цій роботі.

Розділ 2.1 Загальна Будова Нейронних мереж

Розділ 2.1.1 Пряме розповсюдження

Спершу про нейронні мережі. Концептуально нейронні мережі надихалися моделлю імітації нейрону мозку людину – Перцептрону. Яка отримує на вхід лінійну комбінацію вхідних параметрів, а на виході отримуємо її, яка дається на вхід східчатій функції. Сучасні нейронні мережі зазвичай це багатошарові Перцептрони, які окрім того замість східчастої функції викоритовують сигмоїдальні диференційовні функції.

Нейронні мережі є інструментом, який є продовження лінійних моделей для регресії та класифікації, які мали такий вигляд:

$$y(x, w) = f(\sum_{i=1}^{M} w_i \varphi(x)), \qquad (2.1)$$

Де, f – нелінійна активаційна функція і тотожним відображенням у випадку регресії. Наша мета полягає у тому, щоб узагальнити цю модель, припустивши, що базисні функції φ(x) залежать від параметрів, а потім коригувати ці параметри разом з коефіцієнтами $\{w_j\}$ під час навчання. Звичайно, існує безліч способів побудови параметричних не лінійних базових функцій. Нейронні мережі використовують базисні функції, які мають форму описану вище, так що кожна базисна функція сама по собі є нелінійною функцією лінійної комбінації входів, де коефіцієнти лінійної комбінації є адаптивними параметрами. Це призводить до базової моделі нейронної мережі, яка може бути описана рядом

функціональних перетворень[10]. Візьмемо модель нейронної мережі з трьох шарів: вхідного, «прихованого» та вихідного.

Перший шар отримуватиме вхідні параметри $x_1 \dots x_D$, з них ми будуємо лінійні комбінації:

$$a_j = \sum_{i=1}^{D} w_{ji}^1 x_i + w_{j0}^1,$$

$$z_j = h(a_j)$$

Де $j = 1 \dots M$, а верхній індекс (1) вказує, що відповідний параметр знаходиться в першому шарі мережі, перший член суми називають вагою, а другий зміщення. Результат суми називають активаціями, після отримання суми виконується трансформація за домогою диференційовної нелінійної *функції активації* $h(a)$. Вибір функції активації залежить від вхідних даних та поставленої задачі, так для задачі регресії зазвичай використовують функції з тотожним відображенням, для класифікації використовується зазвичай логістичні сігмоїди.

$$y_k = \sigma(a_k),$$

$$\sigma(a) = \frac{1}{1+\exp(-a)},$$

Але для багатокласових задач, на кінцевому шарі нейронної мережі найчастіше використовують функцію активації

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)},$$

Відповідно як і для першого перетворення вхідних даних маємо перетворення прихованого шару, яке передається на вихід нашої нейронної мережі є

аналогічним. Якщо ми хочемо створити функцію з результатом проходження проходження по нейронній мережі – отримуємо

$$y_k(x,w) = \sigma(\sum_{j=0}^{M} w_{kj}^2 h(\sum_{i=0}^{D} w_{ji}^1 x_i)),$$

якщо узагальнити для N-кількості шарів, маємо

$$y_k(x,w) = \sigma(\sum_{j_N=0}^{M} w_{k_N j_N}^N h(\dots w_{k_2 j_2}^2 h(\sum_{j_1=0}^{D} w_{k_1 j_1}^1 x_{j1})) \dots)).$$

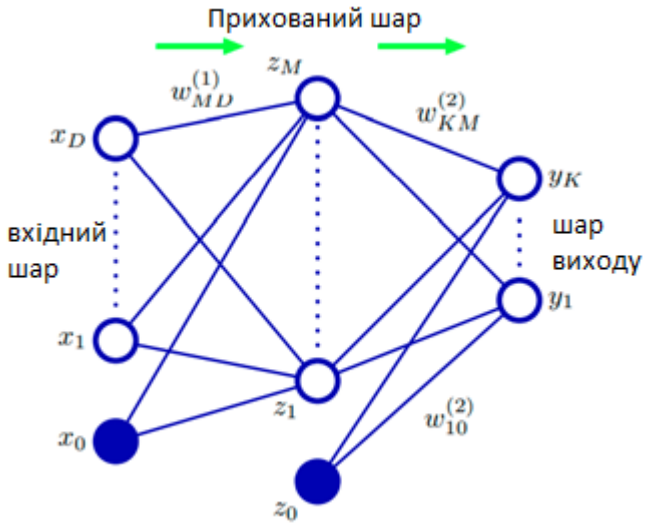


Рис. 1. Схема нейронної мережі, яка складається з трьох шарів.

Розділ 2.1.2 Тренування нейронної мережі

Наступний етап роботи нейронної мережі – це навчання. Під час процесу навчання ваги нейронної мережі оновлюються й оптимізуються відносно функції помилки, яка відповідно пібирається під конкретно задану задачу.

Якщо ми маємо набір даних із N спостережень $X = \{x_1, x_2, \dots., x_N\}$ і цільове значення $t = \{t_1, \dots, t_N\}$, ми можемо побудувати таку задачу максимізації правдоподібності

$$p(\boldsymbol{t}|X, \beta, w) = \prod_{n=1}^{N} p(t_n|x_n, w, \beta).$$

Так, як зазвичай для вирішення задачі потрібно мінімізувати функцію помилки, то цю задачу можна перевести, у випадку регресії, у вирішення задачці мінімізації функції нижче:

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2 .$$

У випадку із класифікацією з умови, якщо припустимо маємо класи $C_1, \dots, C_K$, де K – загальна кількість класів, маємо:

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk} ln y_{nk} + (1 - t_{nk})\ln(1 - y_{nk})\},$$

де y_{nk} означає $y_k(\boldsymbol{x_n}, \boldsymbol{w})$.

Після підрахунку функції помилок є оптимізація ваг, яка в результаті має поступово мінімізовувати функцію помилок. Існують різні алгоритми оптимізації, які передбачають оновлення вагів на кожній ітерації, проте найчастіше використовується метод стохастичного градієнтного спуску. Важливо зазначити, що функція помилок має бути диференційовною, це особливо важливо при використанні даного методу оновлення вагів. Формула оновлення вектора вагових коефіцієнтів на r-тій ітерації:

$$w_{r+1} = w_r - \eta\nabla E_n w_r,$$

Де $\eta > 0$ називається швидкістю навчання. Цей параметр є одним із гіперпараметрів нейронної мережі від якого залежить з якою швидкістю і чи взагалі потрапить у глобальний мінімум цільової функції. Якщо підбирати його емпіричним шляхом, то достатньо розпочати з менших значень, близьких до одиниці.

Наступними етапами є валідація і тестування мережі, ці етапи є зазвичай оціночними для того щоб оцінити точність результату тренування нейронної мережі на даних, на яких вона ще не тренувалася, щоб підтвердити, наприклад

для класифікації зображення, інваріантність точності класифікації від початкової даних на тренуванні.

Розділ 2.1.3 Згорткові Нейронні мережі

Згорткові нейронні мережі(ЗНМ) – це алгоритм глибокого навчання, який отримує на вхід зображення, виділяє її особливості, що дає змогу відрізнити їх одне від одного. Перевагою над іншими алгоритмами є менша кількість ручної обробки зображення конкретно під задану задачу – це додає гнучкості моделі для вивчення більшої кількості класів.

Варто зазначити, що використання попереднього варіанту моделі – просте переведення зображення в масив пікселів – може тільки нашкодити, адже тоді модель буде шукати семантичну близькість між окремими пікселями, а не ознаками високого рівня. Для вирішення цієї проблеми були існують згорткові шари, які виконують задачу субдискретизації, виділення ознак зображення різного рівня.

ЗНМ може успішно фіксувати просторові та часові залежності в зображенні за допомогою застосування відповідних фільтрів. Архітектура краще відповідає набору даних зображення завдяки зменшенню кількості задіяних параметрів і можливості повторного використання ваг. Іншими словами, мережу можна навчити краще розуміти сутність зображення.

Ці поняття реалізовані в свертних нейронних сетях за допомогою трьох механізмів: 1) локальних рецептівних полів, 2) розділення ваг 3) виділення підвибірок. Структура згорткової мережі показана на Рис. 1. В шарах згортки елементи об'єднуються в групи, які ще називаються картами ознак. Кожен елемент на карті ознак отримує вхідні дані тільки з невеликої області

зображення, а всі елементи в карти ознак використовують одні й ті самі значення ваг.

Наприклад, карта ознак може складатися зі 100 елементів, що утворюють мережу 10 х 10, при чому кожен елемент приймає вхідні дані з фрагментів зображень розміром 5 х 5 пікселів. Таким чином, вся карта карта ознак має 25 регульованих вагів і один регульований параметр зміщення. Вхідні значення з фрагмента утворюють лінійну комбінацію за допомогою вагі і зміщення, а результат перетворюється за допомогою сигмоїдальної нелінійної функції за формулою (2.1). В результаті всі елементи в карті визнають один і той самий образ, але в різних місцях вхідного зображення.

Завдяки розділенню ваг обчислення активації цих елементів рівномірно згортці інтенсивності пікселів зображення з ядром, що містять вагові параметри. Якщо вхідне зображення зміщено або має інші модифікації, активація карти ознак буде зміщена або модифікавана на одну і ту же величину, але в іншому не зміниться. Це забезпечує інваріантність вихідних даних до зміщення чи невеликих поворотів зображення і представленням вхідного зображення. Оскільки для створення ефективної моделі нам зазвичай необхідно виявляти безліч визнань, у слоі свертки буде, як правило, кілька карт визнань, які мають свій власний набір вагів і параметрів зміщення.

Наступним шаром у моделях є шари підвибірок. Для кожної карти ознак у шарі згортки існує площина елементів у шарі підвибірки, і кожен елемент приймає вхідні дані з невеликого рецептивного поля у відповідній карті ознак шару згортки. Ці елементи виконують роль вибірку. Наприклад, кожен елемент вибірки може приймати вхідні дані з області 2 х 2 у відповідній карті ознак і обчислювати середнє значення цих входів, помножене на адаптивну вагу з додаванням адаптивного параметра зсуву, а потім перетворюватися за

допомогою сигмоїдальної нелінійної функції активації. Рецептивні поля вибираються так, щоб вони були безперервними і такими, що не

перекриваються, так що в шарі підвибірки буде вдвічі менше рядків та стовпців порівняно з шаром згортки. Таким чином, відповіді елементів у шарі підвибірки будуть відносно мало чутливими до малих зсувів зображення у відповідних областях вхідного простору.[10]

Після створення підвибірок отримані ваги передаються на повнозв'язні шари нейронної мережі, як було згадано в минулому розділі Рис 1. За схемою описаною вище ми виконується пряме розповсюдження і тренування нейронної мережі для класифікації(Рис. 2).

Це є дуже важливою частиною нашої роботи, адже це є основою для виявлення ВП зображення, яке нам знадобиться далі.

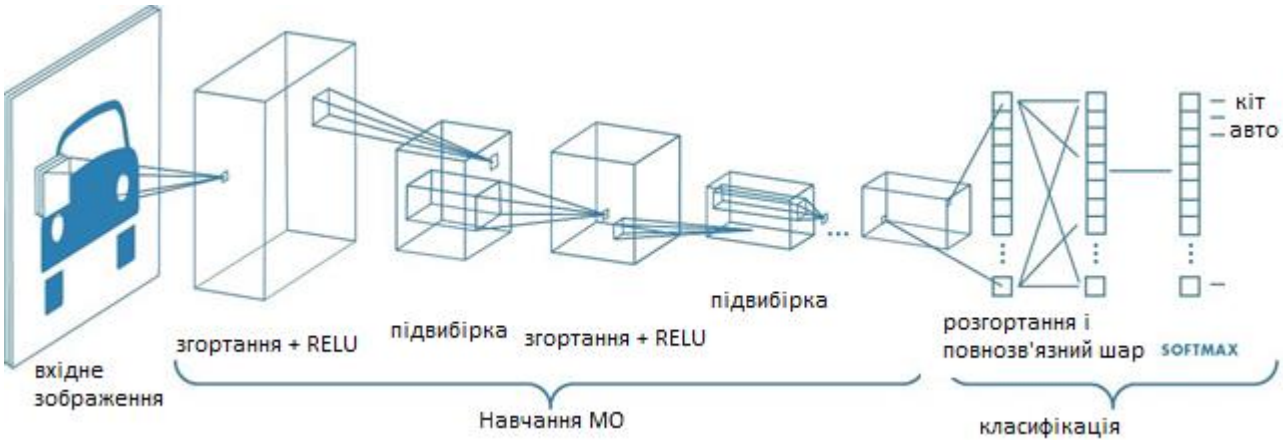


Рис. 2 Загальна схема умовної ЗНМ, яка відповідно скаладається з ліва на право: комбінації шарів згорток та підвибірок, повнозв’язної нейронної мережі для подальшої класифікації

Розділ 2.2 Кластеризація

Також варто розглянути таку частину науки про дані як Кластеризація. Кластеризація або кластерний аналіз це задача групування даних в такому вигляді, що дані з однієї групи, яку називають кластером більш схожі(в якомусь сенсі) аніж дані з іншої групи.

Існує велика кількість методі кластеризації даних, але концепція залишається однією. Нехай існує вектора x_i який належить множині векторів $\{x_1 \dots . x_n\}$, які належать до простору R^m, де m – розмірність вектора. За заданою метрикою $p(x_i, x_j)$ маємо визначити групу для цього вектора. При цьому групи не перетинаються й елементи якої за метрикою p знаходяться найближче одне до одного.

Існують різні методи кластеризації такі як Центроїдні моделі, наприклад K-середніх, Статистичні моделі, Групові моделі метрики, які визначатимуть відстань між векторними представленнями. Зазвичай використовують наступні метрики:
Евклідова відстань:

$$d_{euc}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2},$$

Манхетенська відстань:

$$d_{man}(x, y) = \sum_{i=1}^{n} |(x_i - y_i)|,$$

Де x та y відповідно вектори розмірності n.

В нашій роботі й роботах згаданих в минулих розробках також для наближення зображень часто використовують метрику косинусу подібності, приклад на Рис 3.:

$$\cos(\theta) = \frac{x*y}{\|x\|\|y\|}$$

Косинусна подібність має застосування, що виходить за рамки абстрактної математики. Вимірювання використовується в процесах аналізу даних, пошуку інформації та відповідності тексту. Після того, як вектори присвоюються властивостям змінної, вимірювання стає цінним інструментом для розуміння подібності між об’єктами.

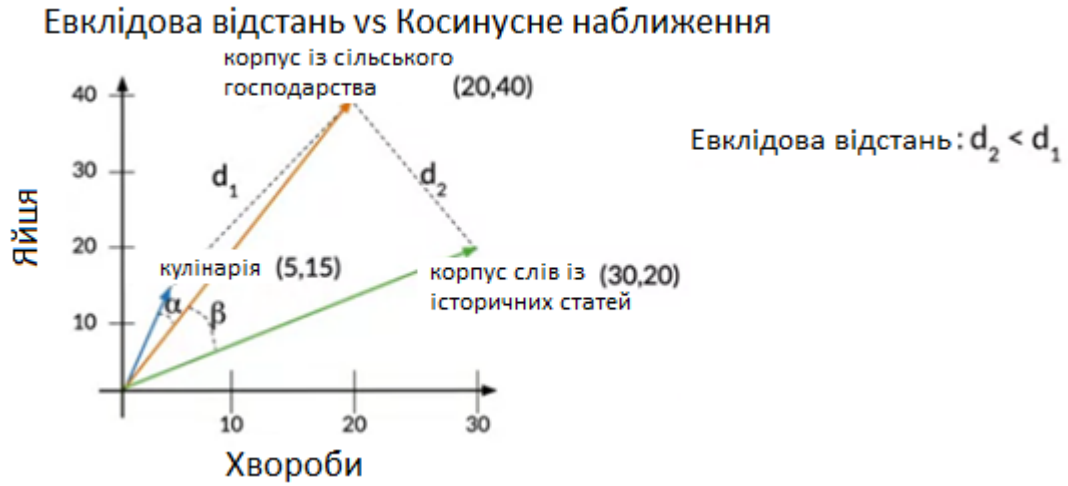


Рис.3 Приклад порівняння векторних представлень для збірок слів різної тематики, можемо помітити, що метрика косинусу подібності краще зближає вектори за сенсом

Розділ 3 Методика Кластеризації Відео

Розділ 3.1 Підготовка датасету

Першим етапом є підготовка даних для кластеризації. В цій роботі ми використовували дані у відкритому доступі, переважно відео з YouTube-shorts – сегменту відеохостингу YouTube із короткими відео не більше хвилини. В цій задачі використовуємо 5 категорій по 10 відео. Приклади кадрів із відео, які було зібрані для роботи можемо побачити на Додатку А. Категорії використані для кластеризації:

- Собаки
- Коти
- Птахи
- Зупинки на зміну колес на авто-перегонах
- Зйомка з камери, яка переслідує бігунів на короткому забігу.

Заздалегідь ми підібрали так відео, які можуть містити різну кількість контексту семантичної інформації, щоб перевірити якість кластеризації в різних умовах. Також об’єкти, за якими ми хочемо кластеризувати відео маю часто рухають або взаємодіють об'єктами. Наприклад в різних частинах відео зміни колес автомобіля на перегонах присутні різні об'єкти, або вони зникають. На одному із відео з собками ми бачимо взаємодії з різними предметами. В наших експериментах ми перевіримо чи це суттєво вплине на наш метод кластеризації, враховуючи ті аугментації над кадрами про які ми говоритимемо пізніше.

Розділ 3.2 Загальна Архітектура Алгоритму кластеризації відео

Як було згадано у розділі про аналіз існуючих методів задача кластеризації починається із претекстової задачі, яка намагається за допомогою нейронної

мережі Φ_Θ бути вирішена і результаті ми отримуємо ВП зображень.Так як ми кластеризуємо відео. Ми розбиваємо відео на кадри, беремо тільки кадри з кроком Δh - гіперпараметр нашої моделі.

Отримавши пакет кадрів для кластеризації ми передаємо його на претекстову задачу. В нашій моделі ми використовуємо готове рішення SimCLR[11] для виділення початкового ВП для відео.

Наприклад, коли НМ Φ_Θ прогнозує параметри перетворення афінного перетворення. Різні афінні перетворення одного і того ж зображення призведуть до різних вихідних прогнозів для Φ_Θ. Це робить представлення вивчених ознак менш придатними для семантичної кластеризації, де представлення ознак мають бути інваріантними до трансформацій зображень. Щоб подолати цю проблему, ми ставимо претекстове завдання τ, щоб також мінімізувати відстань між зображеннями X_iта їх збільшеннями $T[X_i]$, яке можна виразити як:

$$\min_\theta d\big(\Phi_\theta(X_i), \Phi_\Theta(T[X_i])\big),$$

Щоб зрозуміти, чому зображення з подібними ознаками високого рівня відображаються ближче один до одного за допомогою Φ_θ, ми робимо наступні спостереження.

По-перше, претекстове завдання «витягує» семантичну інформацію з вихідного зображення за допомого Φ_θ.

По-друге, оскільки Φ_θ має обмежену ємність, йому доводиться відкидати зі свого входу інформацію, яка не є передбачуваною для завдання високого рівня претекстової задачі.

Наприклад, малоймовірно, що Φ_θ може вирішити задачу розрізнення екземплярів, кодуючи лише колір або один піксель із вхідного зображення. В

результаті зображення з подібними високорівневими характеристиками будуть лежати ближче один до одного в просторі ВП Φ_θ.

Ми робимо висновок, що для отримання семантично значущих ознак можна використовувати претекстні завдання з навчання представлень. Після цього спостереження ми використаємо ВП для подальшої обробки

На наступному етапі ми можемо використати два методи: подальше коригування найближчих сусідів за семантикою за допомогою методу «Семантичної кластеризації при адаптації найближчих сусідів» або використати звичайний метод визначення центрів кластерів за допомогою алгоритму K-середнього. Незважаючи на простоту й зрозумілість другого методу, в результаті експерименті на кластеризації одиничних зображень маємо дегенерацію кластерів, коли один кластер домінує над іншими на моменті визначення їх меж. Для запобігання цьому використовується перший метод згаданий вище – «Семантична кластеризація при адаптації найближчих сусідів».

Сенс методу полягає в тому, що ми спершу знаходимо найближчих сусідів, які за результатами експериментів[8] маємо, що переважна кількість зображень потрапляє у відповідний семантичний кластер Рис 4.

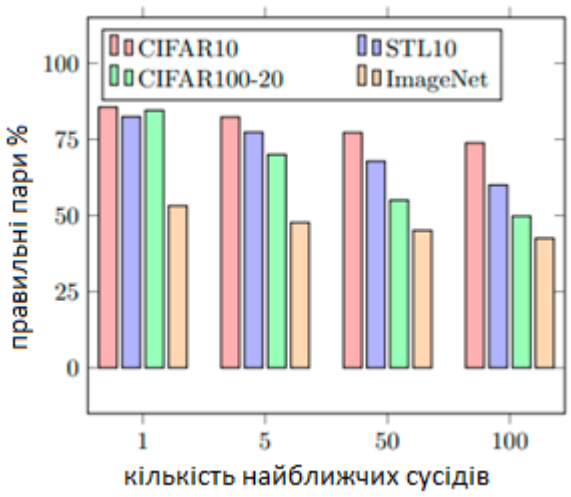


Рис 4. Відсоткове відношення правильної належності зображення до семантичного класу[8]

Наступні кроки зближують найближчих сусідів за допомогою мінімізації скалярного добутку між екземплярами найближчих сусідів.

Розділ 3.3 Метод виділення представлення зображення SimCLR

Розділ 3.3.1 Загальний опис архітектури

Як було згадано в розділі про кластеризацію для успішного вирішення завдання нам потрібно створити ВП нашого відео

Звичайно, ми могли б просто перетворювати кадри відео у вектор розміром $H \times W \times 3$, де H та W, висота та ширина зображення ба навіть, якщо ми використаємо пряме розповсюдження моделі Згорткової Нейронної Мережі результати будуть дегенеративними[7], адже близькість на просторі ВП зображень буде залежна від кольору областей підвибірок отриманих під час проходження через нейронну мережу.

Відповідно для запобігання цих залежностей алгоритм SimCLR використовує метод навчання без учителем разом із використанням додаткових аугментацій зображення.

Концепція методу полягає в тому, щоб наближувати ВП оригінальних зображень до їх аугментації. Схема роботи алгоритму може бути представлена на Рис. 5:

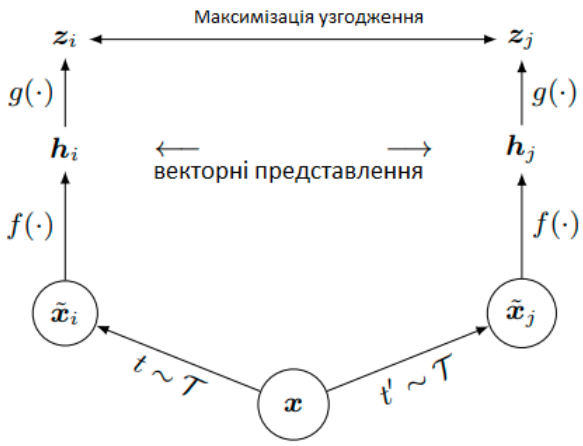


Рис. 5 Схема роботи алгоритму SimCLR

Два окремих оператора модифікації даних відбираються з одного сімейства аугментацій ($t \sim T$ і $t' \sim T$) і застосовуються до кожного представлення для отримання двох корельованих представлень. Мережа кодувальника f(·) і проекційна головка g(·) навчаються максимізувати узгодження з використанням контрастних втрат. Після завершення навчання ми викидаємо проекційну головку g(·) і використовуємо кодер f(·) і представлення h для наступних завдань.

Тож ми можемо розділити модель на чотири головні компоненти:

- Модуль агментацій кадрів x , який створює дві серії випадкових аугментації для кожного кадру відео з визначеним інтервалом s - $\tilde{x}_i$ та $\tilde{x}_j$ відповідно, яку ми визначаємо як позитивну пару, яку потрібно наближати. У цій роботі ми послідовно застосовуємо три прості аугментації: випадкове обрізання з наступним поверненням до початкового розміру, випадкові спотворення кольору та випадкове розмиття за Гауссом.
- Наступною частиною моделі є кодувальник, який виділяє векторні представлення кадрів, для спрощення використовуємо модель

VGG19[12]. Загальну архітектуру VGG19 можемо розглянути в Додатку Б. В результаті отримуємо $h_i = f(\tilde{x}_i) = VGG(\tilde{x}_i)$ де $h_i \in R^d$. При цьому для зменшення розмірності в кінці використовуємо шар GlobalAveragePooling. Перетворивши всі кадри відео ми знаходимо середнє зачення початкового ВП, яке стає загальним векторним представленням відео.

- НМ поменше проекційна головка, що відображає уявлення в просторі, де застосовується метрика контрастної втрати. Для цього використовується багатошаровий персептрон так що $z_i = g(h_i) = w^{(2)}\sigma\left(w^{(1)}h_i\right)$ де σ це нелінійна функція $ReLU$. Схему проекційної головки, можемо побачити на Рис. 6.
- Останньою частиною є розрахунок контрастної втрати для зближення семантично близьких пар, які мають різні аугментації і відпвідно віддалити інші представлення

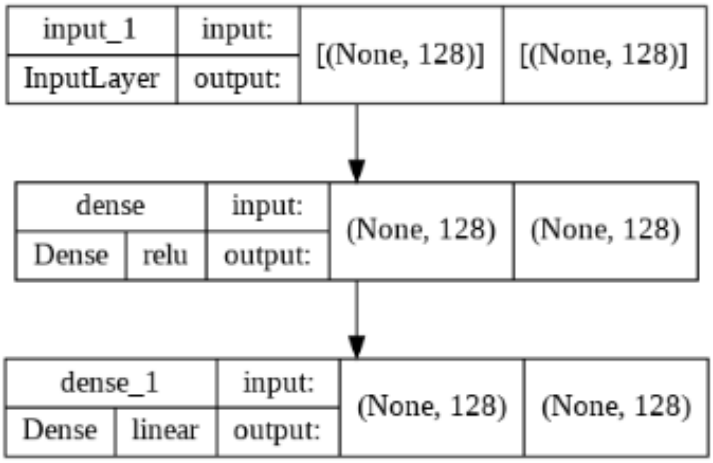


Рис. 6 Схема Багатошарового персептрону з одним прихованим шаром проекційної головки

Ми випадковим чином відбираємо міні-пакет із N(гіперпараметр моделі) прикладів і визначаємо завдання контрастного прогнозування на парах

модифікованих зображень, отриманих з міні-пакету, що призводить до 2N ВП. Ми не вибираємо негативні(несхожі семантичні) представлення явно. Інші частина пакету, а саме $2(N-1)$ ми вважаємо негативними.

Нехай $sim(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u}^T \boldsymbol{v}}{\|\boldsymbol{u}\|\|\boldsymbol{v}\|}$ означає косинусну близькість між L_2 нормалізованими векторами $\boldsymbol{u}$ та $\boldsymbol{v}$. Функція помилки для позитивної пари представлень формулою виглядає ось так:

$$L_2 = -\log \frac{\exp\left(\frac{sim(z_i, z_j)}{\tau}\right)}{\sum_{k=1}^{2N} 1_{[k \neq i]} \exp\left(\frac{sim(z_i, z_k)}{\tau}\right)}, \quad (3.1)$$

де $1_{[k \neq i]} \epsilon \{0,1\}$ це індикатор, як дорівнює 1, якщо $k \neq i$ і τ описує коефіцієнт «температури». Фінальна функція помилок рахується для всіх позитивних для пари (i,j) та (j,i) разом.

Розділ 3.3.2 Аугментація кадрів відео

Для визначення найефективніших аугментацій розробниками SimCLR були проведені експерименти й найефективнішими виявилися:

- Обрізання
- Зміна розміру
- Повороти
- Спотворення Кольору(колірне тремтіння)
- Розмиття Гауса

Хоча збільшення даних широко використовується як у контрольованому, так і в неконтрольованому навчанні репрезентації, воно не розглядається як систематичний спосіб визначення завдання контрастного прогнозування. Багато існуючих підходів визначають завдання контрастного прогнозування шляхом зміни архітектури. Наприклад, [13] досягають прогнозування від глобального до

локального через обмеження сприйнятливого поля в архітектурі мережі, тоді як інші досягають передбачення сусідніх переглядів за допомогою фіксованої процедури поділу зображення та мережі агрегації контексту. Ми показуємо, що цієї складності можна уникнути, виконуючи просте випадкове обрізання (зі зміною розміру) цільових зображень, який створює сімейство прогнозних завдань, що включає два вищезгаданих, як показано на Рис. 7.

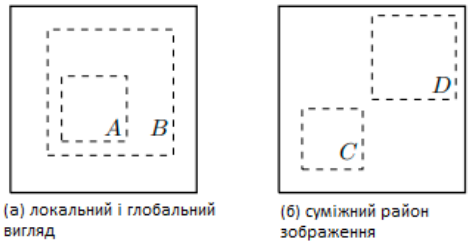


Рис. 7 Два види обрізання, яка беруть участь претекстовій задачі для семантичного наближення.

Цей простий вибір дизайну зручно відокремлює завдання прогнозування від інших компонентів, такі як архітектура нейронної мережі. Більш широкі завдання контрастного прогнозування можна визначити шляхом розширення сімейства аугментацій та їх стохастичне компонування. Приклад аугментації кадрів та список аугментацій можемо побачити на Рис. 8 та Рис. 12 відповідно

Рис. 8 Приклади аугментацій зображення

Прості перетворення колірної палітри або зміни колірної гістограми не досягають такого успіху для семантичного засвоєння. Натомість афінні перетворення в колірному просторі впливають на результат більш суттєво, тому ми виконали імплементацію цього перетворення в рамках фреймворку TensorFlow Рис. 9

```python
class RandomColorAffine(layers.Layer):
    def __init__(self, brightness=0, jitter=0, **kwargs):
        super().__init__(**kwargs)

        self.brightness = brightness
        self.jitter = jitter

    def get_config(self):
        config = super().get_config()
        config.update({"brightness": self.brightness, "jitter": self.jitter})
        return config

    def call(self, images, training=True):
        if training:
            batch_size = tf.shape(images)[0]

            brightness_scales = 1 + tf.random.uniform(
                (batch_size, 1, 1, 1), minval=-self.brightness, maxval=self.brightness
            )
            jitter_matrices = tf.random.uniform(
                (batch_size, 1, 3, 3), minval=-self.jitter, maxval=self.jitter
            )

            color_transforms = (
                tf.eye(3, batch_shape=[batch_size, 1]) * brightness_scales
                + jitter_matrices
            )
            images = tf.clip_by_value(tf.matmul(images, color_transforms), 0, 1)
        return images
```

Рис. 9 Імплементація «колірного тремтіння»

Розділ 3.4 Модель посилення кластеризації відео

Після отримання векторних представлень для відео ми знаходимо найближчих сусідів для кожного ВП. Формалізувавши можемо описати це так: для кожного екземпляру відео $X_i \in \mathcal{D}$ ми знаходимо його $\mathcal{K}$ найближчих сусідів у векторному просторі Φ_θ. Ми визначаємо множину $\mathcal{N}_{X_i}$, як найближчі представлення в множині $\mathcal{D}$.

Наступною ціллю є створити невеличку НМ Φ_η, яка ініціалізується векторним представленням X_i та найближчими сусідами $\mathcal{N}_{X_i}$. Нехай маємо множину

кластерів $C = \{c_1, \dots, c_M\}$. Задачею цієї НМ є мапінг ВП за допомогою функції sofmax на вектор

$$\Phi_\eta^{c_1}, \dots, \Phi_\eta^{c_M},$$

Де $\Phi_\eta^{c_k}(X_i)$ – ймовірність належності ВП кластеру c_k область значень [0,1], M – справжня кількість кластерів.

Для навчання ваг нейронної мережі цільова функція помилок, яку ми мінімізуємо виглядає ось так:

$$\Lambda = -\frac{1}{|\mathcal{D}|}\sum_{X\epsilon\mathcal{D}}\sum_{k\epsilon N_X}\log(\Phi_\eta(X) * \Phi_\eta(k)) + \lambda\sum_{c_k\epsilon}\Phi_\eta^{\prime c_k}\log(\Phi_\eta^{\prime c_k}),$$

$$\Phi_\eta^{\prime c_k} = \frac{1}{|\mathcal{D}|}\sum_{X\in\mathcal{D}}\Phi_\eta^{c_k}(X).$$

Де, (*) – оператор скалярного добутку між векторами результатів тренування на мережі Φ_η. Перший член рівняння створений для кращого зближення між ВП зображення і його сусідами.

Варто зауважити, що скалярний добуток добуток буде максимальним, коли прогнози є унітарним кодом, тобто показує максимальну вірогідність на одному кластері і приписаними до одного кластера, тобто послідовним. Щоб Φ_η н приписував всіх вибірок одному кластеру, ми включаємо другий член (рівняння 3.3), який рівномірно розподіляє передбачення по кластерам C.

Якщо розподіл ймовірності на кластери C відомий заздалегідь, що не відповідає нашому випадку, то ми можемо використати Розходження Кульбака-Лейблера.

Також важливо зауважити, що точна кількість кластерів у C взагалі невідома. Однак, для валідації нашої кластеризації ми вибираємо C, що дорівнює кількості кластерів основної істини. На практиці має бути можливим отримати приблизну оцінку кількості кластерів. Зазвичай цей вибір буде

залежати не від моделі, а до прикладної області для якої використовується кластеризація.

РОЗДІЛ 4 Імплементація

Розділ 4.1 Попередня обробка

Наступною частиною роботи після збору датасету відеофайлів варто створити розбивку кожного відео на розкадровку (Рис. 10). Для цього ми використовуємо відкриту бібліотеку «Python-OpenCV».

```
videos_train = np.empty((train_count, max, frame_shape[0], frame_shape[1], frame_shape[2]), np.dtype('uint8'))
videos_test = np.empty((test_count, max, frame_shape[0], frame_shape[1], frame_shape[2]), np.dtype('uint8'))
labels_test = np.empty((test_count, 1), np.dtype('object'))

for dir in os.listdir('Data'):
  file_count = 0
  for file_name in os.listdir('Data' + '/' + dir):
    cap = cv2.VideoCapture('Data' + '/' +  dir + '/' + file_name)

    if max == 0:
      frameCount = int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) // step
    else:
      frameCount = max

    # frameWidth = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    # frameHeight = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    video = np.empty((frameCount, frame_shape[0], frame_shape[1], 3), np.dtype('uint8'))

    if cap.isOpened() == False:
      print('Cap is not open')
      break

    frames_dir_path = 'Data' + '/' + dir + '/' + file_name + '_frames'
    check_or_remove(frames_dir_path)

    count = 0
    frame_count = 0
    while(cap.isOpened()):

      ret, frame = cap.read()
      if ret == True:

        if count % step != 0:
          count += 1
          continue

        if max == frame_count:
          break

        video[frame_count] = cv2.resize(frame, (frame_shape[0], frame_shape[1]), interpolation= cv2.INTER_LINEAR)

        frame_file_name = file_name + '_frame' + str(count) + '.jpg'
        print(f'Creating: {frame_file_name}')
        cv2.imwrite(os.path.join(frames_dir_path, frame_file_name), frame)

        count += 1
        frame_count += 1
```

Рис. 10 Збір даних в тренувальний і тестувальний датасети

Користувач може обрати сам, яке відношення тренувальних та тестувальних даних використовується. В нашому випадку ми беремо дані з відношенням 80 на 20.

Розділ 4.2 Аугментація кадрів

Окремо розглянемо імплементацію аугментацій зображення. Для цього ми використовуємо послідовний шар із фреймворку TensorFlow, який випадково буде робити відповідні аугментації: зміна розмірності, випадковий поворот, випадковий наближення та обрізання та створення ефекту колірного тремтіння(Рис 11.)

```
def get_augmenter(min_area, brightness, jitter):
    zoom_factor = 1.0 - math.sqrt(min_area)
    return keras.Sequential(
        [
            keras.Input(shape=(image_size, image_size, image_channels)),
            layers.Rescaling(1 / 255),
            layers.RandomFlip("horizontal"),
            layers.RandomTranslation(zoom_factor / 2, zoom_factor / 2),
            layers.RandomZoom((-zoom_factor, 0.0), (-zoom_factor, 0.0)),
            RandomColorAffine(brightness, jitter),
        ]
    )
```

Рис. 11 Приклад коду, зі списком послідовних аугментацій зображення.

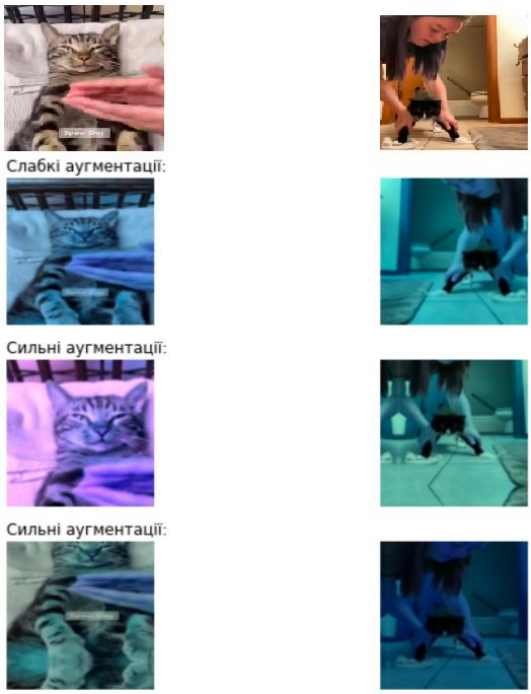


Рис. 12 Приклади аугментації для першого кадру з датасету відео

Розділ 4.3 Отримання векторних представлень

Наступним етапом є отримання векторних представлень, для цього використовуємо модель SimCLR згадану в попередніх розділах. Для кожного відео створюємо дві аугментації, які модель наближає одна до одної на просторі ВП. Кожне відео ми використовуємо, як окремий пакет кадрів, яке проходить через початковий кодувальник, а для вихідних векторів ми знаходимо середнє значення. Далів ми отримуємо готове ВП через проекційну головку (Рис. 13)

```
def train_step(self, videos):
    augmented_video_1 = self.contrastive_augmenter(videos, training=True)
    augmented_video_2 = self.contrastive_augmenter(videos, training=True)
    with tf.GradientTape() as tape:
        features_1 = self.encoder(augmented_video_1, training=True)
        features_2 = self.encoder(augmented_video_2, training=True)

        projections_1 = np.mean(self.projection_head(features_1, training=True))
        projections_2 = np.mean(self.projection_head(features_2, training=True))
        contrastive_loss = self.contrastive_loss(projections_1, projections_2)
    gradients = tape.gradient(
        contrastive_loss,
        self.encoder.trainable_weights + self.projection_head.trainable_weights,
    )
    self.contrastive_optimizer.apply_gradients(
        zip(
            gradients,
            self.encoder.trainable_weights + self.projection_head.trainable_weights,
        )
    )
    self.contrastive_loss_tracker.update_state(contrastive_loss)
```

Рис. 13 Тренування зближення векторних представлень

Після проведення тренування на нашому датасеті ми можемо використати ваги НМ для обчислення векторних представлень зображення. Наступним кроком є знаходження найближчих сусідів. В результаті обчислень на тестовому датасеті отримали точність 58.65%, що є непоганим значенням для початку.

Отримані представлення та найближчі сусіди надалі стають вхідними даними для наступної нейронної мережі, яка має покращити наближення найкращих пар відео (Рис. 14).

Отримавши максимальні вірогідності приналежності до відповідних кластерів можемо визначити остаточну точність для тестового датасету з відео.

Результатом правильного розподілу за категоріями отримали середнє значення 65.71% потраплянь в кластер (Рис. 14)

```
def create_clustering_learner(clustering_model):
    anchor = keras.Input(shape=input_shape, name="anchors")
    neighbours = keras.Input(
        shape=tuple([k_neighbours]) + input_shape, name="neighbours"
    )
    neighbours_reshaped = tf.reshape(neighbours, shape=tuple([-1]) + input_shape)
    anchor_clustering = clustering_model(anchor)
    neighbours_clustering = clustering_model(neighbours_reshaped)
    neighbours_clustering = tf.reshape(
        neighbours_clustering,
        shape=(-1, k_neighbours, tf.shape(neighbours_clustering)[-1]),
    )
    similarity = tf.linalg.einsum(
        "bij,bkj->bik", tf.expand_dims(anchor_clustering, axis=1), neighbours_clustering
    )
    similarity = layers.Lambda(lambda x: tf.squeeze(x, axis=1), name="similarity")(
        similarity
    )
    model = keras.Model(
        inputs=[anchor, neighbours],
        outputs=[similarity, anchor_clustering],
        name="clustering_learner",
    )
    return model
```

Рис. 14 Модель семантичного наближення.

В результаті кластеризації ми отримуємо такі розподіли правильного присвоєння категорії для відео з датасету:

- Бігуни – 78.01 %
- Птахи – 63.12 %
- Зупинки на перегонах 75.34%
- Собаки – 58.76 %
- Коти – 53.32%

Розділ 4.4 Можливі подальші покращення

Варто зазначити, що датасет для отримання ВП є досить невеликим і різноманітним в плані семантичного контексту незважаючи на розділення за категоріями.

Для покращення варто спробувати виконати тюнинг гіперпараметрів моделі – підбір ефективних гіперпараметрів моделі, які покращать середній результат кластеризації.

Наступним покращеня може бути використання файнтюнінгу, для ще кращої цілісності передбачень. Тобто створити копію отриманої моделі, які мають ті самі параметри й ваги, які отримані з тренування на початковому датасеті, але без останнього вихідного шару. Відповідно в процесі «файнтьюнингу» на тестовому датасеті ми покращуємо вже існуючі ваги та тренуємо коефіцієнти на останньому шарі. Також варто спробувати збільшити кількість епох навчання.

ВИСНОВКИ

У роботі було запропоновано та реалізовано алгоритм для кластеризації відео, в цій роботі також були представлені та розроблені моделі НМ для виконання поставленого завдання.

Технологією розробки була обрана мова Python, яка містить бібліотеку Tensorflow Keras. В рамках кваліфікаційної роботи було виконано поставлені завдання, а саме:

- Досліджено і проаналізовано відповідну літературу та останні дослідження кластеризації відео.
- Адаптовано модель виділення особливостей та кластеризації для зображень під відео
- Сформовано та проаналізовано набори даних для навчання моделей.
- Запропоновані моделі імплементовано та натреновано за допомогою TensorFlow Keras.
- Наведено результати роботи моделей та алгоритму кластеризації відео.

Результатом кваліфікаційної роботи є побудований алгоритм для кластеризації, а також його реалізація, та реалізація всіх моделей нейронних мереж, необхідних для його роботи.

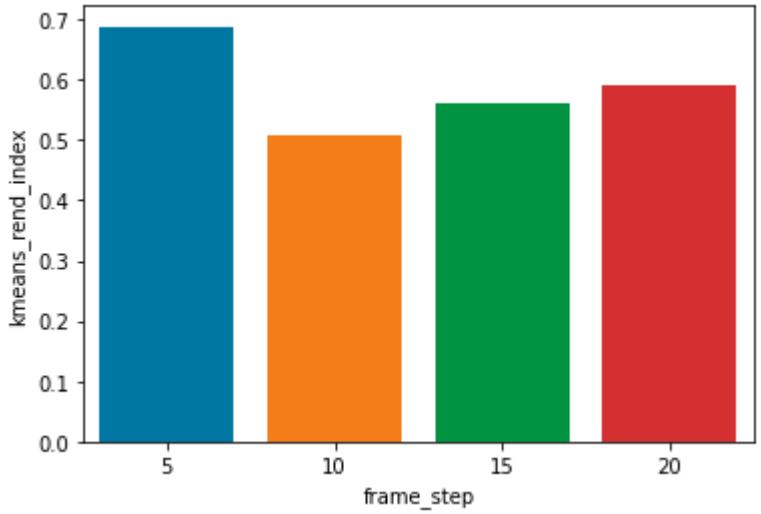
Мета по створенню моделі й алгоритму для кластеризації відео була виконана.

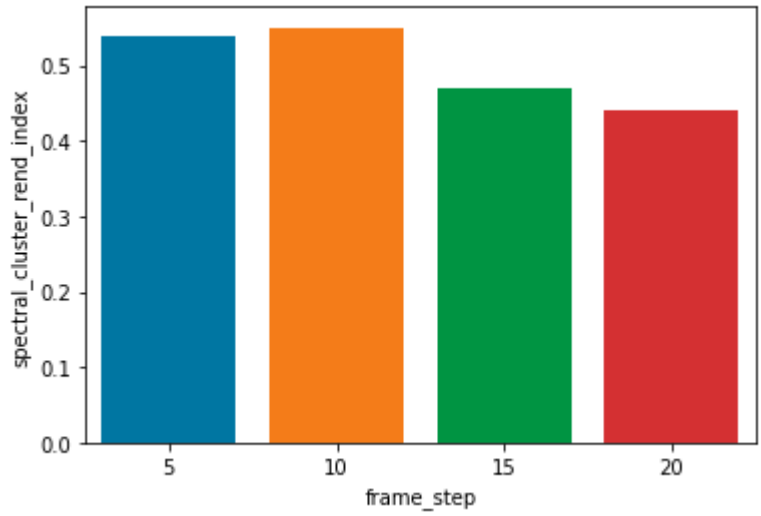
ДОДАТОК А

Приклади вирізаних кадрів із відео-датасету:

ДОДАТОК Б

Залежність точності моделі від кроку, з яким ми збираємо кадри із відео





ДОДАТОК В

Model: "vgg19 encoder"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 64)	1792
conv2d_1 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_2 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_3 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_4 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_5 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590080

max_pooling2d_2 (MaxPooling (None, 28, 28, 256) 0
2D)

conv2d_8 (Conv2D) (None, 28, 28, 512) 1180160

conv2d_9 (Conv2D) (None, 28, 28, 512) 2359808

conv2d_10 (Conv2D) (None, 28, 28, 512) 2359808

conv2d_11 (Conv2D) (None, 28, 28, 512) 2359808

max_pooling2d_3 (MaxPooling (None, 14, 14, 512) 0 2D)

conv2d_12 (Conv2D) (None, 14, 14, 512) 2359808

conv2d_13 (Conv2D) (None, 14, 14, 512) 2359808

conv2d_14 (Conv2D) (None, 14, 14, 512) 2359808

conv2d_15 (Conv2D) (None, 14, 14, 512) 2359808
global_average_pooling2d (G (None, 512) 0
(GlobalAveragePooling2D)

Джерела

[1] **https://docs.microsoft.com/en-us/azure/architecture/data-science-process/overview**

[2] **https://cybersecurityventures.com/the-world-will-store-200-zettabytes-of-data-by-2025/**

[3] J. Xie, R. Girshick, and A. Farhadi, “Unsupervised deep embedding for clustering analysis,” in ICML, 2016, pp. 478–487.

[4] F. Li and et al., “Discriminatively boosted image clustering with fully convolutional auto-encoders,” PR, vol. 83, pp. 161 – 173, 2018.

[5] B. Yang, X. Fu, N. D. Sidiropoulos, and M. Hong, “Towards k-meansfriendly spaces: Simultaneous deep learning and clustering,” in ICML, vol. 70, 2017, pp. 3861–3870.

[6] K. Tian and et al., “DeepCluster: A general clustering framework based on deep learning,” in ECML PKDD, 2017, pp. 809–825.

[7] Ji, X., Henriques, J.F., Vedaldi, A.: Invariant information clustering for unsupervised image classification and segmentation. In: ICCV (2019)

[8] SCAN: Learning to Classify Images without Labels Wouter Van Gansbeke, Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, Luc Van Gool

[9] A Simple Framework for Contrastive Learning of Visual Representations Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton

[10] Bishop Christopher M.. 2006. *Pattern Recognition and Machine Learning*. Springer.

[11] Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton "A Simple Framework for Contrastive Learning of Visual Representations"

[12] Karen Simonyan, Andrew Zisserman Very Deep Convolutional Networks for Large-Scale Image Recognition

[13] Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., and Bengio, Y. Learning deep representations by mutual information estimation and maximization. arXiv preprint arXiv:1808.06670, 2018