МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«___»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

Тема: «Система відеорозпізнавання фільмів за допомогою імовірнісних структур даних»

Виконавець: Богдан САМЧУК

Керівник: к.т.н. Олег ЗУДОВ

Нормоконтролер: к.т.н., доцент Олена ТОЛСТІКОВА

КИЇВ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*
Кафедра к*омп'ютерних інформаційних* технологій
Спеціальність*: 122 "Комп'ютерні науки"*
Освітньо-професійна програма: "*Інформаційні технології проектування*"

ЗАТВЕРДЖУЮ
завідувач кафедри КІТ
_____Аліна САВЧЕНКО
(підпис)

«___»_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Самчука Богдана Володимировича

(ПІБ випускника)

1. Тема роботи: «Система відеорозпізнавання фільмів за допомогою імовірнісних структур даних» затверджена наказом ректора № 1976/ст від 29.09.2023р.

2. Термін виконання роботи: з 02 жовтня 2023 року по 31 грудня 2023 року.

3. Вихідні дані до роботи: Концепт комплексної системи ідентифікації відеофільмів.

4. Зміст пояснювальної записки: 1. Історія і розвиток відеотехнологій. 2. Системи та технології відеоідентифікації. 3. Алгоритми, бази даних і аналіз хешів

5. Перелік обов'язкового ілюстративного матеріалу: 1. Раян Гослінг в ролі репліканта. 2. Перцептивний хеш зображення. 3. Густина хешів. 4. Діаграма дій при хешуванні відео. 5. Схема роботи програми при пошуку відео по фрагменту

6. Календарний план-графік

з/п	Завдання	Термін виконання	Підпис керівника
1	Аналіз предметної області та огляд аналогів. Написання 1 розділу, історія розвитку відео	02.10.2023-16.10.2023	
2	Вибір технологій та їх властивості. Написання 2 розділу, аналіз алгоритмів	17.10.2023-30.10.2023	
3	Написання 3 розділу, розробка прототипу системи	31.10.2023-14.11.2023	
4	Загальне редагування та друк пояснювальної записки	15.11.2023-20.11.2023	
5	Проходження нормоконтролю, перепліт пояснювальної записки.	16.11.2023-20.10.2023	
6	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	19.12.2023-22.12.2023	

7. Дата видачі завдання 02.10.2023 р.

Керівник кваліфікаційної роботи _____ Олег ЗУДОВ
(підпис керівника)

Завдання прийняв до виконання _____ Богдан САМЧУК
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему: «Система розпізнавання відео за допомогою ймовірнісних структур даних» містить: 74 сторінки, 25 рисунків, 22 інформаційних джерела, 2 додатки.

Об'єкт дослідження – це методи та засоби опрацювання відео для його подальшого розпізнавання за його фрагментом.

Предмет дослідження – процес ідентифікації відео за його коротким фрагментом.

Мета кваліфікаційної роботи – отримати систему відеорозпізнавання з ймовірнісним алгоритмом пошуку серед фільмів. Проаналізувати результати швидкодії та ефективності системи.

Методи дослідження – мова програмування Java, утиліта imagemagick

Результати кваліфікаційної роботи рекомендується використовувати для подальших досліджень системи відеорозпізнавання.

ВІДЕО, ВІДЕОРОЗПІЗНАВАННЯ, АЛГОРИТМИ, ЗОБРАЖЕННЯ, ХЕШ, СТРУКТУРИ ДАНИХ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

DVD (Digital Versatile Disc)	—	Цифровий багатоцільовий диск
VHS (Video Home System)	—	Інтегроване середовище розробки
HD (High Definition)	—	Висока роздільна здатність
OCR (Optical character recognition)	—	Технологія оптичного розпізнавання символів
A-Hash (Avarage Hash)	—	Хеш на основі середнього значення

ВСТУП

Сучасний світ насичений інформацією, яка надходить до нас у найрізноманітніших форматах. Однак однією з найбільш доступних та популярних форм сприйняття інформації залишається відео. Відеоролики та фільми, які ми переглядаємо, не лише розважають та освічують нас, але й створюють величезний обсяг цифрових даних. Все це ставить перед нами завдання ефективного аналізу та обробки відеоматеріалів.

В даній роботі буде розглянуто побудову системи розпізнавання фільмів за допомогою ймовірнісної структури даних. Розпізнавання об'єктів та подій у відео стає дедалі важливішим завданням у зв'язку з розвитком сфери мультимедіа та відкриттям нових можливостей для використання відеоданих у різних сферах життя. Ця робота присвячена розгляду та розробці методів аналізу відео, зокрема структуризацію кадрів у відео, створення автоматизованих систем індексації контенту, підвищення безпеки та багатьох інших застосувань.

Також потрібно дослідити методи та засоби опрацювання відео для подальшого розпізнавання за його фрагментом. Це включатиме розбір ймовірнісних структур даних, як вони працюють та приклади таких структур. Також потрібно розглянути методи обробки відео, що таке перцептивне хешування, приклади алгоритмів хешування. Це дасть змогу ефективно спроектувати систему розпізнавання відео.

Також буде побудований прототип системи задля перевірки концепції такої системи. Потрібно провести практичні експерименти та аналіз результатів, визначення обсягу даних, потрібних для зберігання щоб продемонструвати практичну застосовність розроблених методів.

Основною частиною роботи стане розробка алгоритмів, які дозволять ідентифікувати відеоконтент на основі внутрішнього аналізу зображень і звуків. Це включає в себе розробку спеціалізованих фільтрів для виділення ключових елементів у кадрах, створеннях патернів зображення, аналіз сцен у відео.

Важливим аспектом є забезпечення високої точності розпізнавання при мінімальних помилках, що вимагає глибокого аналізу великих обсягів відеоданих.

Розроблені методи мають допомогти вирішити проблему автоматичного виявлення та класифікації відеоматеріалів за різними критеріями, що є актуальним для медіа-індустрії, рекламного сектору та систем безпеки.

Крім того, робота передбачає детальне дослідження різних форматів відео та їхньої сумісності з розроблюваною системою. Це включає в себе оптимізацію алгоритмів для ефективної роботи з високоякісними відеоформатами та стиснутими версіями відео. Також передбачається розробка інтерфейсу для користувачів, який дозволить легко інтегрувати систему в існуючі платформи для керування відеоконтентом.

На заключному етапі проекту планується проведення серії випробувань та валідації системи на реальних даних. Це передбачає тестування системи в різних умовах, таких як зміна освітлення, різноманітність відеоматеріалів та аналіз її ефективності при різних рівнях якості відео. Мета цих випробувань - визначити межі точності розпізнавання та зробити необхідні корективи для поліпшення продуктивності системи.

Також передбачено аналіз системи, що дозволить оцінити потенціал комерційного застосування розробленої системи, виявити можливі області її вдосконалення та визначити додаткові можливості для її розширення.

РОЗДІЛ 1

ІСТОРІЯ І РОЗВИТОК ВІДЕОТЕХНОЛОГІЙ

1.1. Відео та його роль в сучасному світі

Відео - це змінна в часі послідовність візуальних кадрів, що супроводжується звуком. Його різноманітне застосування охоплює розважальні засоби, такі як кінематограф і музичні кліпи, освітні ресурси та звичайні розважальні матеріали. Здатність поєднувати візуальні та звукові елементи робить відео найпопулярнішим, універсальним і впливовим засобом у сучасному суспільстві. А його роль в комунікаціях людини зі світом сьогодні важко переоцінити.

Хронологічний розвиток відеотехнологій демонструє прогрес від зародження аналогових стадій до сучасної епохи цифрових форматів високої чіткості. Це був тривалий процес розвитку, який дав змогу мільйонам людей отримувати щось більше, ніж просто картинку.

Поява аналогового телебачення на початку 20-го століття символізує зародження відеотехнологій. Такі візіонери, як Джон Логі Берд, у 1920-х роках продемонстрували фундаментальні принципи сканування та передачі рухомих зображень в ефірі, заклавши основу для подальших досягнень.

Поява кольорового телебачення стала поворотним моментом, коли з'явилося кольорове телебачення. Протягом 1950-х і 1960-х років відбувся перехід від монохромного до кольорового мовлення, що відкрило нову візуальну парадигму. Ця епоха ознаменувалася стандартизацією кольорового телебачення та його повсюдним поширенням на глобальних платформах мовлення.

1.2. Поява технологій відеозапису

1970-ті роки стали переломними з появою технологій відеозапису, уособленням яких стало поширення відеокасетних відеомагнітофонів (відеомагнітофонів). Цей трансформаційний розвиток дав можливість споживачам записувати та відтворювати контент, започаткувавши зміну парадигми телевізійного споживання. Війна форматів між VHS і Betamax ще більше підкреслила технологічну конкуренцію на ринку споживчого відео.

Рис. 1.1. VHS та Betamax касети

Поява цифрового відео ознаменувала значний зсув. Цифрове відео кодує візуальну та аудіоінформацію в дискретні числові значення, що дозволяє більш точно відтворювати та маніпулювати контентом. Цей перехід призвів до покращення якості зображення, точності передачі кольору та можливості включення розширених функцій, таких як приховані субтитри та кілька звукових доріжок.

1.3. Перехід від VHS до DVD і далі

Одним із ключових моментів в еволюції відеотехнологій був перехід від VHS (Video Home System) до DVD (Digital Versatile Disc). VHS, з його форматом магнітної стрічки, широко використовувався для домашнього відеоспоживання наприкінці 20-го століття. Однак DVD-диски пропонували кращу якість відео та аудіо, компактний розмір і зручність довільного доступу до певних сцен. Спочатку DVD означало «цифровий відеодиск». Пізніше було запропоновано, що абревіатура розшифровується як «Digital Versatile Disc». Багато в чому тому, що DVD-диски можна використовувати набагато більше, ніж для зберігання відеовмісту.

Рис 1.2. Перший відеоплеєр CD-3000

Поява технології Blu-ray ще більше підвищила рівень домашніх розваг, окрім DVD. Диски Blu-ray збільшили ємність зберігання, що дозволило передавати контент високої чіткості з чудовою якістю звуку. Цей перехід вплинув не лише на домашні розваги, але й на способи виробництва та розповсюдження відеоконтенту.

1.4. Відеоформати високої чіткості (HD) і 4K

Попит на вищу роздільну здатність зображення призвів до розробки та широкого впровадження форматів відео високої чіткості (HD). HD-відео пропонує збільшену кількість пікселів, що призводить до чіткішого та детальнішого зображення. Дві основні роздільні здатності HD - 720p (1280x720 пікселів) і 1080p (1920x1080 пікселів).

Впровадження відео 4K, заснованого на HD, ознаменувало революційний прогрес у якості відео. 4K, також відома як Ultra High Definition (UHD), забезпечує в чотири рази більшу роздільну здатність, ніж 1080p, завдяки чому досягається приголомшлива чіткість і деталізація. Стандарт 4K набув поширення в цифровому кіно, потокових сервісах і високоякісних споживчих дисплеях.

З появою широкосмугового Інтернету та підвищенням його швидкостей, користувачі отримали можливість переглядати відео високої якості без необхідності завантажувати великі файли. Це призвело до зміни звичок споживання медіаконтенту: тепер користувачі мають доступ до стрімінгових сервісів, які пропонують фільми та серіали в HD та 4K якості.

Розвиток форматів HD та 4K також сприяв популяризації великих екранів та розумних телевізорів, що дозволяє глядачам насолоджуватися кінематографічним досвідом удома. Інтернет забезпечив легкий доступ до цих високоякісних відеоформатів, що, у свою чергу, вплинуло на стандарти виробництва відеоконтенту. Виробники контенту тепер змушені забезпечувати високу якість зображення, щоб задовольнити вимогливого споживача.

Інтернет також відіграє ключову роль у дистрибуції відео. З появою таких платформ як YouTube, Vimeo та інших стрімінгових сервісів, які підтримують HD і 4K формати, створення та розповсюдження відеоконтенту стало доступним не тільки для професійних студій, але й для аматорів. Це

призвело до великого різноманіття контенту, доступного для перегляду в Інтернеті.

Враховуючи швидкий розвиток технологій та зростаючу потребу в широкосмуговому Інтернеті, можна очікувати, що тенденції розвитку відеоформатів продовжуватимуться. Це включає подальше вдосконалення якості зображення, зменшення об'ємів файлів для спрощення передачі даних через Інтернет та розробку нових технологій компресії, які дозволять ефективніше використовувати пропускну здатність мережі.

1.5. Розвиток потокових сервісів та їхній вплив на споживання відеоконтенту

Еволюція відеотехнологій тісно пов'язана з розвитком потокових сервісів, які докорінно змінили спосіб споживання відеоконтенту аудиторією. Такі платформи, як Netflix, Hulu та Amazon Prime Video, зруйнували традиційні моделі дистрибуції, пропонуючи доступ на вимогу до величезної бібліотеки контенту.

Потокові сервіси використовують технології цифрового стиснення для передачі високоякісного відео через інтернет. Це призвело до переходу від фізичних носіїв (DVD, Blu-ray) і лінійного телевізійного мовлення до персоналізованого споживання контенту в інтернеті. Феномен "запійного перегляду", який став можливим завдяки потоковому мовленню, змінив глядацькі звички та очікування. Він передбачав практику невпинного перегляду певної телевізійної програми чи серіалу, яка часом навіть створювала проблеми зі здоров’ям людей. Люди були більш схильними до депресії, ожиріння, самотності тощо. Хоча суперечки щодо його шкідливості досі не вщухають.

З іншого боку, ера потокового мовлення каталізувала виробництво оригінального контенту на потокових платформах, сприяючи новій ері креативності та різноманітності в оповіданні історій. Цей зсув також

спонукав традиційних мовників і студії кінозапису адаптуватися до мінливого ландшафту, підкреслюючи важливість цифрової дистрибуції та онлайн-доступності.

Кількість відео в мережі росте з кожною секундою. Наприклад, в сервіс YouTube, кожну секунду публікується приблизно 6000 годин відеоматеріалів. Ростуть і кількість фільмів. Згідно з даними Motion Picture Association of America (MPAA), кожного року виходить близько 1000 нових фільмів. Часто, під проводження часу в соціальних мережах, ми натрапляємо на уривки кіно або трейлери, але назва фільму не вказана. В такому випадку зручно було б віднайти оригінал лише за його маленькою частинкою.

Для таких цілей, потрібно зробити з цілого фільму такий ідентифікаційний та універсальний об’єкт, який був би легко впізнаваний за своїми унікальними ознаками.

1.6. Поняття відеоідентифікації

Відеоідентифікація є ключовим елементом в сучасному світі цифрових технологій та медіа. Цей процес включає в себе використання різноманітних технік та алгоритмів для аналізу відео з метою ідентифікації конкретних осіб, об'єктів або патернів поведінки. Відеоідентифікація використовується в широкому спектрі застосувань, від охорони та нагляду до маркетингу та розважальної індустрії.

Одним з ключових елементів відеоідентифікації є машинне навчання та штучний інтелект. Ці технології дозволяють системам самостійно "навчатися" на основі великих масивів відеоданих, що підвищує точність ідентифікації та робить процес більш ефективним. Наприклад, алгоритми глибокого навчання можуть розпізнавати обличчя або інші специфічні характеристики відеозображення, що має велике значення у сферах безпеки та правоохоронних органів[1,6].

Крім того, відеоідентифікація відіграє важливу роль в аналізі поведінки споживачів та рекламній індустрії. Завдяки здатності точно ідентифікувати реакції людей на різні стимули, вона може допомагати в розробці більш ефективних маркетингових стратегій.

Розвиток технологій відеоідентифікації також відкриває двері для нових можливостей у розважальній індустрії, таких як інтерактивні медіа та відеоігри, де розпізнавання жестів та емоцій грає ключову роль

Хоча більш ефективно для розпізнавання відео було б використовувати машинне навчання, в цій роботі я обмежусь звичайними алгоритмами та спеціальними структурами даних.

1.7. Історичний огляд систем відеоідентифікації

Системи відеоідентифікацій мають тернистий шлях розвитку технологій, що відображає загальний поступ у сфері обробки відеоданих та комп'ютерного зору. Починаючи з ранніх спроб в автоматизації відеоспостереження, історія відеоідентифікації характеризується швидким розвитком та інтеграцією новітніх технологій.

Перші системи відеоідентифікації з'явились в середині 20-го століття, коли вперше були використані прості методи автоматичного розпізнавання образів. Ці системи були досить обмежені у своїх можливостях, зазвичай використовуючи прості алгоритми для визначення руху або змін у відео. На той час, це було значним досягненням, враховуючи обмежені обчислювальні ресурси.

Протягом 1980-х та 1990-х років, з розвитком комп'ютерних технологій, відеоідентифікація зробила великий крок вперед. Покращення у області цифрової обробки зображень та введення алгоритмів машинного навчання дозволили створювати більш складні системи. Це був час, коли почали розвиватись технології розпізнавання облич, які сьогодні широко використовуються в різноманітних сферах, від безпеки до смартфонів.

На початку 21-го століття, з появою штучного інтелекту та глибокого навчання, системи відеоідентифікації зазнали ще одного значного стрибка у своєму розвитку. Технології глибокого навчання, зокрема, нейронні мережі, дозволили обробляти величезні обсяги відеоданих з високою точністю, відкриваючи нові можливості для розпізнавання складних патернів та поведінкових моделей.

Сучасні системи відеоідентифікації неймовірно різноманітні та потужні. Вони здатні аналізувати не тільки візуальні аспекти, але й інтегрувати інформацію з інших датчиків, що відкриває шлях до створення комплексних систем спостереження та аналітики.

Очікується, що майбутнє розвитку систем відеоідентифікації буде зосереджено на подальшому підвищенні точності, інтеграції з іншими видами інтелектуальних систем.

1.8. Система відеоідентифікації фільмів

Головна мета моєї роботи – розробка системи відеоідентифікації фільмів, на основі короткого фрагмента візуального контенту. Користувачі завантажують короткий фрагмент фільму, а система повинна точно розпізнати і надати інформацію про фільм, наприклад, його назву, рік випуску, режисера і, можливо, інші релевантні деталі.

Для появи такого сервісу, уже створені всі передумови. По перше це бажання людей миттєво отримати інформацію. В епоху з прискореним темпом життя і ненаситним попитом на миттєву інформацію, система ідентифікації фільмів задовольняє фундаментальну потребу сучасного світу. Цей аспект системи безпосередньо пов'язаний зі зміною динаміки того, як люди споживають та взаємодіють з медіа, особливо з індустрією кінофільму. Користувачі часто опиняються в ситуаціях, коли швидкоплинний момент або коротке знайомство з фрагментом фільму викликає цікавість і бажання

якнайшвидше ідентифікувати відповідний фільм. І моя система може задовольните це бажання людей.

Дозволяючи користувачам завантажувати короткі вирізки з фільму, система має використовувати передові алгоритми і величезні бази даних для швидкої ідентифікації відповідного фільму. Цей швидкий та ефективний процес замінює громіздкі альтернативи ручного пошуку, нечітких онлайн-запитів або розчарування від нездійсненних спогадів. У світі, де час є дорогоцінним товаром, система стає цінним інструментом для тих, хто шукає негайні відповіді на свої запити, пов'язані з фільмами.

Розглянемо сценарії, коли люди бачать захопливі сцени з якогось фільму в соціальних мережах. Програма перетворює ці моменти інтриги на можливості швидко знайти потрібне кіно, не чекаючи відповіді в коментарях від інших користувачів, чи довго шукати джерело сцени по акторському складу. Натомість вони можуть швидко ввести в систему фрагмент фільму, який мають під рукою, і отримати у відповідь багато інформації про нього - від назви та року випуску до знімальної групи та схожих фільмів.

Ця зручність виходить за межі простого пошуку інформації; вона формує спосіб взаємодії людей з фільмами на соціальному та культурному рівнях. Можливість швидко знаходити фрагменти фільмів і ділитися ними сприяє спонтанним розмовам, дозволяючи користувачам обмінюватися враженнями від спільного перегляду.

Не останню роль відіграє і сам попит ринку. У міру того, як технології розвиваються безпрецедентними темпами, користувачі занурюються в цифрову екосистему, де доступ до інформації є не лише бажаним, але й очікуваним у будь-яку мить. Це підвищує потребу в інструментах, які спрощують і покращують розважальний досвід. Система ідентифікації фільмів, буде відповіддю на ці запити зі сфери технологічних послуг.

Система ідентифікації фільмів не лише задовольняє попит, але й відкриває цілу низку можливостей монетизації для розробників та зацікавлених сторін в індустрії розваг.

Шляхи монетизації проявляються через різні канали, кожен з яких робить свій внесок у фінансову життєздатність системи. Партнерство з ключовими компаніями стрімінгових сервісів може призвести до співпраці над ексклюзивним контентом, рекламними зв'язками.

Реклама, ще один спосіб отримання прибутку, капіталізує залучення користувачів системи. Оскільки користувачі активно шукають ідентифікований контент і діляться ним, платформа стає найкращим місцем для таргетованої реклами. Рекламодавці можуть використовувати базу користувачів системи для просування відповідного контенту, послуг або продуктів, створюючи симбіоз, де користувацький досвід і реклама безперешкодно співіснують.

Також, з ціллю збільшити потенційні прибутки, можна запровадити преміум-функції. Користувачі, які бажають покращити свій досвід або отримати доступ до розширених функцій, чи отримувати рекомендації на основі історії пошуку, можуть обрати преміум-пропозиції, безпосередньо сприяючи збільшенню доходу системи.

1.9. Аналоги систем розпізнавання контенту

1.9.1 Shazam

Shazam - неймовірно популярний додаток для ідентифікації музики, який дозволяє користувачам ідентифікувати пісні всього за кілька секунд. Він став миттєвим хітом після свого випуску у 2002 році і продовжує залишатися вкрай необхідним і важливим інструментом у світі музичних технологій. З моменту свого запуску Shazam розширив спектр своїх послуг, вийшовши за рамки ідентифікації музики. Він також взяв на себе роль просування музичних відкриттів, співпрацюючи з такими платформами, як Instagram та Snapchat.

Рис. 1.3. Логотип Шазаму

Попередня обробка аудіо в Shazam - це не тільки перший, але й найбільш важливий крок у процесі ідентифікації пісні. На етапі попередньої обробки, для створення відбитку пісні Shazam використовує наступний алгоритм дій:

Спочатку потрібно застосувати перетворення Фур'є - це математичний інструмент, який розкладає сигнал на його складові частоти. Названа на честь французького математика Жозефа Фур'є, ця техніка є особливо цінною в обробці та аналізі сигналів. У контексті аудіосигналів перетворення Фур'є застосовується для перетворення часового представлення, яке показує, як амплітуда сигналу змінюється з часом, у частотне представлення.

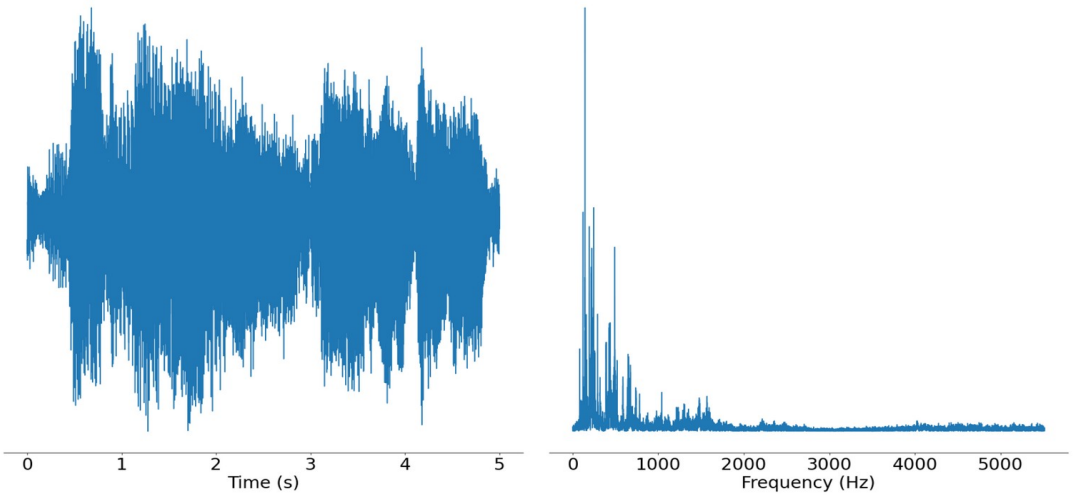

Рис. 1.4. Зліва — пісня, справа спектр її частот

Існує два основних варіанти перетворення Фур'є: безперервне перетворення Фур'є і дискретне перетворення Фур'є. Безперервне підходить для неперервних сигналів, але оскільки цифрові системи обробляють дискретні дані, в практичному застосуванні часто використовується Дискретне. Швидке перетворення Фур’є (ШПФ) - це алгоритмічна реалізація Дискретного перетворення Фур’є, яка значно прискорює процес обчислень. ШПФ бере кінцеву послідовність рівновіддалених відліків сигналу і обчислює його частотні компоненти. Використовуючи ШПФ, Shazam ефективно аналізує фрагменти аудіо в реальному часі, визначаючи присутні частоти і створюючи частотний спектр, який є основою для подальшого аналізу.[19]

Результатом перетворення Фур'є є частотний спектр, який відображає інформацію про амплітуду і фазу для кожної частотної складової, присутньої в сигналі. У контексті аудіосигналів цей спектр дає уявлення про різні висоти і тони, з яких складається звук. У музиці кожен інструмент і вокальний елемент вносить певні частоти, а перетворення Фур'є допомагає виокремити і зрозуміти ці внески. Перетворення з часової області в частотну є фундаментальним кроком в обробці сигналів і дозволяє більш тонко аналізувати складні сигнали, такі як музика.

2. Спектрограма: Спектрограма - це двовимірне представлення аудіосигналу, яке показує, як частотний склад сигналу змінюється з часом. Вона візуалізує розподіл частот в аудіосигналі і дозволяє ідентифікувати різні характеристики звуку. У контексті аудіосигналів спектрограма забезпечує детальне та інтуїтивно зрозуміле відображення різної інтенсивності різних частот, присутніх у сигналі. Вісь x зазвичай представляє час, вісь y - частоту, а колір або інтенсивність в кожній точці вказує на амплітуду або енергію відповідної частоти в певний момент часу.

Спектрограми особливо корисні для аналізу сигналів зі змінним у часі частотним вмістом, таких як музика. Вони генеруються шляхом поділу сигналу на короткі сегменти, що перекриваються, застосування перетворення

Фур'є до кожного сегмента для отримання частотного вмісту, а потім побудови графіка результатів у часі. Такий частотно-часовий аналіз дозволяє виявити закономірності, перехідні явища та інші динамічні особливості, які можуть бути не помітними в простому частотному спектрі. У випадку з Shazam спектрограми допомагають виявити унікальні характеристики пісень, візуалізуючи, як змінюються частотні компоненти протягом звукового фрагмента.

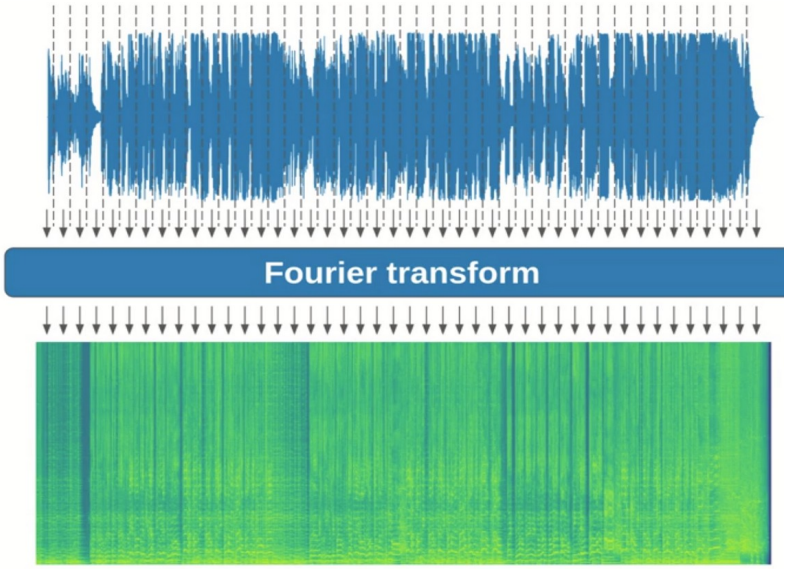

Рис. 1.5. Приклад створення спектрограми

3. Визначення піків: Спектрограма графічно відображає частоти звуку на вертикальній осі та час на горизонтальній осі. Цей графік дає детальне уявлення про те, як частоти пісні змінюються з часом, фіксуючи нюанси та варіації звуку.

Під час аналізу зміни частоти з часом, конкретні точки чи визначні пункти в спектрограмі визначаються як піки на графіку Ці точки часто відповідають значущим подіям у звуці, таким як зміни акордів, інструментальні соло чи вокальні фрази. Фокусуючись на цих унікальних особливостях, Shazam створює відбиток, (рис. 2.4) який охоплює суттєві елементи пісні без використання повного аудіохвилі.

Процес відбитку Shazam розроблено так, щоб бути надійним та ефективним, дозволяючи швидко та точно визначати пісні навіть за коротких аудіо-відрізків. Аналіз частоти проти часу є важливим етапом цього процесу, що дозволяє Shazam фокусуватися на найважливіших аспектах аудіо-сигналу для ефективної ідентифікації та порівняння.

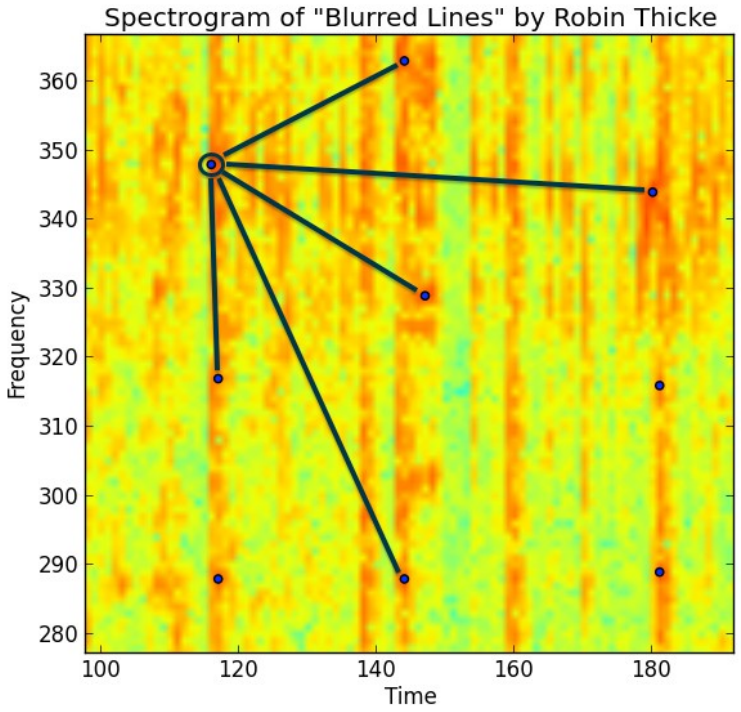

Рис. 1.6. Ключові точки спектрограми

Отже, отримуючи якусь невідому пісню на вхід, Shazam робить всі ті ж самі операції, визначаючи ключові моменти цього відрізку пісні. Додаток порівнює цей акустичний відбиток з акустичними відбитками, що зберігаються в його величезній базі даних, і шукає найближчий збіг. Цей процес може бути виконаний за лічені секунди, надаючи своїм користувачам миттєве розпізнавання пісень.

Додаток не лише аналізує спектрограму, але й враховує інші фактори, такі як темп, ритм і тональність. Такий багатовимірний аналіз дозволяє Shazam точно ідентифікувати пісні, навіть якщо вони звучать у різних версіях або кавер-версіях.

Крім того, база даних Shazam регулярно оновлюється, додаючи нові пісні від молодих виконавців. Це постійне розширення гарантує, що додаток залишається актуальним і може точно визначати останні музичні тенденції. Крім того, алгоритм Shazam постійно допрацьовується та вдосконалюється на основі відгуків користувачів та аналізу даних, що ще більше підвищує його точність та ефективність.

Вплив Shazam на музичну індустрію неможливо переоцінити. Окрім неймовірної здатності ідентифікувати пісні, додаток легко інтегрується з різними платформами, такими як Snapchat, WhatsApp та Apple Music. Завдяки цим партнерствам Shazam став чимось більшим, ніж просто інструментом для пошуку музики. Він перетворився на потужного гравця у сфері просування та відкриття музики.

1.9.2 Google Lens

Хоча Google Lens не призначений спеціально для фільмів, він дозволяє користувачам шукати та ідентифікувати об'єкти на зображеннях. Подібна концепція може бути застосована для ідентифікації сцен або постерів з фільмів.

Google Lens - це технологія розпізнавання зображень, розроблена компанією Google, яка дозволяє користувачам шукати та взаємодіяти з реальним світом за допомогою камери свого смартфона.

Принцип роботи приблизно наступний: зображення, які користувач передає у систему, обробляються за допомогою високорівневих алгоритмів комп'ютерного зору, які аналізують візуальну інформацію на зображеннях.

Як тільки користувач захоплює зображення за допомогою камери смартфона, Google Lens ініціює багатоетапний процес для розуміння та інтерпретації вмісту на зображенні. Цей процес є ключовим для функціонування Google Lens, оскільки він прагне витягнути значущу інформацію з візуальних даних.

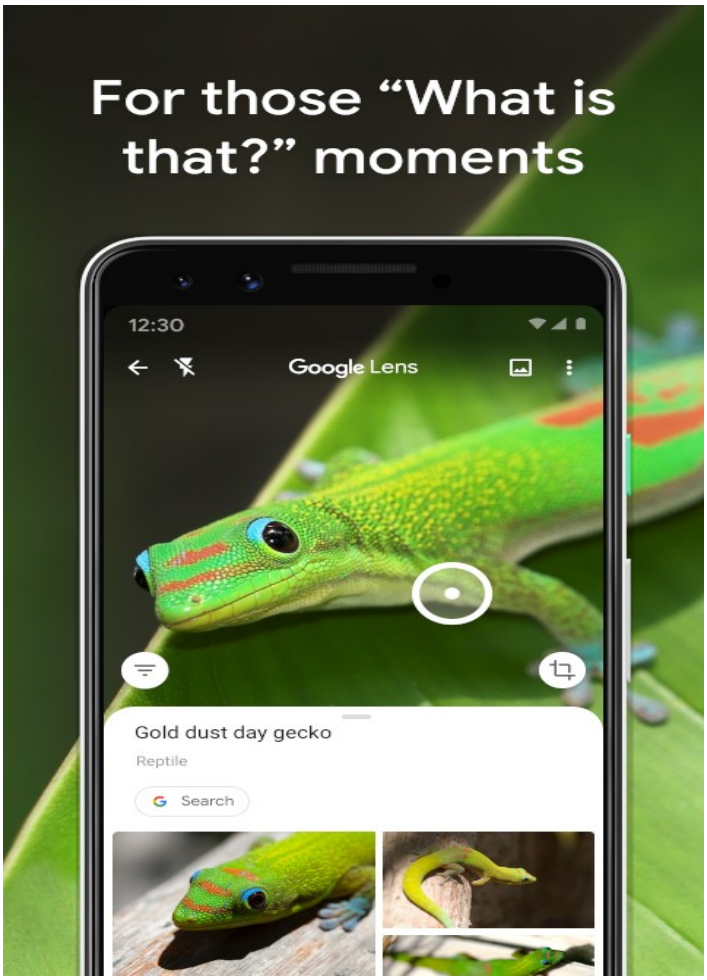

Рис. 1.7. Фото з презентації програми на Google Play

Google Lens використовує високорівневі техніки комп'ютерного зору, що включають глибоке та машинне навчання. Ці моделі тренуються на величезних наборах даних, що дозволяє їм розпізнавати широкий спектр об'єктів, текстів та візуальних паттернів на зображеннях. Глибоке навчання, яке є підмножиною машинного навчання, включає використання штучних нейронних мереж із кількома шарами для автоматичного вивчення та вилучення характеристик з даних. Ця можливість дозволяє Google Lens не лише розпізнавати об'єкти на зображеннях, але і розуміти та інтерпретувати контекст, в якому ці об'єкти з'являються. Застосування машинного та глибокого навчання в обробці зображень дозволяє Google Lens постійно покращувати свою точність і розширювати здатність розпізнавати

різноманітні візуальні елементи, роблячи його потужним інструментом для аналізу та взаємодії з об'єктами реального світу.

Google Lens використовує глибоке навчання та моделі машинного навчання для розпізнавання об'єктів, тексту та візуальних шаблонів на зображеннях.

Технологія розпізнавання об'єктів ґрунтується на обширній базі даних зображень та інформації, які використовуються для відповідності візуальних особливостей на зафіксованих зображеннях з відомими об'єктами. Google Lens використовує потужність передових алгоритмів комп'ютерного зору для надійного розпізнавання об'єктів і точного вилучення тексту із зображень. В основі цього складного процесу лежать згорткові нейронні мережі (Convolutional Neural Networks) - клас моделей глибокого навчання, спеціально розроблених для задач, пов'язаних із зображеннями. ЗНМ чудово розпізнають ієрархічні та просторові особливості зображень, що робить їх добре пристосованими до складного завдання ідентифікації об'єктів у різноманітних візуальних контекстах. Завдяки процесу, відомому як згортка, ці мережі систематично аналізують шаблони та ієрархії зображень, що дозволяє Google Lens розпізнавати об'єкти з високим ступенем точності.

Розпізнавання об'єктів, ключовий аспект функціональності Google Lens, передбачає навчання ШНМ на великих наборах даних, що містять марковані зображення. Під час цього етапу навчання нейромережа вчиться розпізнавати відмінні риси різних об'єктів, такі як форми, текстури та кольорові патерни. Коли модель навчена, вона може узагальнити ці знання для нових, небачених зображень, що дозволяє Google Lens ідентифікувати широкий спектр об'єктів, знятих камерою смартфона користувача. Таке використання CNNs дає змогу Google Lens забезпечувати надійне розпізнавання об'єктів у реальному часі, збагачуючи користувацький досвід контекстною інформацією про навколишній світ.

Також, Google Lens використовує оптичне розпізнавання символів (Optical Character Recognition) для вилучення тексту із зображень. OCR - це

технологія, яка перетворює текст всередині зображень на сприйнятний для комп’ютера. Це дозволяє Google Lens не тільки розпізнавати і розуміти візуальні елементи зображення, але й витягувати значущу текстову інформацію. Інтегруючи CNN для розпізнавання об'єктів і OCR для вилучення тексту, Google Lens створює комплексний підхід до візуального розуміння, дозволяючи користувачам безперешкодно взаємодіяти з текстовими і візуальними аспектами свого оточення.

Розпізнавання тексту є ключовою функцією Google Lens, що дозволяє технології визначати та вилучати текст із зображень. Ця можливість дозволяє користувачам взаємодіяти з текстовим вмістом, зафіксованим за допомогою камер своїх смартфонів, легко та ефективно. Незалежно від того, чи це витяг фрагменту документа, знака чи будь-якої письмової інформації в полі зору, Google Lens використовує передову технологію оптичного розпізнавання символів (OCR), щоб перетворити візуальне представлення тексту в читабельні для комп’ютера дані.

Рис 1.8. Принцип попередньої обробки фото для ідентифікації тексту

Також, у нього є можливість розпізнавання тексту у багатьох мовах. Він може читати та розуміти текст на різних мовах, що сприяє роботі з різноманітною аудиторією користувачів. Ця підтримка багатьох мов означає, що користувачі можуть використовувати Google Lens для перекладу тексту з

однієї мови на іншу, копіювання тексту для подальшого використання або пошуку додаткової інформації, пов'язаної з визначеним текстом. Технологія використовує моделі машинного навчання для розуміння контексту та структури розпізнаного тексту, роблячи її універсальним інструментом для завдань, пов'язаних з мовою.

Google Lens надає користувачам інтерактивні опції на основі розпізнаних об'єктів чи тексту. Наприклад, якщо вона визначає пам'ятник, вона може пропонувати деталі про місце, історичну інформацію або відповідні веб-сайти.

Користувачі можуть виконувати веб-пошуки, знаходити товари в Інтернеті, дізнаватися інформацію про фільми, книги чи твори мистецтва, просто вказавши свою камеру на відповідний об'єкт.

У підсумку, Google Lens поєднує технології розпізнавання об’єктів, машинного навчання для того, щоб дозволити користувачам досліджувати та взаємодіяти з навколишнім світом за допомогою камери свого смартфона, суттєво прискоривши отримання інформації з навколишнього світу.

ВИСНОВОК ДО РОЗДІЛУ 1

У цьому розділі було проведено всебічний аналіз еволюції відеотехнологій та їх впливу на сучасний світ. Відео зіграло важливу роль у культурі та комунікації, змінюючи спосіб, яким ми сприймаємо інформацію та розваги. Поява технологій відеозапису відкрила нові горизонти для творчості та зберігання інформації, відображаючи суттєві зміни в нашому способі взаємодії з медіа.

Перехід від аналогових форматів, таких як VHS, до цифрових форматів, зокрема DVD, підкреслив технологічний прогрес та підвищення доступності відеоконтенту. Розвиток інтернету призвів до еволюції відеоформатів до більшої чіткості, включаючи HD та 4K, знаменував нову еру у відеопродукції та перегляді, забезпечуючи неймовірну якість зображення та занурення в контент. Паралельно з цим, розвиток потокових сервісів кардинально змінив пейзаж медіаіндустрії, роблячи відеоконтент широко доступним та впливаючи на звички споживачів.

Важливою складовою цього розділу є огляд поняття відеоідентифікації, яке відіграє ключову роль у безпеці, медіа та інших галузях. Історичний огляд систем відеоідентифікації відкриває розуміння того, як технологічний прогрес сприяв розвитку більш складних і точних систем, які сьогодні використовуються у всьому світі.

РОЗДІЛ 2

СИСТЕМИ ТА ТЕХНОЛОГІЇ ВІДЕОІДЕНТИФІКАЦІЇ

2.1. Відеовідбитки пальців (фінгерпринтінг)

Дактилоскопія(визначення відбитків пальців) відео - важлива і складна технологія в галузі ідентифікації мультимедійного контенту. Оскільки цифровий ландшафт переживає різкий сплеск відеоконтенту, потреба в ефективних методах унікальної ідентифікації та управління цими ресурсами стає все більш першочерговою. Основний фокус в моїй системі відеоідентифікації полягає саме в точному та швидкому створенню цих відбитків для кожного відео.

Це поняття означає створення унікальних "відбитків пальців" для відеофайлів, що дозволяє їхню ідентифікацію та відстеження. Відеофінгерпринтинг, або ж хешування відео, має важливе застосування у сфері авторських прав, виявленні подвійних копій відео та боротьбі з фальсифікацією відеоматеріалів.

Варто зазначити, що при звичайне хешування даних не підходить для таких цілей, адже воно дуже чутливе до зміни вмісту. А перцептивне (сприйнятне) хешування, допускає зміни в початковій формі, без зміни кінцевого результату. Завдяки швидкому розвитку технологій, особливо за допомогою нейронних мереж, відеофінгерпринтінг стає більш потужним та ефективним. У наш час телебачення/кінематограф, соціальні мережі, цифровий маркетинг і відеоспостереження поступово й кумулятивно перетворили відеоконтент на пристрасний тип даних для обміну, зберігання та обробки. Згідно з даними Statista за 2022 рік, телевізійний інтернет-трафік зріс утричі між 2016 і 2021 роками, досягнувши 42 000 петабайт даних на місяць.

Такий величезний обсяг інформації в поєднанні з безліччю побутових та професійних застосувань повинен підтримуватися сильними науковими та методологічними парадигмами обробки відео, і дактилоскопія відео є однією з них. Дактилоскопія відео ідентифікує дубльовані, відтворені та/або дещо модифіковані версії певної відеопослідовності (запиту) у еталонному наборі відеоданих. Його також називають виявленням майже дублікатів або виявленням копій на основі вмісту. Термін хешування відео також використовується для програм відеофінгерпринтінгу, яке застосовуються до пошуку у дуже великій відеобазі даних.

Принцип можна проілюструвати на прикладі відбитків пальців людини. Візерунки дермальних виступів на кінчиках пальців людини є природними ідентифікаторами для людей. Хоча вони крихітні порівняно з усім тілом людини, людські відбитки пальців можуть унікально ідентифікувати людину незалежно від її фізіономічних змін і потенційних маскувань. . Аналогічно, хеш відео може бути однозначним ідентифікатором відео, навіть якщо його вміст піддається попередньо визначеному набору перетворень, що залежить від програми.

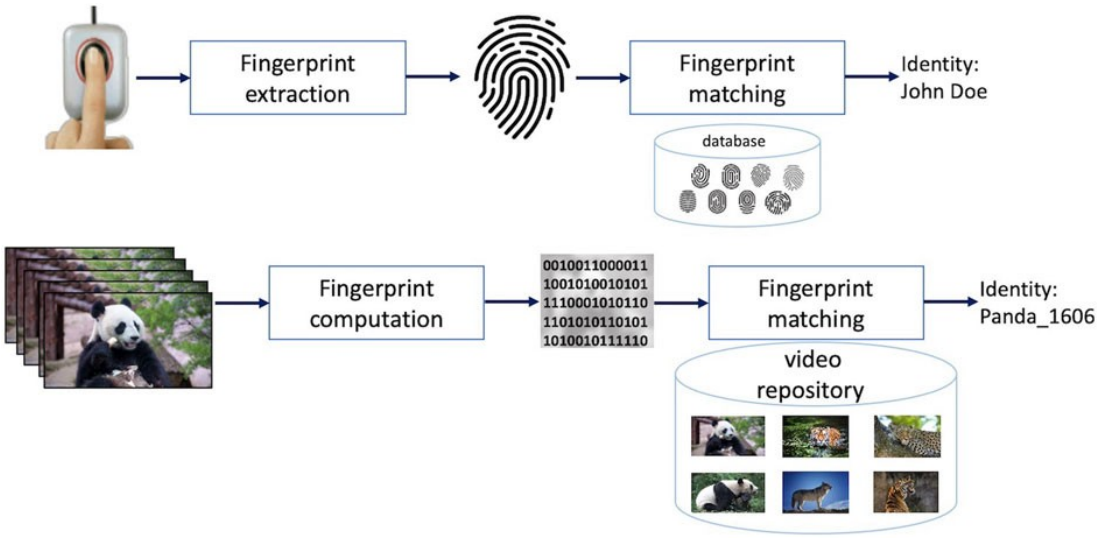

Рис. 2.1 Аналогія відбитків пальців людини та відео

Індексування відео можна розглядати як першу структуру для пошуку відео на основі вмісту. Метою індексування відео є пошук усіх відеопослідовностей, які візуально пов’язані із запитом. Наприклад, припустимо, що є відео, яке показує деяких ведмедів панд, а репозиторій відео складається з кількох послідовностей диких тварин. Рішення для індексування відео шукає всі послідовності у цьому сховищі, які містять ведмедів панд, а також зображення, що мають однаковий тип фону, як показано на рис. 2.8.

З цією метою основна інформація (іменована дескриптором) витягується із запиту та порівнюється з дескрипторами всіх послідовностей у цьому сховищі (які були попередньо обчислені та збережені). Таке порівняння неявно передбачає, що визначено міру подібності для візуальної близькості між двома відеопослідовностями та встановлено поріг, згідно з яким два дескриптори можуть бути зіставлені.

2.2. Варіанти використання відеофінгерпринтингу

Один із варіантів використання відеофінгерпринтингу це створення так званих водяних знаків. Цифровий водяний знак стосується ідентифікації будь-якої модифікованої версії відеоконтенту, як показано на рисунку 2. Наприклад, якщо припустити, що на екрані відображається відеоряд, який представляє деяких ведмедів панди, і вміст екрана записаний зовнішньою камерою, оригінальний вміст повинен бути ідентифікованим із версії, записаної на відеокамеру.

З цією метою, відповідно до системи цифрових водяних знаків, непомітно вставляється (або, як синонім, вбудовується) додаткова інформація (називається знаком або водяним знаком) у відеоконтент до його випуску (розповсюдження, зберігання, показ і т. д.). Виявляючи водяний знак у потенційно модифікованій версії відеовмісту з водяним знаком, оригінальний вміст має бути однозначно ідентифікованим.

Звичайно, водяний знак не можна відновлювати з будь-якого непозначеного вмісту (незалежно від того, чи він візуально пов'язаний з оригінальним вмістом, чи ні). [11]

Відбитки відео також пов'язані з ідентифікацією дещо зміненого (відтвореного або майже дубльованого) вмісту, але їх підхід відрізняється як щодо індексування, так і водяних знаків, як показано на малюнку 4. Повертаючись до попередніх двох прикладів, відбитки відео також повинні відстежувати майже дубльовану відеопослідовність (наприклад, записану на екрані послідовність панди) до її оригіналу (наприклад, оригінальну послідовність панди), яка зберігається у відеорепозиторії.

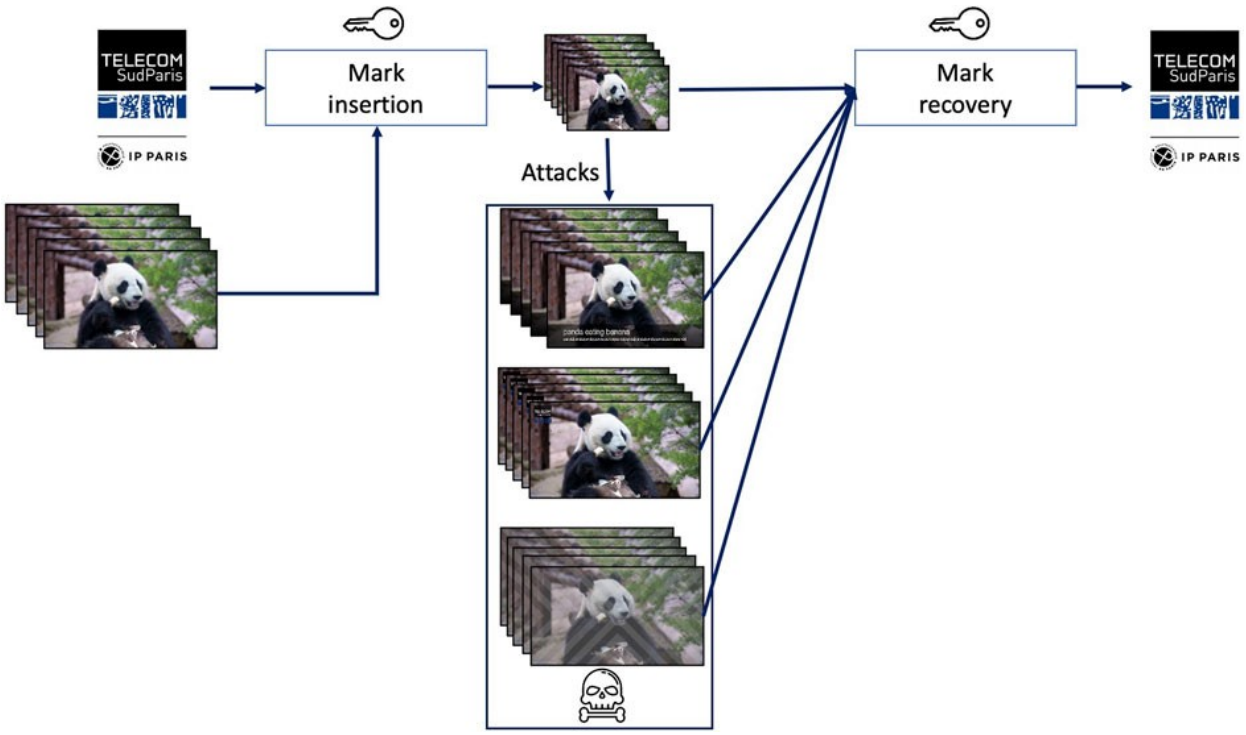

Рис. 2.2. Принцип водяного знака відео

Однак, на відміну від індексування, будь-яка інша послідовність, навіть візуально пов'язана з нею (наприклад, той самий ведмідь панда в інший час дня та/або в різних позах), не повинна визнаватися як ідентична. З цією метою, деяка важлива інформація (названа відбитком пальця або перцептивним хешем) витягується з відеопослідовності запиту (зауважте, що

ця інформація не вставляється раніше у вміст, як у випадку з послідовностями водяних знаків). Шляхом порівняння (відповідно до міри подібності та попередньо встановленого порогу) відбитка запиту з відбитками еталонної послідовності приймається рішення щодо візуальної ідентичності між відеопослідовностями.

2.3. Характеристики створення водяних знаків

Зазвичай, для створення водяних знаків розглядають три основні характеристики. Перша характеристика - це унікальність, що передбачає, що різний вміст призводить до різних водяних знаків за рівнем схожості та пов'язаним порогом. Друга характеристика - стійкість, яка стосується ідентифікації схожих послідовностей, навіть якщо вони близькі до дубльованих. Трансформації, такі як модифікації чи атаки, можуть змінити відео, але важливо зберігати його здатність до ідентифікації.

Крім того, важливо відзначити, що терміни, такі як копія, репліка або атаковане відео, хоч і концептуально схожі, можуть мати тонкі розрізнення в залежності від конкретного контексту використання відео.

Метод водяних знаків вважається ефективним для пошуку в наборі даних, якщо обчислення водяних знаків і визначення їх відповідності забезпечують низький час обчислення. Ефективність оцінюється середнім часом обчислення для ідентифікації запиту в контексті використання водяних знаків відео на конкретному обчислювальному середовищі та з використанням певного сховища даних.

Порівнюючи індексування та створення водяних знаків, можна визначити, що обидва підходи використовують концепцію відстеження контенту, проте вони відрізняються властивістю унікальності. Водяні знаки дозволяють відстежувати як вихідний контент, так і його репліки, але вимагають вставки додаткової інформації, тоді як індексування відстежує цілу семантичну групу, пов'язану з контентом, без вставки додаткових даних.

Крім того, важливо відзначити, що водяні знаки відео іноді також називають перцептивними відеохешами. Проте варто робити розрізнення щодо стійких відеохешів, які відносяться до областей застосування в сфері безпеки та судової експертизи і, як правило, застосовуються у випадках, де потрібно визначати різницю між атаками на збереження та маніпуляцією контентом.

Ці характеристики роблять створення водяних знаків парадигмою з потенційним впливом на велику кількість застосувань. Здатність ідентифікувати та відновлювати відео навіть під час спотворень є потужним інструментом для автоматичної фільтрації та витягання відео, запобігання порушенням авторських прав, моніторингу медіа-контенту на багатоканальних платформах, контекстуальної реклами чи бізнес-аналітики.

У кінцевому підсумку, продуктивність системи водяних знаків відео може бути об'єктивно оцінена шляхом оцінки її властивостей (унікальності, стійкості та ефективності пошуку в наборі даних) на великому наборі даних.

Застосування індексування та створення водяних знаків у сфері відеоаналітики може виявитися ключовим аспектом для вирішення завдань ідентифікації та валідації відеоконтенту. Йдучи далі, потрібно детальніше розглянути, як ці технології допоможуть при створенні потужних інструментів у сфері ідентифікації відеоконтенту.

Перцептивне хешування виступає як важливий компонент, особливо в аспекті відстеження зображень, де воно стає ефективним засобом перевірки автентичності та ідентифікації мультимедійних даних. Розглянемо, як ця техніка може співпрацювати з цифровим водяним знаком для забезпечення комплексного підходу до забезпечення безпеки та контролю за використанням відеоінформації.

2.4. Перцептивне хешування

Перцептивне хешування виступає як важливий компонент, особливо в аспекті відстеження зображень, де воно стає ефективним засобом перевірки автентичності та ідентифікації мультимедійних даних. Розглянемо, як ця техніка може співпрацювати з цифровим водяним знаком для забезпечення комплексного підходу до забезпечення безпеки та контролю за використанням відеоінформації.

Перцептивне хешування — це використання алгоритму відбитків пальців , який створює фрагмент, хеш або відбиток пальців різних форм мультимедіа. Перцептивний хеш — це тип хешу, чутливого до місцевості , який є аналогічним, якщо вхідні дані подібні.

Перцептивне хешування зображень, яке є однією з можливих технік, що можуть використовуватися разом із цифровим водяним знаком, стає однією з найбільш поширених областей досліджень і виявляється ефективним засобом перевірки автентичності та ідентифікації мультимедійних даних (зображень і відео).

Функції перцептивного хешування зображень базуються на витягненні певних надійних або інваріантних ознак з зображення для створення хешу (або відбитка), з властивістю, що дві абсолютно різні картини надають некорельовані хеші, в той час як два візуально схожі зображення (тобто сприйняті око людини як подібні) генерують висококорельовані хеші. У цьому випадку ефективна техніка перцептивного хешування повинна бути в змозі виявити, що зображення походить від іншого способом, який забезпечує його збереження перцептивно схожим, навіть якщо відповідні файли значно відрізняються.

Загалом, система перцептивного хешування для аутентифікації зображень складається з трьох основних етапів: етапу попередньої обробки, етапу генерації хешів та етапу прийняття рішення. Основна мета етапу попередньої обробки - покращити надійність ознак шляхом зменшення

впливу можливих спотворень за допомогою операцій обробки зображень, таких як зміна розміру, фільтрація, зменшення розмірності кольорового простору і т.д.

На наступному етапі генеруються та зберігаються посилання на хеші в наборі даних. У контексті аутентифікації зображень той же процес перцептивного хешування застосовується до зображення, яке потрібно аутентифікувати, для генерації тестового хешу. [4]

Після цього, на етапі прийняття рішення тестовий хеш порівнюється з посиланнями на хеш, щоб перевірити автентичність тестового зображення на основі вибраної метрики, такої як евклідова відстань, відстань Хемінга, нормалізована відстань Гемінга і т.д.

У контексті застосування для аутентифікації більшість алгоритмів перцептивного хешування можна загалом класифікувати наступним чином:

1. Хешування на основі інваріантних ознак (SVD-Hash)
2. Хешування на основі локальних ознак (D-Hash)
3. Хешування на основі зменшення розмірності (A-Hash)
4. Хешування на основі статистичних ознак (DCT-Hash)
5. Хешування на основі навчання (W-Hash).

Кожен алгоритм вартий уваги. І після розгляду усіх можна буде вибрати найбільш оптимальний, або комбінацію декількох, для створення алгоритму відеофінгерпринту.

2.4.1. A-Hash

Техніка середнього(Avarage) хешування, також відома як A-Hash, є простим і фундаментальним методом для створення перцептивних хешів для зображень. Ця техніка генерує хеш-значення зображення, акцентуючи увагу на низьких частотах, які відображають структуру зображення, відкидаючи високі частоти, що представляють деталі зображення. Для досягнення цього

A-Hash застосовує декілька перед обробкою операцій, включаючи розмиття, зміну розміру, скорочення кількості кольорів і нормалізацію.

Основною метою A-Hash є знаходження середнього кольору всіх значень пікселів у матриці зображення, розраховуючи середнє значення матриці. Ось кроки, які включає цей алгоритм:

1. Вхідне зображення змінюється на розмір 8 × 8 пікселів.

2. Виконується конвертація кольорового простору з RGB у відтінки сірого (YCbCr).

3. Розраховується середнє значення всіх значень яскравості в попередній матриці зображення.

4. Порівнюється кожен елемент матриці зображення із розрахованим середнім значенням. Формується нова бінарна матриця з 64 елементами, де 1 вказує на те, що інтенсивність елемента більша, ніж середнє значення, і 0 - навпаки.

5. Формується 64-бітний хеш, створюючи вектор із отриманої бінарної матриці, починаючи з верхнього лівого елемента і переходячи до нижнього правого.

Отриманий хеш може бути порівняний із хешами інших зображень для визначення "оцінки схожості" на основі метрики відстані між двома хешами.

AHash має декілька переваг, які роблять його популярним вибором для хешування зображень:

1. Швидкість та Ефективність. Простота A-Hash забезпечує швидке обчислення, навіть для великих наборів зображень. Етапи зміни розміру та перетворення в відтінки сірого зменшують обчислювальну складність, що робить його підходящим для застосувань в реальному часі.

2. Компактне Представлення. A-Hash генерує хеш фіксованого розміру, зазвичай 64 біти або 128 біт, незалежно від початкового розміру зображення. Це компактне представлення дозволяє ефективно зберігати та отримувати хеші зображень, що робить його ідеальним для пошуку схожих зображень у великих масштабах.

3. Стійкість до Трансформацій Зображень. A-Hash є відносно стійким до незначних трансформацій, таких як невеликі обертання, зсуви чи зміни яскравості. Навіть з цими трансформаціями генерований хеш залишається подібним, що дозволяє ефективно здійснювати пошук схожих зображень.

2.4.2 D-Hash

Техніка хешування різниць(Difference), відома також як D-Hash, є альтернативним методом і схожою на метод A-Hash. Подібно до A-Hash, D-Hash зосереджується на структурі зображення, досягаючи цього шляхом зменшення розміру зображення, тобто видаленням вищих частот зображення.

Основна різниця між цими двома техніками полягає в тому, що D-Hash генерує хеші, обчислюючи схожість хешу різниці між зображенням на основі зміни колірного градієнту між сусідніми пікселями в матриці зображення. Щодо алгоритму A-Hash, ми слідували таким же крокам, які вказані для реалізації алгоритму D-Hash. Ці кроки узагальнюються наступним чином:

1. зображення зменшується до розміру блоку 9 × 8 пікселів, що дає загальну кількість 72 пікселів;
2. зображення перетворюється у відтінки сірого кольору;
3. для кожного ряду ми порівнюємо різницю між кожними двома сусідніми пікселями, щоб отримати загальну кількість 8 різниць на ряд;
4. обчислюються 64 різниці для кожного зображення, які потім використовуються для побудови хешу зображення, так що, якщо значення різниці є від'ємним, то біт хешу встановлюється на 0, в іншому випадку - на 1. На кінець отримується 64-бітний хеш. Значення хешу представляють відносні зміни інтенсивності яскравості.

Щоб порівняти два хеші, достатньо підрахувати кількість бітів, які відрізняються. (Це відстань Хемінга.) Значення 0 вказує на однаковий хеш і, ймовірно, схоже зображення. Значення більше 10 ймовірно свідчить про

різне зображення, а значення між 1 і 10 потенційно може вказувати на варіацію.

Змінюючи розмір на 9x8 пікселів, ми отримуємо компактний блок зображення, де кожен піксель може взаємодіяти зі своїми сусідами. Під час обчислення різниці між сусідніми пікселями у кожному рядку в 9 пікселях, ми фактично прослідковуємо, як змінюється градієнт кольору вздовж цього рядка. Це допомагає виявляти структуру та напрям градієнта у зображенні.

Завдяки такому розміру блока 9x8 пікселів D-Hash може ефективно виділяти ключові риси зображення, які важливі для подальшого порівняння з іншими зображеннями та визначення їх подібності. Такий підхід робить D-Hash гнучким до змін розміру і пропорцій сторін зображення.

2.4.3 P-Hash

P-Hash (Perceptual Hash) - це метод створення хешів для зображень, який розширює метод A-Hash, додавши чутливість до особливостей людського зору. Основна ідея застосування P-Hash полягає в використанні Дискретного косинусного перетворення (DCT), щоб виділити найважливіші інформаційні деталі, які сприймає система людського зору.

Алгоритм методу хешування P-Hash:

1. Зменшення розміру: Спершу зображення зменшується до матриці розміром 32 × 32 пікселів. Це зменшення розміру допомагає знизити вплив високочастотних компонентів та деталей на хеш.

2. Перетворення в відтінки сірого: Зменшене зображення перетворюється у відтінки сірого, щоб спростити обробку та отримати більш чутливі результати.

3. Дискретне косинусне перетворення (DCT): Виконується DCT на зображенні розміром 32 × 32 пікселів. DCT дозволяє виділити низькочастотні складові зображення, які містять його основну інформацію.

4. Формування вектора з DCT-коефіцієнтів: Коефіцієнти DCT зліва вгорі до правого нижнього кута матриці розміром 32 × 32 об'єднуються у вектор довжиною 64.

5. Середнє значення коефіцієнтів: Обчислюється середнє значення всіх 64 коефіцієнтів DCT.

6. Порівняння та створення хешу: Кожен коефіцієнт DCT порівнюється зі середнім значенням. Якщо коефіцієнт більший за середнє значення, відповідний біт хешу встановлюється на 1, інакше він залишається 0. Це створює 64-бітовий бінарний хеш, який представляє зображення.

Переваги P-Hash включають високу чутливість до структурних особливостей зображення, а також можливість ефективного виявлення змін в зображеннях. Він також залишається стійким до змін масштабу, яскравості та контрастності, що робить його корисним для задач визначення схожості зображень.

2.4.4 W-Hash

Техніка хешування хвильового перетворення, відома як W-Hash, є методом хешування в області частот, який використовує Дискретне хвильове перетворення (DWT), щоб створювати перцептивні хеші. Вона базується на аналізі зображення в області хвильового перетворення, при цьому зберігається інформація про час.

Важливо зазначити, що це перетворення часто використовується для видалення зайвої інформації в даних з висококорельованими сусідніми значеннями, такими як пікселі на зображеннях. Основні кроки алгоритму W-Hash наступні:

1. Обчислення випадкового розміщення кожного піддіапазону зображення: Виконується розбиття кожного піддіапазону зображення на

задану кількість рівнів хвильового розкладання за допомогою хвильового перетворення Хаара

2. Квантування статистичного вектора: Отриманий вектор статистики піддається квантуванню за допомогою випадкового квантувальника.

3. Декодування квантованого вектора статистики: Квантований вектор статистики декодується за допомогою декодера з першим порядком виправлення помилок Ріда-Мюллера, щоб отримати бінарний хеш з довжиною n.

4. Додаткове декодування хешу: Отриманий проміжний хеш-значення проходить через ще один етап декодування лінійного коду з випадковими параметрами, щоб перетворити його в ще коротший хеш-код.

W-Hash використовує аналіз в області хвильового перетворення для створення стійкого хешу, який може бути використаний для виявлення схожості зображень і великих масштабів пошуку.

2.4.5 SVD-Hash

Хеш-функція, на основі розкладу сингулярних чисел, позначається як SVD-Hash і була вперше запропонована Козат та іншими [18]. Загальний механізм цієї техніки полягає в отриманні вторинного зображення з оригінального, використовуючи псевдовипадковий відбір ознак, які приблизно відображають напів-глобальні геометричні характеристики. Потім остаточні ознаки вилучаються і відповідно квантується для створення кінцевого хешу. Кроки реалізації алгоритму SVD-Hash можна узагальнити наступним чином:

1. Розбиття зображення на блоки: З вхідної матриці зображення розміром $n \times n$ формуються, можливо, перекриваються, блоки розміром $m \times m$.

2. Отримання вектора ознак для блоків: Для кожного отриманого блоку генерується відповідний вектор ознак, використовуючи сингулярний розклад (SVD).

3. Створення вторинного зображення: Здійснюється формування вторинного зображення, об'єднуючи всі проміжні вектори ознак шляхом псевдовипадкової комбінації.

4. Застосування кроків 1 і 2 до нового зображення: Повторно застосовуються кроки 1 і 2 до новоутвореного зображення.

5. Побудова кінцевого хеш-вектора: Нарешті, об'єднуються згенеровані вектори ознак з другого зображення, щоб сформувати кінцевий хеш-вектор.

SVD-Hash є інноваційним методом формування хешів, оскільки він використовує псевдовипадковий відбір ознак і сингулярний розклад, що враховують напів-глобальні геометричні особливості зображення. Ця методика дозволяє створювати хеш-коди, які відображають суттєві особливості зображення, і можуть бути ефективно використовувати для порівняння та визначення схожості між зображеннями.

Важливою перевагою SVD-Hash є його здатність виявляти схожість між зображеннями, навіть якщо вони мають невеликі геометричні або структурні відмінності. Це робить метод ефективним для розпізнавання об'єктів або сцен незалежно від їхньої точної позиції або розміру на зображенні.

SVD-Hash також використовує випадковий відбір ознак, що робить його відносно стійким до зміни кольору чи яскравості зображень, а також до деяких видів змін в освітленні чи фільтрації. Це робить метод досить надійним для виявлення схожості між зображеннями, які можуть бути трохи змінені через різні фактори.

У результаті, SVD-Hash виявляється ефективним інструментом для порівняння та аутентифікації зображень, і його застосування може бути

корисним в різних галузях, включаючи комп'ютерне бачення, обробку зображень та безпеку даних.

2.5. Бази даних і зберігання інформації

Задля правильної роботи системи, потрібно організовано шукати по великим об’ємам даних потрібну нам інформацію. Попередньо, можна сказати, що саме ми шукаємо: це ідентифікатор фільму.

Реляційна база даних, для кожного фільму буде видавати згенероване число, яке буде його ідентифікатором. Саме це число, можна зберігати під кожен хеш фрейму. Відповідно той, чий хеш зійдеться найбільш точно, матиме свій ідентифікатор фільму. Отримавши його, можна повернути користувачу всю інформацію з реляційної БД, яка його цікавить щодо фільму.

Отже, під значенням буде зберігатись згенерований ID фільму. Тоді під ключем має бути сам хеш. Схожість хешу, який уже зберігається разом із ID, із хешом, за допомогою якого проводиться сам пошук, буде співставним із схожістю двох рядків.

Для таких операцій, знадобиться NoSQL база даних("не тільки SQL"). Це категорія баз даних, які надають механізм для зберігання та пошуку даних, що моделюються не так, як традиційні реляційні бази даних. На відміну від реляційних баз даних, бази даних NoSQL не мають схем, що означає, що дані, які зберігаються в них, не повинні слідувати заздалегідь визначеній структурі. Така гнучкість особливо корисна при роботі з великими обсягами неструктурованих або напівструктурованих даних.

NoSQL бази даних краще підходять для такої системи, через наступні характеристики:

1. Горизонтальна масштабованість.

База даних NoSQL призначена для горизонтального масштабування, що означає можливість додавання більше серверів або вузлів до

інфраструктури бази даних, щоб впоратися зі зростаючим навантаженням даних. Така масштабованість добре підходить для розподілених і великомасштабних систем.

2. Продуктивність

Бази даних NoSQL часто забезпечують високу продуктивність для конкретних випадків використання, таких як інтенсивні операції читання та запису, завдяки своїй здатності розподіляти дані та робоче навантаження між кількома вузлами.

3. Висока доступність і відмовостійкість

Багато баз даних NoSQL розроблені з вбудованими функціями для забезпечення високої доступності та відмовостійкості. Вони можуть реплікувати дані на декілька вузлів, гарантуючи, що навіть якщо один вузол вийде з ладу, система продовжить працювати.

4. Масштабованість

Бази даних NoSQL пропонують різні моделі, такі як сховища ключ-значення, сховища документів, сховища сімейств стовпців і бази даних графів, кожна з яких пристосована до конкретних типів даних і сценаріїв використання. Це дозволяє краще оптимізувати їх відповідно до вимог програми.

5. Призначені для конкретних випадків використання

Бази даних NoSQL часто спеціалізовані для конкретних випадків використання. Наприклад, документно-орієнтовані бази даних, такі як MongoDB, добре підходять для зберігання та пошуку документів у форматі JSON, тоді як бази даних графів, такі як Neo4j, відмінно справляються з обробкою зв'язків між сутностями.

6. Економічність:

Бази даних NoSQL можуть бути більш економічно вигідними для певних додатків, особливо коли йдеться про роботу з великими обсягами даних. Їх розподілена природа дозволяє організаціям використовувати

звичайне апаратне забезпечення і горизонтально масштабуватися, що потенційно знижує витрати на інфраструктуру.

Всі ці фактори істотно вплинули на вибір, як вони ще називаються, ключ-значення бази даних.

2.6. Apache Cassandra

Apache Cassandra - це високомасштабована і розподілена база даних NoSQL, яка була розроблена для обробки великих обсягів даних на декількох розподілених серверах без єдиної точки відмови. Вона є частиною проекту, який спочатку розроблявся компанією Facebook, а згодом став open-source.

Рис. 2.3. Логотип бази даних Cassandra

Cassandra розроблена як розподілена система, що дозволяє їй горизонтально масштабуватися шляхом додавання нових вузлів до кластеру. Ця архітектура забезпечує безперешкодну роботу з великими наборами даних і високу швидкість запису та читання. Дані розподіляються між вузлами, що гарантує відсутність єдиної точки відмови. [3]

Також вона має лінійну масштабованість. Зі збільшенням кількості вузлів у кластері пропускна здатність і продуктивність системи зростають лінійно. Така масштабованість має вирішальне значення для обробки зростаючих обсягів даних і підтримки продуктивності при масштабуванні нашої системи.

Cassandra побудована з акцентом на високу доступність та відмовостійкість. Дані реплікуються на декілька вузлів, і в разі виходу з ладу одного з них Cassandra може продовжувати обслуговувати дані з інших реплік. Це гарантує, що система залишатиметься доступною навіть у випадку апаратних збоїв. Така архітектура Cassandra усуває єдину точку відмови. Кожен вузол у кластері ідентичний, і немає головних вузлів. Така децентралізована структура сприяє як відмовостійкості, так і високій доступності. Також зручною є мова запитів Cassandra (CQL) надає SQL-подібний інтерфейс для взаємодії з базами даних Cassandra.

Ця база даних дозволяє налаштовувати рівень узгодженості операцій читання та запису. Ця гнучкість є цінною у сценаріях, де можна надавати перевагу продуктивності над суворою узгодженістю даних або навпаки.

Окрім того, вона підтримує стиснення даних для мінімізації вимог до сховища. Вона також використовує стратегії ущільнення для ефективного управління простором сховища і підтримки оптимальної продуктивності з плином часу.

2.7. Ймовірнісні структури даних

При роботі з великими обсягами даних, задля ефективних операцій з пошуку даних можна застосовувати ймовірнісні структури даних. Це може сильно підвищити швидкодію системи.

Ймовірнісні структури даних – це тип структур даних, призначених для ефективної обробки та зберігання даних із керованою ймовірністю помилки. Ці структури жертвують ідеальною точністю на користь поліпшеної продуктивності та зменшених вимог до ресурсів. Основна ідея полягає в застосуванні ймовірнісних алгоритмів, які надають приблизні відповіді з певними гарантіями, замість точних рішень. [11]

Основний компроміс у ймовірнісних структурах даних полягає між точністю та ефективністю. Інтродукція випадковості та ймовірнісних

алгоритмів дозволяє цим структурам досягати високої продуктивності за рахунок іноді відхиленої точності.

Багато ймовірнісних структур даних покладаються на хеш-функції чи інші техніки випадковості. Хешування допомагає рівномірно розподіляти дані, зменшуючи ймовірність зіткнень та покращуючи ефективність операцій.

Ймовірнісні структури даних пропонують потужний підхід до управління та обробки великих наборів даних ефективно. За допомогою випадковості та прийняття контрольованої ймовірності невизначеності, ці структури дозволяють створювати масштабовані рішення в різноманітних застосунках, від баз даних до мережевих протоколів, де компроміс між точністю та ефективністю прийнятний. Розуміння конкретних вимог кожного випадку використання є ключовим для вибору найбільш підходящої ймовірнісної структури даних.

За реалізацію такої структури даних, я вибрав Фільтр Блума, який є класичним прикладом ймовірнісної структури даних. Він забезпечує економне тестування належності до множини, але може давати помилкові позитиви.

2.8. Фільтр Блума

Фільтр Блума - це компактна імовірнісна структура даних, призначена для перевірки того, чи є даний елемент членом множини. Він був введений Бертоном Говардом Блумом у 1970 році. Основна мета фільтра Блума - мінімізувати обсяг пам'яті, необхідний для перевірки на приналежність, що робить його особливо корисним у додатках, де пам'ять є критично важливим ресурсом.[13]

Якщо коротко, то це бітовий масив фіксованого розміру, спочатку заповнений нулями. Він використовує декілька хеш-функцій для зіставлення елементів з позиціями в масиві. Коли елемент вставляється, хеш-функції

генерують декілька індексів, а відповідні біти у цих позиціях встановлюються у 1.

Ключовою операцією фільтра Блума є перевірка на приналежність. Отримавши елемент, фільтр може швидко визначити, чи є він "можливо, у множині", "точно не у множині" або "можливо, не у множині". Однак він не може остаточно підтвердити приналежність до множини.

Фільтри Блума можуть давати хибні спрацьовування, вказуючи на те, що елемент є у множині, хоча це не так. Хибнонегативні результати неможливі; якщо фільтр стверджує, що елемент не входить до множини, це означає, що він є точним.

Ефективність фільтра Блума залежить від якості його хеш-функцій. В ідеалі, ці функції повинні створювати рівномірно розподілені хеш-значення, щоб мінімізувати колізії та хибні спрацьовування.

Розмір бітового масиву і кількість використовуваних хеш-функцій визначають компроміс між ефективністю використання місця і ймовірністю хибних спрацьовувань. Менші розміри і менша кількість хеш-функцій зменшують використання пам'яті, але збільшують ймовірність хибних спрацьовувань.

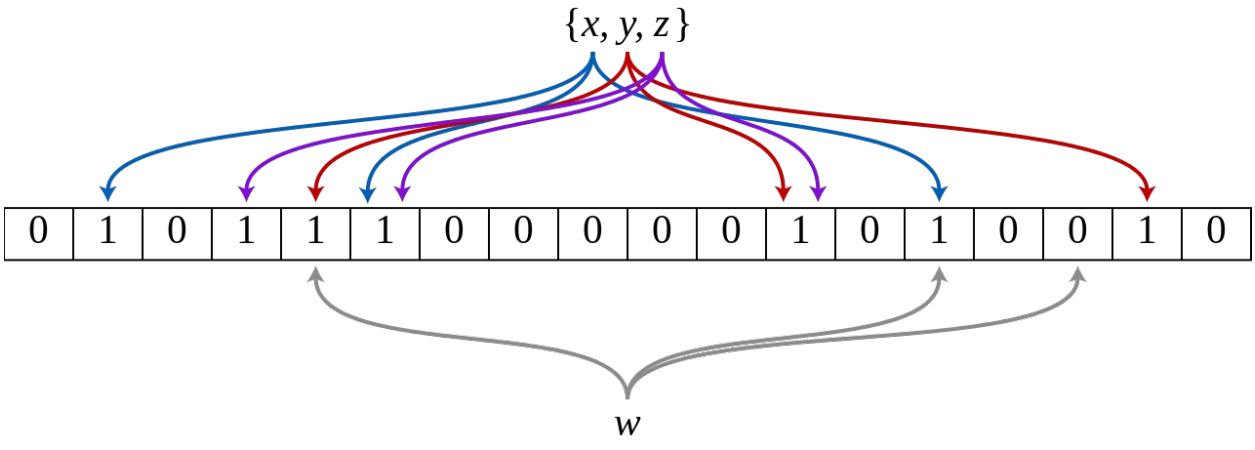

Рис. 2.4. Схема роботи Блум Фільтра

Традиційні фільтри Блума є статичними, тобто елементи не можуть бути видалені або розмір фільтра не може бути змінений без потенційної втрати точності. Динамічні фільтри Блума були запропоновані для усунення цих обмежень, дозволяючи додавати і видаляти елементи.

Фільтри Блума знаходять застосування в різних сферах, включаючи мережеві маршрутизатори для кешування, перевірки орфографії, аналізу послідовностей ДНК і розподілені системи для синхронізації даних і тестування на приналежність до великих наборів даних.

Колізії в хеш-функціях можуть призвести до помилкових спрацьовувань. Хоча це прийнятно в деяких додатках, це може бути проблематично в контекстах, чутливих до безпеки.

Отже, фільтри Блума пропонують ефективне рішення для перевірки на приналежність у ситуаціях, коли хибні спрацьовування є прийнятними. Вони широко застосовуються в різних програмах завдяки своїй простоті, швидкості та здатності економити ресурси пам'яті.

2.9. Кореляційні методи визначення схожості хешів

Кореляційні методи визначення схожості хешів є важливим елементом у системах відеоідентифікації, особливо коли мова йде про порівняння та виявлення схожих відеофрагментів або зображень. Ці методи забезпечують можливість порівняти хеші (унікальні цифрові відбитки), створені для різних відео, щоб визначити ступінь їхньої подібності або відмінності. Основна мета цих методів - забезпечити швидке та точне порівняння великих масивів даних без необхідності перегляду всього відео. [12]

Один з основних кореляційних методів - це визначення відстані Хемінга. Вона вимірює кількість відмінностей між двома бінарними рядками однакової довжини, що представляють собою хеші. Чим менша відстань Хемінга, тим більша схожість між двома порівнюваними хешами. Цей метод

є особливо ефективним для швидкого порівняння хешів, оскільки він вимагає обчислення лише кількості відмінних бітів у двох хешах.

Інший важливий аспект кореляційних методів - це здатність виявляти не лише точні відповідності, але й схожість на основі структурних чи візуальних характеристик. Наприклад, це може бути корисним при виявленні відео, що містять однакові сцени, але з різними рівнями якості або зміненими параметрами відео (як-от кадрування або кольоровість).

Використання кореляційних методів у системах відеоідентифікації важливе також для забезпечення захисту авторських прав і боротьби з піратством. Ці методи дозволяють автоматично виявляти та порівнювати контент, що містить ознаки порушення авторських прав, з оригінальними версіями.

У загальному, кореляційні методи визначення схожості хешів є фундаментальними для розвитку та ефективності сучасних систем відеоідентифікації, забезпечуючи точність, швидкість та надійність у пошуку та аналізі відеоданих.

ВИСНОВОК ДО РОЗДІЛУ 2

У рамках цього розділу було розглянуто різні аспекти та техніки, що стосуються системи відеоідентифікації фільмів. Починаючи з загального огляду систем відеоідентифікації, ми вивчили, як ці системи еволюціонували та яку роль вони відіграють у сучасному світі медіа.

Аналіз аналогів систем розпізнавання контенту, таких як Shazam та Google Lens, демонструє широкий спектр застосувань відеоідентифікації, від музики до візуальних образів. Це підкреслює важливість інновацій у цій області та їх вплив на повсякденне життя користувачів.

Особливу увагу приділено поняттю відеовідбитків пальців, або фінгерпринтингу, яке відіграє ключову роль у відеоідентифікації. Було розглянуто різні варіанти використання відеофінгерпринтингу, включаючи захист авторських прав та контент-аналітику.

Розділ також містить інформацію про перцептивне хешування, включаючи A-Hash, D-Hash, P-Hash, W-Hash та SVD-Hash. Ці методи представляють собою сучасні підходи до створення унікальних ідентифікаторів відеоконтенту, що важливо для ефективного зберігання та пошуку даних.

У контексті зберігання інформації ми вивчили роль баз даних, зокрема Apache Cassandra, та їх важливість у зберіганні великих обсягів відеоданих. Також ми зосередили увагу на ймовірнісних структурах даних та Фільтрі Блума, які є ключовими для оптимізації процесів пошуку та доступу до інформації.

Загалом, цей розділ надає глибокий огляд розвитку та поточного стану систем відеоідентифікації фільмів, підкреслюючи їх важливість у сучасному цифровому ландшафті. Ми виявили, що ці системи не тільки підвищують ефективність роботи з відеоданими, але й відкривають нові можливості для інновацій та креативних рішень у сфері медіа.

РОЗДІЛ 3
АЛГОРИТМИ, БАЗИ ДАНИХ І АНАЛІЗ ХЕШІВ

3.1. Вибір алгоритму хешування

Аналізуючи можливі варіанти, для розробки прототипу системи розпізнавання фільмів, було вибрано хешування на основі алгоритму A-Hash, так як воно простіше в реалізації, та розмір хешу кожного фрейму буде мінімальним.

Простота не тільки пришвидшить імплементацію, але й забезпечить більшу ефективність обчислень, що зробить його придатним для програм реального часу та великомасштабних баз даних зображень.

Для таких завдань, як дедуплікація вмісту, де метою є виявлення та видалення дублікатів або дуже схожого вмісту, A-хешування може ефективно фіксувати сприйнятливу подібність. Його стійкість до незначних змін гарантує, що подібні зображення розпізнаються як такі, навіть якщо вони зазнають незначних змін.

Хоча A-хешування може не підходити для всіх завдань обробки зображень, його переваги з точки зору простоти, надійності та обчислювальної ефективності роблять його переконливим вибором для перцептивного хешування, особливо в програмах, де ці характеристики цінуються.

3.2. Перший хеш та його характеристики

Так як відео, це послідовність кадрів, для прикладу можна взяти хеш з однієї картинки. Щоб проілюструвати генерацію геша в дії, візьмемо кадр з фільму «Blade Runner 2049», який був випущений 2017 року.

Рис. 3.1. Раян Гослінг в ролі репліканта

Для початку потрібно кадр зменшити до відповідного розміру. Існує безліч інструментів для таких цілей. Я скористаюсь термінальною утилітою під назвою imagemagick. За допомогою наступних команд, зображення було стиснуто до розміру 16 на 16 пікселів.

```
bohdan@bs-dell:~/Downloads$ identify bladerunner.png
bladerunner.png PNG 1000x562 1000x562+0+0 8-bit sRGB 537131B 0.000u 0:00.000
bohdan@bs-dell:~/Downloads$ convert bladerunner.png -resize 16x16 bladerunner-cropped.png
bohdan@bs-dell:~/Downloads$
```

Рис. 3.2. Виконання команди на зміну розмірності зображення

Результатом даної операції, стала картинка ідентична до оригінальної, розміром 16x16. Дане зображення, доволі темне, тому для подальших розахунів потрібно вирахувати середню яскравість пікселів.

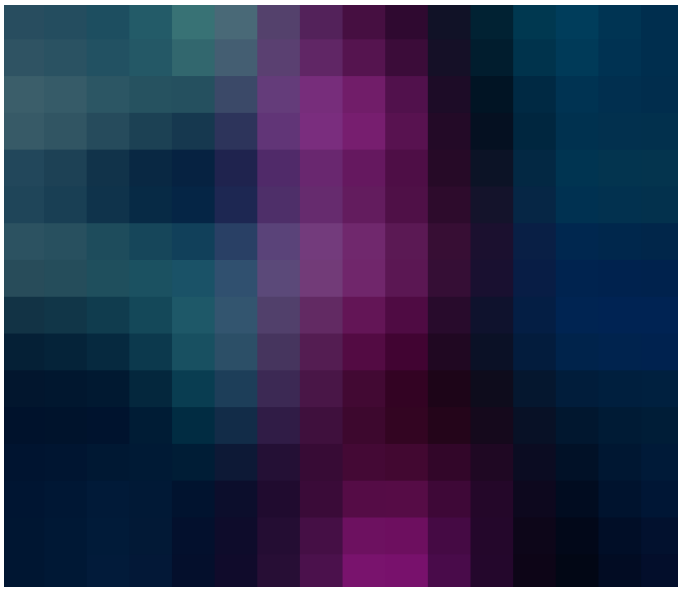

Рис. 3.3. Стиснуте зображення

Щоб розрахувати яскравість, потрібно перевести зображення в Grayscale режим. Для цього можна використати інструмент з попереднього кроку ImageMagick.

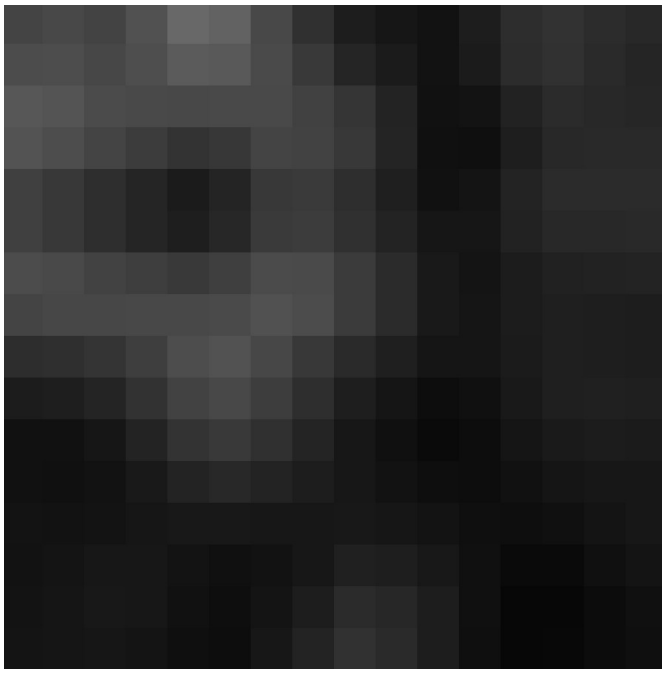

Рис. 3.4. Зображення лише з відтінками сірого

Потім, сумуються значення усіх пікселів, і ділиться на їх кількість.

```
bohdan@bs-dell:~/Downloads$ convert br-cropped.png -colorspace Gray -resize 16x16! input_resized.jpg
bohdan@bs-dell:~/Downloads$ avarage_pixel=$(convert input_resized.jpg -format "%[fx:mean]" info:)
bohdan@bs-dell:~/Downloads$ echo $avarage_pixel
0.151961
bohdan@bs-dell:~/Downloads$
```

Рис. 3.5. Обчислення середнього значення пікселя

Отримавши величину середнього пікселя в 0.15196, можна скласти матрицю, порівнявши всі пікселі зі значенням середнього. Якщо більше, результатом буде «1», інакше «0». Або ж для візуалізації перетворимо в чорні та білі пікселі відповідно.

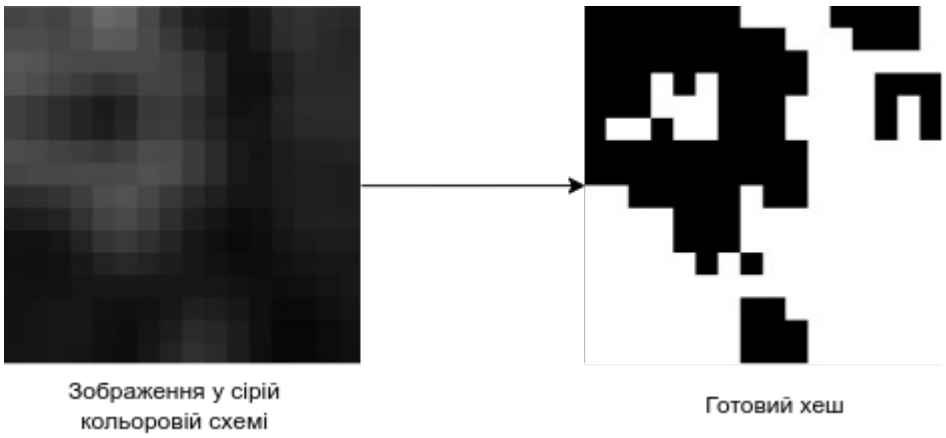

Рис. 3.6. Перцептивний хеш зображення

Даний хеш, по суті складається з 256 послідовних біта. На його основі зробимо 256-бітне число, та перетворимо у HEX формат(16-кова система числення). Результатом став наступний рядок: «fe1eff8effc0ebc7e3859385ffc0ffc03ec00e000e0005000000018001c001c0»

Саме такі дані ми будемо зберігати послідовно. І отримуючи якесь коротке відео, будемо робити всі ті ж самі дії для отримання послідовного

набору хешів такої форми, та шукати найбільш вдале співпадіння серед великого набору даних.

3.3 Зберігання метаданих фільму

Так як дана система буде зберігати доволі великі об’єми даних, потрібно завчасно обговорити спосіб їх збереження. Окрім того, можна вдатись до різних механізмів оптимізації пошуку та зберігання даних. Перш за все, отримавши якийсь фільм, потрібно зберегти його назву, метадату, рік випуску і таке інше. Для таких задач чудово підійде звичайна реляційна база даних, наприклад PostgreSQL чи MySQL.

Реляційні бази даних призначені для зберігання структурованих даних у таблицях, де кожна таблиця складається з рядків і стовпців. У нашому випадку, кожен елемент метаданих відеофільму, такий як назва, рік, ім'я продюсера та постер, можна організувати в таблицю з окремими стовпчиками для кожного атрибуту. Така структурована організація сприяє ефективному зберіганню та пошуку інформації.

Окрім того, реляційні бази даних чудово справляються з управлінням зв'язками між різними сутностями. Наприклад, у базі даних фільмів можна тримати окремі таблиці для фільмів, продюсерів та інших пов'язаних сутностей. За допомогою зовнішніх ключів встановити зв'язки між цими таблицями. Це дозволяє легко пов'язати фільм з його продюсером, забезпечуючи цілісність даних і створюючи складні запити, які включають пов'язану інформацію.

Це корисно для зберігання даних, які будуть в більшій мірі орієнтовані на користувача нашої системи. Наприклад, якщо хтось знайде по сніппету потрібний йому фільм, для кращого відображення, користувачу буде показаний постер з фільму, акторами у складі, можливі схожі фільми тощо.

Маючи на руках приклад готового хешу, можна встановити можливий спосіб зберігання даних та алгоритмів для пошуку серед них. Так як кожен

хеш, це 256-бітне число, записане у 16 системі числення, можна спростити варіації вигляду ключа, під яким буде зберігатись дані про фільм.

3.4. Визначення потрібного обсягу даних для зберігання

Так як система буде фокусуватись саме на розпізнаванні фільмів, можна визначити середню довжину відео. Хоча зараз спостерігається тенденція до збільшення середньої довжини фільму, зазвичай він триває півтори-дві години. Ця цифра потрібна, щоб визначити середній розмір хешу, який потрібно зберігати. Це в свою чергу, впливатиме на вибір оптимального сховища, та, можливо, більш специфічного алгоритму пошуку.

Розрахувати скільки пам’яті буде буде займає в середньому хеш відео, можна наступним чином:

1. Середня довжина відео 2 години, або ж 120 хвилин, або ж 7200 секунд. У кожній секунді зазвичай близько 30 кадрів(фреймів). Загальна кількість кадрів у фільмі тоді: 7200 * 30 = 216 000.
2. До кожного кадру, буде застосовуватись хеш-функція A-hash з стисканням кадру до розміру 8*8 пікселів(або 16*16). Її результат відомий – матриця з 64 елементів з можливими значеннями 0 або 1. Іншими словами, вага хешу кожного кадру: 64 біта(або 256).
3. Послідовно кадри мало змінюються дуже мало, тому, в цілях оптимізації, в межах однієї секунди, ми будемо відсіювати кадри, які схожі один на одного більше ніж на 90%. В залежності від динамічності відео, кількість кадрів потрібна для хешування буде змінюватись. Для спокійних фільмів, без частої зміни картинки, Кількість кадрів може зменшитись до 2 на секунду. Як наслідок, кількість кадрів потрібна для хешування може зменшитись приблизно у 15 разів.
4. В результаті, об’єм пам’яті, потрібен для зберігання хешу середньо-статистичного фільму буде дорівнювати 13 824 000 бітів, що

еквівалентно 1 728 000 байтів(1.73 MB), але з оптимізацією, що наведена вище ця цифра буде дорівнювати приблизно 0.5 MB.

Слід зауважити, що параметр схожості, для пропускання однакових кадрів, має бути параметризованим. Оскільки при хешуванні кадр стискається до надмалих розмірів, і трансформується в чорно-білу матрицю, інформації сам кадр передає небагато. Тому нас цікавить більше саме послідовність таких кадрів один за одним. Адже вона буде унікальна для кожного фільма, але якщо брати лише її невеликий відрізок – можливі колізії. Чим більше інформації буде нести один кадр, трансформований в хеш, тим менше кадрів потрібно для ідентифікації конкретного відео.

Відповідно, такі параметри як розмір матриці хеша, мінімальна кількість кадрів в секунду, що підлягатиме хешуванню та параметр схожості для відсіювання дублікатів, мусять бути конфігурабельними на початковому етапі, щоб досягти максимальної продуктивності при обмеженому розмірі ресурів.

Попередні розрахунки показують, що зберігання хеша кожного фрейму, розміром 16 на 16 пікселів, для фільму, тривалістю в 2 години, обійдеться системі в 55,296 MB. Але з оптимізацією кожного кадру, відсіювання схожих підряд фреймів, цей розмір може буде зменшений до ~0.4 MB, що цілком підходить для зберігання майже будь якими БД, навіть якщо таких будуть тисячі.

3.5. Алгоритми пошуку та ідентифікації

Об’єкт, який ми будемо зберігати буде складний та багаторівневий, з додатковими структурами даних для оптимізацій операції пошуку. Перш за все, він буде зберігати ідентифікатор фільму, що знаходиться у реляційній базі даних з супутньою інформацією.

Друга властивість, яка знадобиться для розрахунків, це порядок фрейма, для якого рахувався A-hash.

Третя властивість – сам хеш, який складається з 256-біт.

Кожен фільм – це список таких відеохешів, кількість яких може буде досить велика, в залежності від тривалості та динамічності фільму. Кількість фільмів теж може бути доволі великою та вимірюватись тисячами. Як наслідок, пошук серед такого великого об’єму даних буде складним.

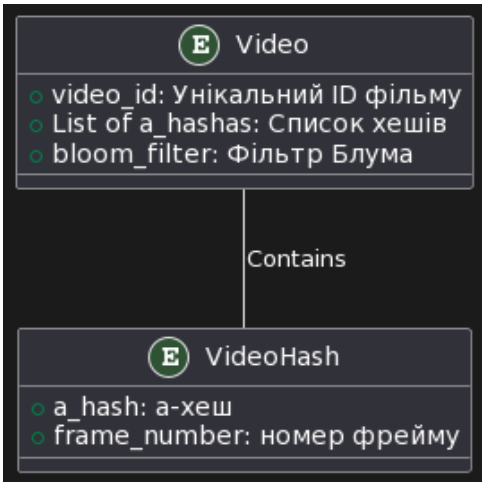

Рис. 3.7. Схема зберігання хешів

Тут нам знадобиться Фільтр Блума, описаний у попередньому розділі. Вибір розміру фільтра Блума (m) залежить від декількох факторів, включаючи кількість елементів, які треба додати до фільтра (n), бажаного рівня помилкових спрацьовувань (p) і кількості використовуваних хеш-функцій (k).

Розрахунок m задається формулою:

$$m = -\frac{n * \ln(p)}{\ln(2)^2}$$

За кількістю елементів буде виступати кількість a-хешів, а рівень помилок зазвичай варіюється від 0.001 до 0.01. В даному випадку, було вибрано 0.01. Як наслідок, кількість бітів, потрібна для фільму з 100 000 фреймів з ймовірністю помилки у 1% буде дорівнювати приблизно 959,375, або ж, округливши, один мільйон бітів(0.125 MB).

Створюючи фільтр Блума з 1 000 000 біт, спочатку всі біти потрібно встановити на 0. Потім, потрібно визначити хеш-функції для зіставлення A-хешів з позиціями у фільтрі Блума. Я буду використовувати MurmurHash3, так як це популярна, швидка та ефективна хеш-функція, але з різним сідом.

Я буду використовувати хеш функції з сідом 42 та 87. Але це число можна буде змінити в майбутньою за потреби.

Наприклад, якщо Ahash буде дорівнювати:

«fe1eff8effc0ebc7e3859385ffc0ffc03ec00e000e0005000000018001c001c0» , то при застосуванні хеш функцій вийдуть наступні результати:

murmurHash3.hash(ahash, 42) = 42 623

murmurHash3.hash(ahash, 87) = 793 861

Далі потрібно встановити біти в отриманих індексах у фільтрі Блума рівними 1.

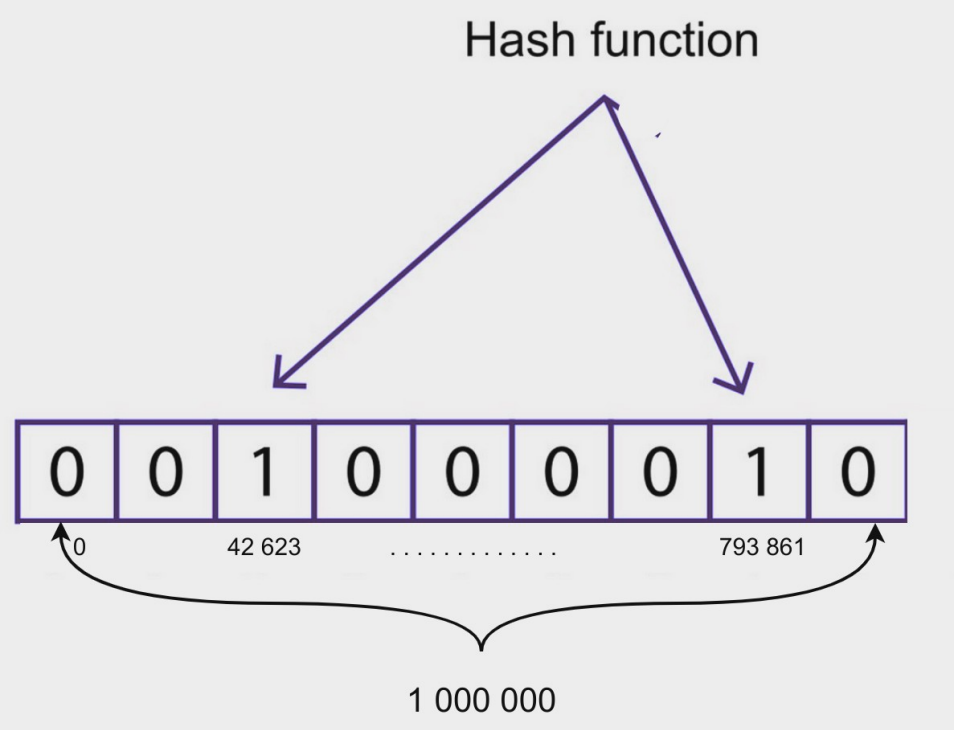

Рис. 3.8. Виставлення бітів під індексом в занчення «1»

Якщо хеш-функція поверне значення більше за кількість бітів у фільтрі, потрібно обрізати його на ту ж саму кількість бітів, що у фільтрі, стільки раз, допоки воно не стане менше його.

Ці ж самі дії потрібно повторити для кожного A-хеша з фільму. Це допоможе визначити ймовірну приналежність кадру до його фільму, але не може дати 100% гарантію. Тому для точнішої ідентифікації, потрібно скористуватись додатковими перевірками.

Отримуючи фрагмент відео, потрібно зробити всі ті ж самі операції: розбиття на кадри, перетворення їх за допомогою A-hash алгоритму, визначити хешкод кожного A-хешу тими ж самими функціями хешування.

Далі потрібно перевірити, чи всі біти в отриманих індексах у фільтрі Bloom дорівнюють 1. Якщо хоча б один біт дорівнює 0, відео точно не потрапляє у фільтр. Якщо всі біти дорівнюють 1, відео може бути у фільтрі (з невеликою ймовірністю помилкового спрацьовування).

Ця перевірка на наявність кадру у фільтрі блума допоможе відсіяти більшість фільмів, які не підпадають під наші хеші. Задля додаткової перевірки, можна передавати не один хеш в пошук, а відразу декілька з різних частин відео. Той фільм, який має найбільше співпадінь у фільтрі, і буде кандидатом №1 на повернення.

3.6. Момент, з якого починати порівняння

Отримавши один або декілька можливих варіантів, потрібно точно визначити схожість даного відрізку відео, що прийшов нам від користувача, з найденим фільмом, що зберігається у нас в базі даних.

Для візьмемо вопрядковані хеші, і розіб’ємо їх на частини, наприклад по 10 штук. Далі для цих 10 хешів, потрібно обичислити умовну «похідну», а саме характеристику, щодо зміни цих кадрів. Для цього потрібно узяти ці хеши, і «накласти» один на одного. Так пікселі, які повторюються, будуть більш підігріті, ніж інші. Ті, що повторювались найчастіше, наприклад 70%

від максимуму, потрібно зберегти з відповідними індексами. Саме це і буде виступати характеристикою для цілої послідовності кадрів.

Але сам цей масив даних не буде виступати ідентифікатором конкретного відео. Його ціль полягає в іншому: пошук ймовірного місця, з якого можна проводити порівняння хеш-кадрів.

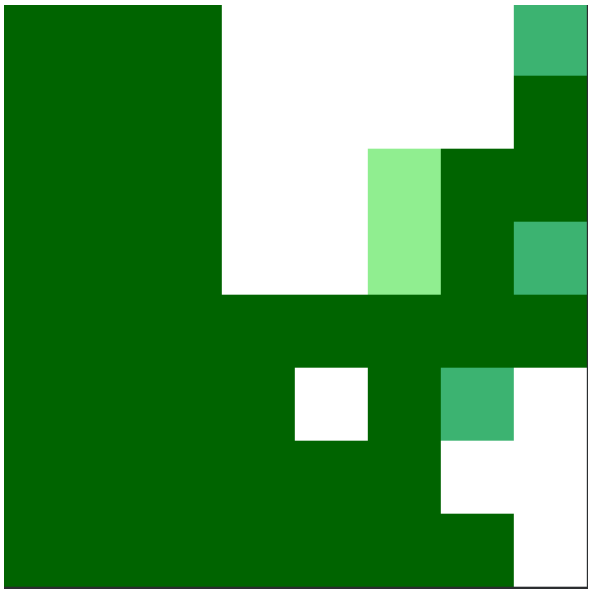

Рис. 3.9. Густина хешів(темніший – більш часто повторюваний)

Виділивши лише найтемніші, можна їх представити у вигляді окремого числа, де кожен темніший піксель, насправді означає «1», а решта «0». Представлення має бути у тій послідовності, яка використовувалась і при створенні A-хешів. Відповідно, це число буде насправді ключем, за допомогою якого ми будемо шукати такий же самий патерн.

Після цього, потібно знайти такий же патерн, у структурі, якій ми підготували раніше, разом зі створення хешів. Це буде ідентифікатором кадрів, з якого потрібно починати порівнювати хеші, що ми отримали з початкового відео. І Навіть якщо отримаємо декілька варантів, можна всі варіанти перевіряти асинхронно, що значно прискорить роботу.

3.7. Перевірка схожості хешів

Після того, як було визначено ймовірні фільми за допомогою фільтра Блума, а ймовірні епізоди з фільму, що підходять під вхідне відео, знайдені за попереднім алгоритмом, можна рахувати характеристику схожості вхідних кадрів. Для цього потрібно уточнити як саме можна визначити чи схожі хеші, і на скільки відсотків. Так як всі хеші, це насправді 64-біта, можна визначити відстань Хемінга, на бітових даних, хоча те ж саме можна робити і для рядкового представлення.

Відстань Хемінга є математичною метрикою, яка вимірює різницю між двома бітовими послідовностями однакової довжини. Це поняття було вперше введено в 1950 році Річардом Хемінгом та знаходить широке застосування в областях теорії інформації, телекомунікацій, кодування та криптографії. Основна ідея полягає в підрахунку кількості бітів, які відрізняються між двома послідовностями.

Відстань Хемінга є математичною метрикою, яка вимірює різницю між двома бітовими послідовностями однакової довжини. Це поняття було вперше введено в 1950 році Річардом Хемінгом та знаходить широке застосування в областях теорії інформації, телекомунікацій, кодування та криптографії. Основна ідея полягає в підрахунку кількості бітів, які відрізняються між двома послідовностями.

Відстань Хемінга між двома бітовими послідовностями A та B довжини n визначається як сума бітів, на яких ці послідовності відрізняються. Якщо A = (a_1, a_2, ..., a_n) та B = (b_1, b_2, ..., b_n), то формула відстані Хемінга H(A, B) виглядає наступним чином:

$$H(A,B)=\sum i=1^{n}(a_i \neq b_i)$$

Відстань Хемінга може приймати значення від 0 (якщо послідовності ідентичні) до n (якщо всі біти відрізняються).

У теорії кодування відстань Хеммінга має вирішальне значення для виявлення та виправлення помилок. Вона лежить в основі кодів Хеммінга,

класу кодів, що виправляють помилки, які можуть виявляти і виправляти однобітові помилки в даних, що передаються. Аналізуючи відстань Хеммінга між отриманим та очікуваним кодовим словом, можна виявити та виправити помилки, внесені під час передачі даних.

Хоча відстань Хеммінга ефективна для порівняння рядків однакової довжини, вона має обмеження. Одним з таких обмежень є його чутливість до довжини порівнюваних рядків. Якщо рядки відрізняються за довжиною, виникає необхідність або вкоротити, або накласти прокладку на коротшу струну, щоб зробити порівняння осмисленим. Крім того, відстань Хеммінга фокусується виключно на позиційних відмінностях і може не враховувати більш складні структурні розбіжності між послідовностями.

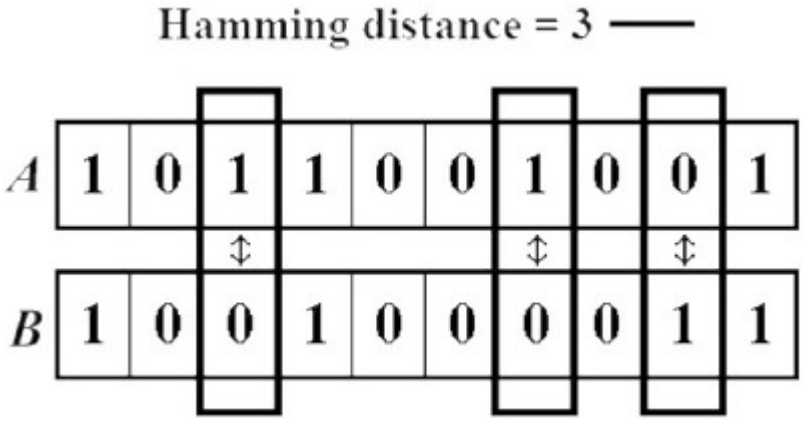

Рис. 3.10. Відстань Хемінга для бітової послідовності

Отже, відстань Хеммінга є цінною метрикою для вимірювання несхожості між двійковими послідовностями, що знаходить застосування у виправленні помилок, теорії кодування та криптографії. Її простота та ефективність роблять її фундаментальним інструментом у різних галузях, надаючи уявлення про розбіжності або помилки, присутні в рядках однакової довжини.

Враховуючи дані, які у нас є, схожість двох хешів можна вирахувати, застосувавши операцію XOR, та порахувавши кількість бітів, що відрізняються.

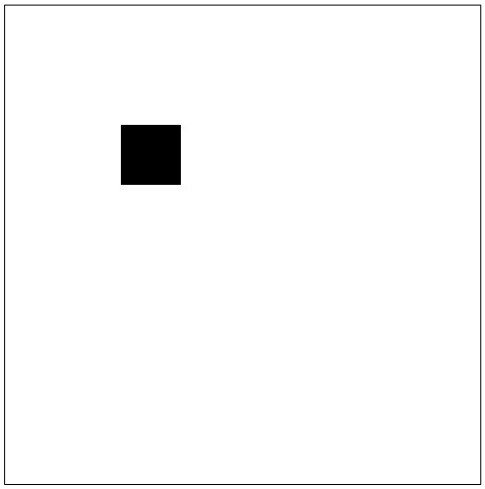

Рис. 3.11. Результат XOR на два хеша, що відрізняютсья пікселем

Якщо різниця лише в один піксель, а хеші мають всього 64, відсоток схожості буде 1/64, або 100% - 1,5625 ~= 98,43%. Дуже схоже! Але для того, щоб визначити чи це той самий фільм, потрібно встановити рамки, за якими можна відсіювати малоймовірні варіанти.

Наприклад, якщо ми отримали за одним відео два ймовірних фільми, і перевірили з їх хеші для певних сцен на схожість. Потрібно вибрати кращого з них. Але якщо обидва мали гранично малу схожість, то потрібно визначити такий відсоток схожості, який можна було вважати кандидата, а не похибку обчислень. Цей параметр можна буде змінювати, тому для експерементів я вибрав 80%.

Отже, якщо схожість кадрів в середньому буде переважати 80% для однієї чи декількох сцен(в залежності від довжини вхідного відео), то кандидата на повернення користувачу було знайдено.

3.8. Алгоритмічна складність

Для вирахування алгоритмічної складності системи відеоідентифікації, яка включає алгоритми хешування, Фільтр Блума, пошук схожих сцен та перевірку схожості хешів за допомогою відстані Гемінга, потрібно врахувати кілька ключових елементів.

Створення A-Hash має відносно низьку алгоритмічну складність. Він зазвичай включає зменшення розміру зображення, перетворення відтінків сірого, та обчислення середнього значення кольору. Потім кожен піксель порівнюється з середнім значенням для створення бінарного хешу. Це можна розглядати як O(n) для n пікселів у зображенні, але з огляду на зменшення розміру, це число завжди дорівнює 64. Але так як цю операцію потрібно застосувати до кожного кадру з вхідного відео, а це зазвичай не відома кількість, То алгоритмічна складність буде дорівнювати O(n).

Алгоритмічна складність Фільтра Блума для перевірки наявності елементу має алгоритмічну складність O(k), де k - кількість хеш-функцій. Оскільки кількість хеш-функцій зазвичай невелика і фіксована, а у нашому випадку завжди дорівнює 2, складність константна.

Складність пошуку сцен у фільмі, у відповідності до нашого алгоритму вище, буде дорівнювати O(h*n), де h – це кількість кадрів в одній сцені. Після цього буде етап накладення хешів один на одного та рахування кількості співпадінь для кожного біта(n).

Алгоритмічна складність рахування схожості хешів, використовуючи Відстань Гемінга має O(n), де n - довжина хешу. Вона полягає у порівнянні двох рядків однакової довжини та підрахунку кількості позицій, в яких відповідні символи відрізняються.

Загалом, ця система відеоідентифікації здається ефективною з точки зору алгоритмічної складності, особливо для процесів A-Hash і відстані Гемінга. Фільтр Блума також є високоефективним для операцій перевірки наявності. Найбільші виклики для ефективності можуть виникнути у фазі

пошуку схожих сцен, де складність може залежати від обраного методу аналізу кадрів.

3.9. Огляд прототипу системи

Моя система складається з двох компонентів. Перший займається саме кодуванням та збереженням відео у базу даних. Другий займається кодуванням вхідного відео та пошуку його наймовірніших кандидатів.

Звичайно, кодування відбувається точно таким самим чином, задля збереження детермінованості системи.

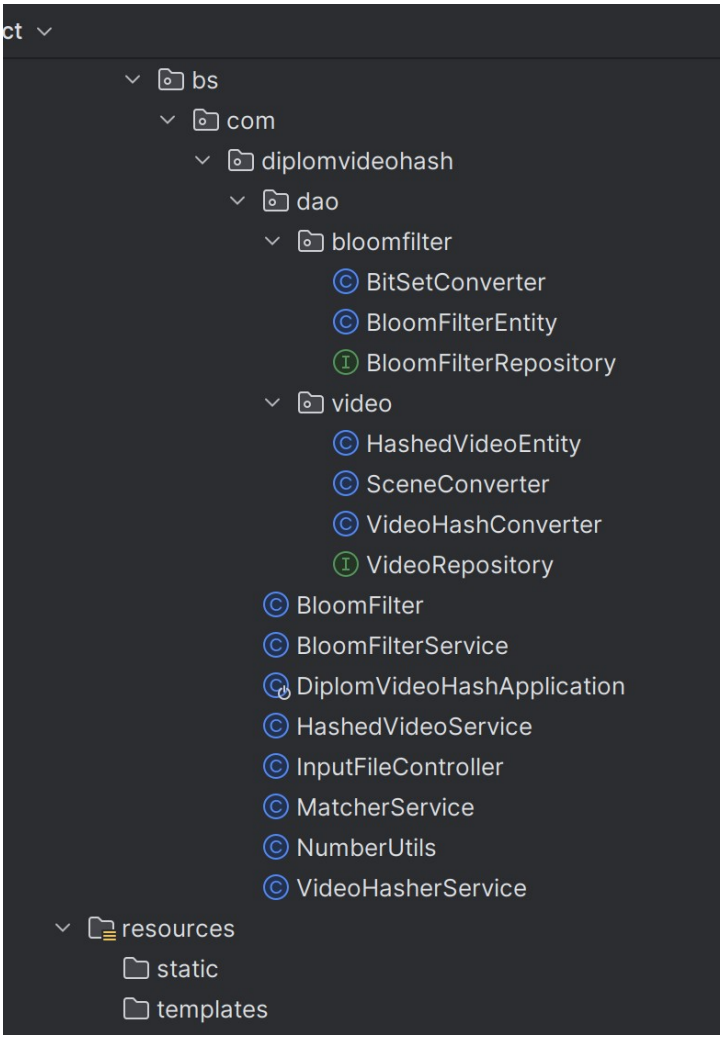

Рис. 3.12. Структура проекту

Прототип побудовано за допомогою мови програмування Java, використовуючи Spring фреймворк для ін’єкції залежностей, JavaCV для дефрагментування відео.

Клас InputFileController - Це Spring REST контролер, який обробляє HTTP запити. Він має два ендпойнти: хешування та пошук відео.

Перший для прийняття назви файлу відео, розташованого локально, ініціювання його хешування, створення додаткових структур та збереження у базу даних. Схема його роботи зображеня на рис. 3.13.

Клас VideoHasherService займається обробкою та хешуванням відеофайлів наступним чином: він використовує FFmpegFrameGrabber для вилучення кадрів з відео. Кожен вилучений кадр проходить різні етапи обробки. Це включає зміну розміру, перетворення у відтінки сірого, визначення середнього значення пікселя. Оброблені кадри потім використовуються для генерації хешів. Як алгоритм перцептивного хешування, було використано A-хеш.

Далі для всіх новостворених послідовних хешів, був створений фільтр Блума, за допомогою BitSet структури даних. Ця структура даних, яка знаходиться в стандартному пакеті java.util точно відображає функціонал, який потрібен для фільтра Блума – послідовність бітів певної довжини.

Далі йде процес створення розбиття хешів на сцени – сума співпадінь по бітам для 10 хешів підрят. Всі співпадіння рахуються, та записуються лише 30% найбільших значень.

Далі всі знайдені сцени перевіряються на відстань Гемінга, та записуються в список, найкращі співпадіння(найменша відстань) мають найбільший пріоритет.

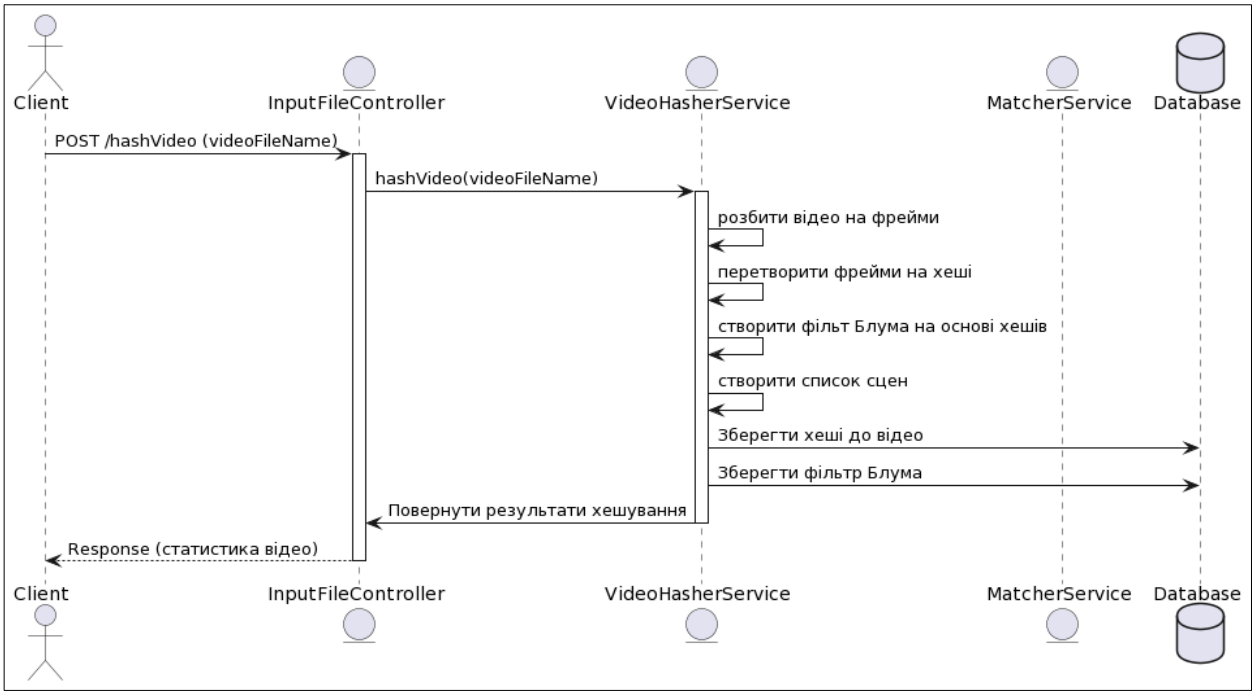

Рис. 3.13. Діаграма дій при хешуванні відео

Наступний функціонал мого прототипа це пошук відео по його фрагменту. Він реалізований дещо схожим чином.

Спочатку, фрагмент відео потрібно захешувати точно таким самим чином, як і решту відео. Отримані хеші потрібно перевірити через фільтри Блума, які були створені раніше для інших відео. Знайшовши співпадіння, кращі з результатів, зберігаються. Кожен з цих фільтрів мав прив’язку до ідендифікатора відео. Тому наступним кроком буде діставання з бази даних всіх хешів та сцен по цим відео.

Отримавши фрагмент відео, та дістивши його хеші, потрібно знайти відповідну сцену з оригінального відео, та запам’ятати фрейм, з якого можливо проводити порівняння.

Знайшовши фрагменти відео, можна розпочати порівняння хешів. Для прикладу я брав кількість хешів, яка була використана при створенні відбитків сцен – 10. До 10 хешів, з відповілного фрейму буде застосовуватись операція XOR, та рахуватись кількість бітів, що відрізняються. Приводяться до відсоткового співвідношення, та рахується середній відсоток схожості для 10 сцен. Так для кожної знайденої сцени, та кожного знайденого відео.

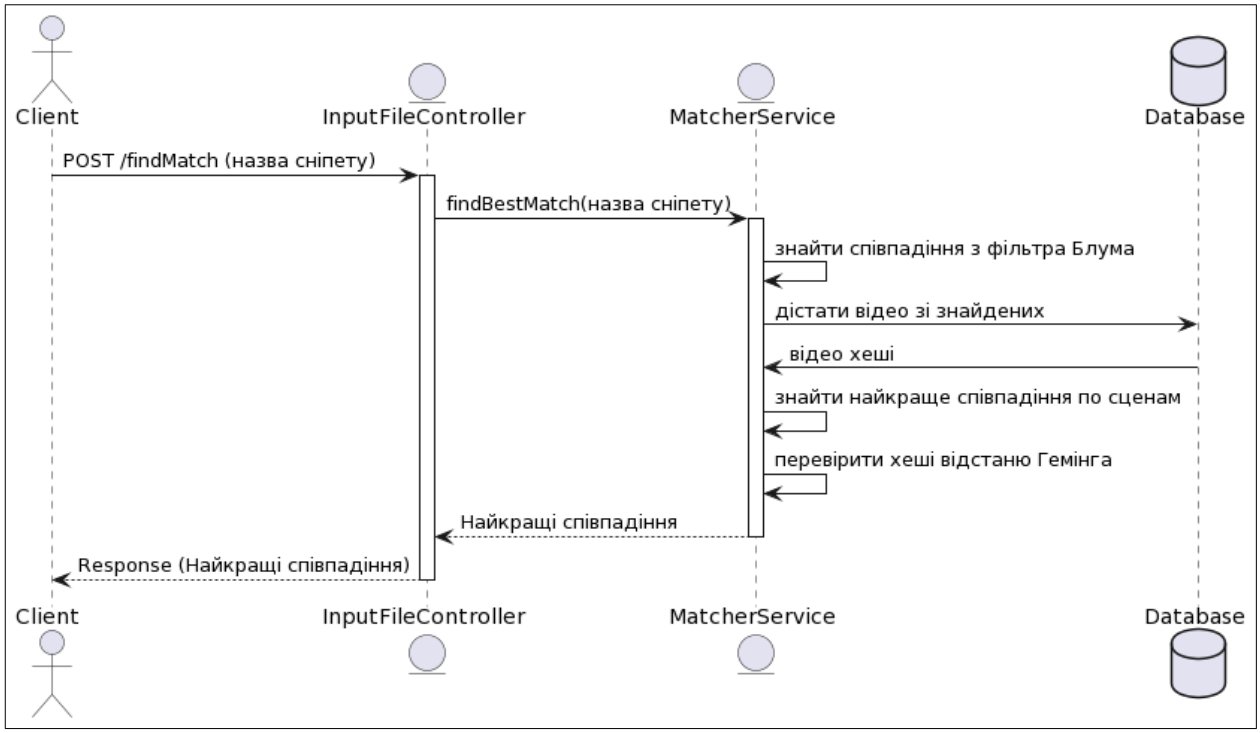

Рис. 3.14. Схема роботи при пошуку відео по фрагменту

По ходу роботи програми, я логував всі найважливіші моменти. Вивід в консоль був наступний при хешуванні. Відео на 2 хвилини та 10 секунд було захешовано у1451 хеш, хоча всього фреймів було більше 4 тисяч. Зайняло часу близько 45 секунд. Операції по створенню блум фільтра та сцен були швидкими – 9 мілісекунд для обох випадків.

```
: Skipped frame №4258
: Skipped frame №4259
: Hash №1451
: Skipped frame №4261
: Skipped frame №4262
: Skipped frame №4263
: Hashing video took 46658ms
: Starting to creating Blooms Filter. Hashes size: 1451
: Creating Blooms filter took 9ms
: Starting to creating Scenes
: Creating Scenes took 9 ms, number of scenes: 146
```

Рис. 3.15. Вивід в консоль при хешуванні відео

А при пошуку відео, логи були наступними:

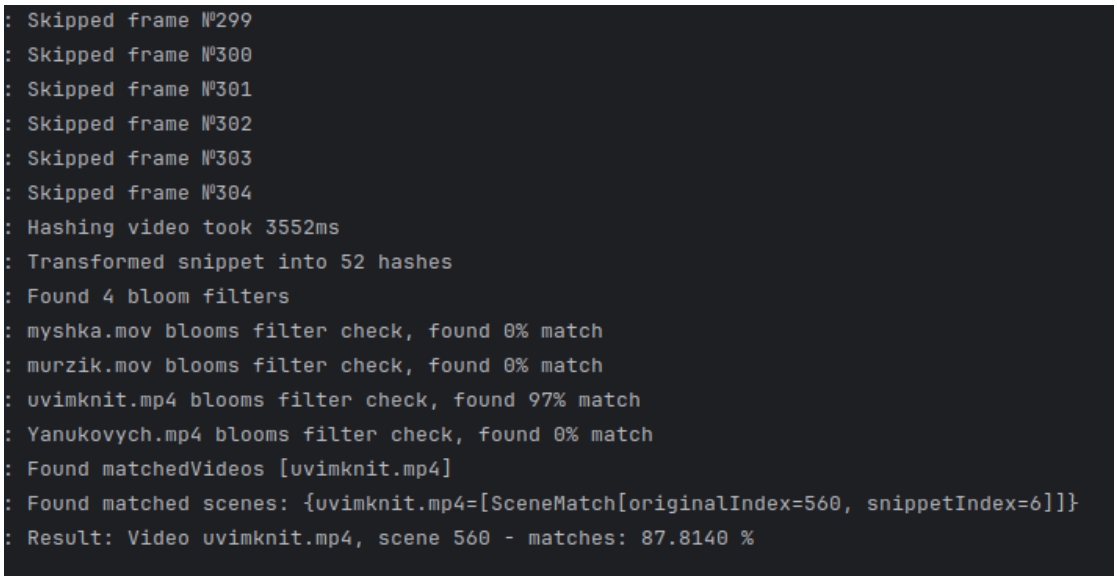

Рис. 3.16. Вивід в консоль при операції пошуку

Так як відеофрагмент був на 10 секунд, він трансформувався у 52 хеша, і це зайняло близько трьох секунд часу. При перевірці блум фільтрів лише один з чотирьох пройшов далі, знайдено відповідну сцену та визначено схожіть у відсотковому співвідношені.

ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі було визначено ключові аспекти, пов’язані з алгоритмами хешування та ідентифікації в контексті систем відеоідентифікації. Визначення та вибір ефективного алгоритму хешування є фундаментальним для забезпечення точності та надійності ідентифікації відеоконтенту.

Далі, ми детально розглянули перший A-хеш, його особливості та характеристики. Цей алгоритм є одним з основоположних у сфері відеоідентифікації, і його аналіз дозволив нам глибше зрозуміти принципи роботи хешування відеоданих.

Особлива увага приділена питанням зберігання хешів та вибору алгоритму пошуку, зокрема використанню Фільтра Блума. Це допомагає оптимізувати процеси зберігання та витягування даних, забезпечуючи швидкий та ефективний доступ до відеоінформації.

Ми також зосередили увагу на визначенні моменту початку порівняння, що є ключовим для точного виявлення схожих сцен та аналізу відеоконтенту. Кореляційні методи, включаючи Відстань Гемінга, розглядались як інструменти для визначення схожості хешів, підкреслюючи їхню важливість у процесі ідентифікації відео.

Завершальна частина розділу присвячена перевірці схожості хешів та розгляду алгоритмічної складності цих процесів. Це дозволяє краще зрозуміти виклики, пов'язані з обробкою великих обсягів відеоданих, та важливість розробки оптимізованих та ефективних рішень.

ВИСНОВКИ

В ході роботи було досліджено системи розпізнавання відео з використанням імовірнісних структур даних. Вивчення історичної еволюції відеотехнологій показало їх глибокий вплив на медіа та суспільство, зокрема, як технологічний прогрес змінив способи створення, розповсюдження та споживання відеоконтенту. В роботі були розглянуті не тільки сучасні системи відеоідентифікації, але й їх історичні попередники, що дало змогу оцінити еволюцію технологій у цій галузі. Окрему увагу було приділено аналізу технік створення відбитків пальців відео, в тому числі розробці ефективних методів хешування, які спрощують процес ідентифікації відео.

Критичний аналіз алгоритмів хешування, включаючи A-Hash, D-Hash, P-Hash, W-Hash та SVD-Hash, виявив їх важливість та різноманітність застосувань у сфері відеоідентифікації. Робота також зосередилась на значенні водяних знаків у захисті авторських прав та їх ролі у підвищенні стійкості відео до несанкціонованого використання. Подальше дослідження звернуло увагу на способи інтеграції цих алгоритмів у різноманітні платформи та сервіси, підкреслюючи їхню адаптивність та ефективність у різних середовищах обробки відео. Особливий акцент було зроблено на забезпеченні балансу між швидкістю обробки та точністю ідентифікації, що є критично важливим у швидкозмінному медіа-пейзажі.

Також було висвітлено виклики, пов'язані з обробкою відео високої роздільної здатності та забезпеченням надійності системи в умовах постійно зростаючих обсягів відеоданих. Особливу увагу було приділено аналізу стійкості вищезгаданих алгоритмів до змін у якості відео, що є важливим аспектом у сучасних умовах широкого спектра відеоформатів. Завершальною частиною роботи стала оцінка можливостей їхнього застосування у різних сценаріях, від аналітики в медіа до систем безпеки, що демонструє гнучкість та широкі перспективи використання цих технологій.

В роботі також досліджується зберігання та аналіз хешів, висвітлюється використання баз даних, таких як Apache Cassandra, та імовірнісних структур даних, таких як фільтри Блума. Особливу увагу було приділено можливостям Apache Cassandra в контексті обробки та зберігання великих обсягів відеоданих, включаючи її високу продуктивність та масштабованість, які є критично важливими для систем відеоідентифікації. Фільтри Блума використовувалися для ефективного виявлення та фільтрації хешів, що дозволило знизити обсяг даних для пошуку та підвищити швидкість доступу до них. Важливість визначення необхідного обсягу даних для зберігання була обгрунтована аналізом потреб системи та вимог до ефективності.

Кореляційні методи, включаючи відстань Гемінга, були використані для точного визначення схожості між різними хешами. Дослідження охоплювало детальний аналіз цих методів, їх ефективності та придатності у різних сценаріях застосування. Важливість цього аспекту полягала у забезпеченні високої точності ідентифікації, уникаючи помилкових співставлень та забезпечуючи надійність системи. Дослідження також дає уявлення про алгоритмічну складність, пов'язану з цими процесами, і виявляє важливість балансу між швидкістю та точністю при обробці відеоданих.

В результаті цього дослідження був розроблений робочий прототип системи розпізнавання відео за коротким фрагментом. Система, інтегруючи всі згадані властивості та компоненти, продемонструвала високу швидкість обробки та ефективність у розпізнаванні. Використання сучасних технологій баз даних та імовірнісних структур дозволило досягти масштабування системи без втрати її продуктивності. Також було звернуто увагу на гнучкість системи, її здатність адаптуватися до різних форматів відео та забезпечення високого рівня безпеки даних. Це створило перспективну основу для її подальшого використання в різних галузях, від медіа-аналітики до систем безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Baris, C., Bulent, S., and Nasir, M. Spatio-temporal transform based video hashing, IEEE Trans. , U.K: Packt Pub., 2018.

2. Hampapur, A., and Bolle, R. M. “Comparison of distance measures for video copy detection,” in International conference on multimedia and expo, 2010.

3. Apache Cassandra - Architecture - Overview [Електронний ресурс] – Режим доступу: https://cassandra.apache.org/doc/latest/cassandra/architecture/overview.html (дата звернення: 21.09.2023). – Назва з екрану.

4. Overview of Perceptual Hashing Technology — Ofcom [Електронний ресурс] – Режим доступу: https://www.ofcom.org.uk/__data/assets/pdf_file/0036/247977/Perceptual-hashing-technology.pdf (дата звернення: 21.09.2023). – Назва з екрану.

5. Analysis of Perceptual Hashing Algorithms in Image Manipulation Detection[Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/science/article/pii/S1877050921011030 (дата звернення: 12.10.2023). – Назва з екрану.

6. Multimedia Image and Video Processing: Anaya Publishers, 2003. 336 с.

7. Probabilistic Data Structures and Algorithms for Big Data Applications / Books on Demand— Books on Demand, Inc., 2022. — 123с. .

8. Computer Graphics for Java Programmers / Leen Ammeraal, Kang Zhang — Springer International Publishing, Inc., 2017. — 320с. .

9. Spring 4 для професіоналів/ Кріс Шефер, Кларенс Хо, Роб Харроп — Springer International Publishing, Inc., 2017. — 240с.

10. An overview of perceptual hashing - Farid, Hany. Journal of Online Trust and Safety 2021 — 15с.

11. Video fingerprinting for copy identification: from research to industry applications / Lu, Jian.. / Media Forensics and Security 7254. – 2019.

12. Introduction to video Fingerprinting / Chao, Wei-Lun. / Graduate Institute of Communication of Engineering, NTU — 2020.

13. Theory and practice of bloom filters for distributed systems. / Tarkoma, Sasu, Christian Esteve Rothenberg, Eemil Lagerspetz / IEEE Communications Surveys & Tutorials 14.1: 131-155 — 2017.

14. Camera Fingerprint Extraction via Spatial Domain Averaged Frames. IEEE Xplore. [Електронний ресурс] https://ieeexplore.ieee.org/abstract/document/9056806 (дата звернення: 10.12.2023) — Назва з екрану.

15. A Hash-Based Fast Image Encryption Algorithm. / Han, Ruifeng. / Wireless Communications and Mobile Computing, 2022 .

16. NoSQL Databases / Li Ziqi / 2019.

17. Imaging framework: An interoperable and extendable connector for image-related Java frameworks [Електронний ресурс] – Режим доступу: https://www.sciencedirect.com/science/article/pii/S2352711021001357 (дата звернення: 15.09.2023). – Назва з екрану.

18. Video streaming application with post-decoding concealment technique [Електронний ресурс] – Режим доступу: https://iopscience.iop.org/article/10.1088/1742-6596/1783/1/012068/meta (дата звернення: 25.10.2023). – Назва з екрану.

19. Experiments with the Shazam music identification algorithm / XIAO, Fangjian Flora / 2018.

ДОДАТОК А

«Хешування відео»

```
package org.bs.com.diplomvideohash;

import com.google.common.collect.Lists;
import org.bytedeco.javacv.FFmpegFrameGrabber;
import org.bytedeco.javacv.FFmpegLogCallback;
import org.bytedeco.javacv.Frame;
import org.bytedeco.javacv.Java2DFrameConverter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.List;
import java.util.*;

@Component
public class VideoHasherService {

    public static final Logger log =
LoggerFactory.getLogger(VideoHasherService.class);
    public static final int BATCH_SIZE = 10;
    public static final int HASH_MATRIX_SIZE = 8;
    public static final int THRESHOLD = 3;
    public static final String PATH = "src/main/resources/videos/";
    private final Java2DFrameConverter converter;
    private final HashedVideoService videoHashService;
    private final BloomFilterService bloomFilterService;
```

```
    public VideoHasherService(Java2DFrameConverter converter,
HashedVideoService videoHashService, BloomFilterService
bloomFilterService) {
        this.converter = converter;
        this.videoHashService = videoHashService;
        this.bloomFilterService = bloomFilterService;
    }

    public void hashVideo(String filepath) {
        List<VideoHash> hashes = startHashingVideo(filepath);
        BloomFilter bloomFilter = startCreatingBloomFilter(hashes);
        List<int[]> scenes = startScenesCompressing(hashes);

        videoHashService.saveVideo(filepath, hashes, scenes);
        bloomFilterService.saveBloomFilter(filepath, bloomFilter);
    }

    private List<int[]> startScenesCompressing(List<VideoHash> hashes) {
        long scenesStart = System.currentTimeMillis();
        log.info("Starting to creating Scenes");
        List<int[]> scenes = getScenesPicks(hashes);
        long scenesEnd = System.currentTimeMillis();
        log.info("Creating Scenes took {} ms, number of scenes: {}",
scenesEnd - scenesStart, scenes.size());
        return scenes;
    }

    private BloomFilter startCreatingBloomFilter(List<VideoHash> hashes) {
        long bloomStart = System.currentTimeMillis();
        log.info("Starting to creating Blooms Filter. Hashes size: {}",
hashes.size());
        BloomFilter bloomFilter =
bloomFilterService.getBloomFilterFromBits(hashes.stream()
                .map(VideoHash::hash)
```

```
                .toList());
        long bloomEnd = System.currentTimeMillis();
        log.info("Creating Blooms filter took " + (bloomEnd - bloomStart) +
"ms");
        return bloomFilter;
    }

    public List<VideoHash> startHashingVideo(String filepath) {
        long start = System.currentTimeMillis();
        List<VideoHash> hashes = getVideoHashes(filepath);
        long end = System.currentTimeMillis();
        log.info("Hashing video took " + (end - start) + "ms");
        return hashes;
    }

    public List<VideoHash> getVideoHashes(String filepath) {
        List<VideoHash> hashes = new LinkedList<>();
        FFmpegLogCallback.set();

        try (FFmpegFrameGrabber grabber = new FFmpegFrameGrabber(PATH
+ filepath)) {
            grabber.start();

            Frame frame;
            int frameNumber = 0;
            long duration = grabber.getLengthInTime() / 1_000_000;
            log.info("Starting to hashing video {}, duration: {} seconds",
filepath, duration);
            while ((frame = grabber.grabImage()) != null) {
                try {
                    frameNumber++;
                    BufferedImage singleFrame = converter.convert(frame);
                    BufferedImage fixedFrame =
resizeAndGrayscaleFrame(singleFrame);
```

```
                double averagePixelValue = getAveragePixelValue(fixedFrame);
                BitSet hash = getBitsStr(fixedFrame, averagePixelValue,
hashes.size() + 1);

                if (isEqualToLast(hashes, hash)) {
                    log.info("Skipped frame №" + frameNumber);
                    continue; // skip
                }

                hashes.add(new VideoHash(hash, hashes.size() + 1));
                log.info("Hash №" + hashes.size());
            } catch (Exception e) {
                log.error("Exception during hashing " + frameNumber + "
frame.", e);
            }
        }
    } catch (Exception e) {
        log.error("Hashing video error", e);
    }
    return hashes;
}

public List<int[]> getScenesPicks(List<VideoHash> hashes) {
    List<List<VideoHash>> partitionedHashes = Lists.partition(hashes,
BATCH_SIZE);
    List<int[]> scenes = new ArrayList<>();

    for (List<VideoHash> partHashes : partitionedHashes) {
        int[] spectr = new int[HASH_MATRIX_SIZE * HASH_MATRIX_SIZE];
        for (VideoHash videoHash : partHashes) {
            for (int i = 0; i < HASH_MATRIX_SIZE * HASH_MATRIX_SIZE; i++) {
                spectr[i] += videoHash.hash().get(i) ? 1 : 0;
            }
        }
```

```
            scenes.add(spectr);
        }

        return foundPicks(scenes);
    }

    private static List<int[]> foundPicks(List<int[]> scenes) {
        List<int[]> hashedScenes = new ArrayList<>();

        for (int[] scene : scenes) {
            int[] result = new int[scene.length];
            int[] sortedArr = scene.clone();
            Arrays.sort(sortedArr);

            // Threshold is the smallest of the top N values
            int threshold = sortedArr[sortedArr.length - THRESHOLD];

            for (int i = 0; i < scene.length; i++) {
                // Keep only the values that are greater or equal to the threshold
                result[i] = scene[i] >= threshold ? scene[i] : 0;
            }

            hashedScenes.add(result);
        }
        return hashedScenes;
    }

    private boolean isEqualToLast(List<VideoHash> hashes, BitSet newHash)
{
        if (hashes.isEmpty()) {
            return false;
        }
```

```
        BitSet oldHash = hashes.get(hashes.size() - 1).hash();
        return oldHash.toString().equals(newHash.toString());
    }

    private BitSet getBitsStr(BufferedImage grayscaleImage, double
averagePixelValue, int i) {
        BitSet hash = new BitSet(grayscaleImage.getHeight() *
grayscaleImage.getWidth());
        int counter = 0;
        for (int y = 0; y < grayscaleImage.getHeight(); y++) {
            for (int x = 0; x < grayscaleImage.getWidth(); x++) {
                int pixelValue = grayscaleImage.getRaster().getSample(x, y, 0);
                hash.set(counter++, pixelValue > averagePixelValue);
                if (pixelValue > averagePixelValue) {
                    grayscaleImage.setRGB(x, y, -16777216);
                    hash.set(counter++, true);
                } else {
                    grayscaleImage.setRGB(x, y, -1);
                    hash.set(counter++, false);
                }
            }
        }

        return hash;
    }

    private double getAveragePixelValue(BufferedImage grayscaleImage) {
        double sum = 0;
        int height = grayscaleImage.getHeight();
        int width = grayscaleImage.getWidth();

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int pixelValue = grayscaleImage.getRaster().getSample(x, y, 0);
```

```
                sum += pixelValue;
            }
        }

        return sum / (width * height);
    }

    private BufferedImage resizeAndGrayscaleFrame(BufferedImage
originalImage) {
        BufferedImage grayscaleImage = new
BufferedImage(HASH_MATRIX_SIZE, HASH_MATRIX_SIZE,
BufferedImage.TYPE_BYTE_GRAY);

        Graphics2D g2d = grayscaleImage.createGraphics();
        g2d.drawImage(originalImage.getScaledInstance(HASH_MATRIX_SIZE,
HASH_MATRIX_SIZE, Image.SCALE_SMOOTH), 0, 0, null);
        g2d.dispose();

        return grayscaleImage;
    }

    public record VideoHash(BitSet hash, long frameNumber) {
    }
}
```

ДОДАТОК Б

«Пошук відео по фрагменту»

```
package org.bs.com.diplomvideohash;

import com.google.common.collect.Lists;
import org.bs.com.diplomvideohash.dao.bloomfilter.BloomFilterEntity;
import org.bs.com.diplomvideohash.dao.video.HashedVideoEntity;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import java.math.BigDecimal;
import java.util.*;
import java.util.function.BiFunction;
import java.util.function.Function;
import java.util.stream.Collectors;

@Component
public class MatcherService {

    public static final Logger log =
LoggerFactory.getLogger(MatcherService.class);

    private final static Integer MIN_MATCH = 90; // %
    public static final String IS_MATCHES_TMP = "Video %s, scene %s -
matches: %s %%";
    private final BloomFilterService bloomFilterService;
    private final HashedVideoService hashedVideoService;
    private final VideoHasherService videoHasherService;

    public MatcherService(BloomFilterService bloomFilterService,
HashedVideoService hashedVideoService, VideoHasherService
videoHasherService) {
```

```
        this.bloomFilterService = bloomFilterService;
        this.hashedVideoService = hashedVideoService;
        this.videoHasherService = videoHasherService;
    }

    public String findMatches(String snippetName) {
        log.info("Starting finding matches for {} snippet", snippetName);
        List<VideoHasherService.VideoHash> snippetHashes =
videoHasherService.startHashingVideo(snippetName);
        log.info("Transformed snippet into {} hashes",
snippetHashes.size());

        List<BloomFilterEntity> bloomFilters =
bloomFilterService.getAllBloomFilters();
        log.info("Found {} bloom filters", bloomFilters.size());

        List<String> matchedVideos = foundMatchedVideos(bloomFilters,
snippetHashes);
        log.info("Found matchedVideos {}", matchedVideos);

        Map<String, HashedVideoEntity> hashedVideos =
hashedVideoService.findAll(matchedVideos).stream()
                .collect(Collectors.toMap(HashedVideoEntity::getVideoId,
Function.identity()));


        Map<String, List<SceneMatch>> startPointsPerVideo =
fillWithScenes(List.copyOf(hashedVideos.values()), snippetHashes);
        Map<VideoMatch, BigDecimal> matches = new HashMap<>();

        for (Map.Entry<String, List<SceneMatch>> entry :
startPointsPerVideo.entrySet()) {
            String video = entry.getKey();
            List<SceneMatch> frames = entry.getValue().stream()
```

```
                    .sorted((a, b) -> Integer.compare(b.originalIndex(),
a.originalIndex()))
                    .toList();

            for (SceneMatch startPoint : frames) {
                ArrayList<BigDecimal> percentage = new ArrayList<>();
                for (int i = 0; i < 10; i++) {
                    percentage.add(getMatchPercent(hashedVideos, video,
snippetHashes,
                            startPoint.originalIndex() + i,
startPoint.snippetIndex() + i));
                }
                BigDecimal full =
percentage.stream().reduce(BigDecimal.ZERO, BigDecimal::add);
                BigDecimal avarage =
full.divide(BigDecimal.valueOf(percentage.size()));
                matches.put(new VideoMatch(video,
startPoint.originalIndex()), avarage.add(BigDecimal.valueOf(50)));
            }
        }
        StringBuilder builder = new StringBuilder();
        for (Map.Entry<VideoMatch, BigDecimal> entry :
matches.entrySet()) {

builder.append(IS_MATCHES_TMP.formatted(entry.getKey().videoId(),
entry.getKey().scene(), entry.getValue()));
            builder.append("\n");
        }

        log.info("Result: {}", builder);
        return builder.toString();
    }
```

```
        private static BigDecimal getMatchPercent(Map<String,
HashedVideoEntity> hashedVideos, String video,
List<VideoHasherService.VideoHash> snippetHashes, long frameIndex,
Integer snippetIndex) {
            HashedVideoEntity hashedVideoEntity =
hashedVideos.get(video);
            Long hash = hashedVideoEntity.getHashes().get(frameIndex);

            BitSet bitSet = BitSet.valueOf(new long[]{hash});

            BitSet snippetHash = snippetHashes.get(snippetIndex).hash();
            BitSet copy = BitSet.valueOf(bitSet.toByteArray());

            copy.xor(snippetHash);

            int cardinality = copy.cardinality();
            return NumberUtils.toPercentageOf(64 - cardinality, 64);
        }

        private Map<String, List<SceneMatch>>
fillWithScenes(List<HashedVideoEntity> videoEntities,
List<VideoHasherService.VideoHash> videoHashes) {
            Map<String, List<SceneMatch>> startPointsPerVideo = new
HashMap<>();

            for (HashedVideoEntity videoEntity : videoEntities) {
                List<String> scenes = new
ArrayList<>(videoEntity.getScenes().stream().map(Arrays::toString).toList());

                for (int i = 0; i < 10; i++) {
                    List<VideoHasherService.VideoHash> hashes =
videoHashes.subList(i, i + 10);
                    List<int[]> scenesPicks =
videoHasherService.getScenesPicks(hashes);
```

```
                    int[] hash = scenesPicks.get(0);
                    String stringHash = Arrays.toString(hash);

                    if (scenes.contains(stringHash)) {
                        int index = scenes.indexOf(stringHash);
                        startPointsPerVideo.compute(videoEntity.getVideoId(),
addOrCreate((index + 1) * 10, i));
                        scenes.remove(index);
                    }
                }
            }
            log.info("Found matched scenes: {}", startPointsPerVideo);
            return startPointsPerVideo;
        }

        private static BiFunction<String, List<SceneMatch>,
List<SceneMatch>> addOrCreate(int index, int i) {
            return (k, v) -> {
                SceneMatch sceneMatch = new SceneMatch(index, i);
                if (v == null) {
                    return Lists.newArrayList(sceneMatch);
                } else {
                    v.add(sceneMatch);
                    return v;
                }
            };
        }

        private List<String> foundMatchedVideos(List<BloomFilterEntity>
bloomFilters, List<VideoHasherService.VideoHash> videoHashes) {
            List<String> prematchedVideos = new ArrayList<>();
            for (BloomFilterEntity bloomFilterEntity : bloomFilters) {
                BitSet bitSet = bloomFilterEntity.getBloom_filter();
```

```
                BloomFilter bloomFilter = BloomFilter.parseBloomFilter(bitSet,
bloomFilterEntity.getSize());

                long numberOfMatched = videoHashes.stream()
                        .map(VideoHasherService.VideoHash::hash)
                        .filter(bits -> bloomFilter.mightContain(bits.toString()))
                        .count();

                BigDecimal value = BigDecimal.valueOf(numberOfMatched);
                BigDecimal total = BigDecimal.valueOf(videoHashes.size());

                int matchPercentage = NumberUtils.toPercentageOf(value,
total).intValue();
                if (numberOfMatched > 0) {
                   prematchedVideos.add(bloomFilterEntity.getVideoId());
                }
                log.info("{} blooms filter check, found {}% match",
bloomFilterEntity.getVideoId(),
                        (matchPercentage > 0 ? 100 - matchPercentage : 0));
            }
            return prematchedVideos;
        }

        public record VideoMatch(String videoId, Integer scene) {
        }

        public record SceneMatch(int originalIndex, int snippetIndex) {
        }
    }
```