

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА
ФРАНКА

Факультет прикладної математики та інформатики

Кафедра прикладної математики

Дипломна робота

АНАЛІЗ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ З РІЗНИМИ ГРАФАМИ
РОЗРАХУНКІВ

Виконала: студентка групи ПМП-42
спеціальності
113 - прикладна математика

Скорочкіна М. М.

(прізвище та ініціали)

Керівник Марчук Ю. Б.

(прізвище та ініціали)

Рецензент _____

(прізвище та ініціали)

Львів - 2023

ЗМІСТ

ВСТУП	3
РОЗДІЛ 1. ОГЛЯД РІЗНИХ МЕТОДІВ АНАЛІЗУ ЗОБРАЖЕНЬ ТА ЇХ ЗАСТОСУВАННЯ В СИСТЕМАХ КОМП'ЮТЕРНОГО ЗОРУ	6
1.1 Класифікація систем комп'ютерного зору	6
1.2 Методи аналізу зображень в системах комп'ютерного зору.....	7
1.3 Штучні нейронні мережі (ШНМ)	9
РОЗДІЛ 2. МАТЕМАТИЧНІ МЕТОДИ АНАЛІЗУ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ	13
2.1 Математичні основи систем комп'ютерного зору	13
2.2 Теорія ймовірностей та статистика у системах комп'ютерного зору	15
2.3 Геометричні методи аналізу зображень.....	18
РОЗДІЛ 3. РОЗГЛЯД ПРИКЛАДІВ РЕАЛІЗАЦІЇ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ З РІЗНИМИ ГРАФАМИ РОЗРАХУНКІВ.....	22
3.1 Бібліотека TensorFlow.....	22
3.2 Бібліотека Tensor Flow 1.0 та її особливості	24
3.3 Бібліотека Tensor Flow 2.0 та її особливості	25
3.4 Порівняння реалізації бібліотек Tensor Flow 1.0 і Tensor Flow 2.0	28
ВИСНОВОК.....	32
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	34
ДОДАТКИ.....	36

ВСТУП

Об'єкт дослідження: Швидкодія бібліотеки TensorFlow в різних версіях.

Предмет дослідження: Аналіз систем комп'ютерного зору з використанням різних графів розрахунків та порівняння швидкодії бібліотеки TensorFlow двох версій.

Мета досліджень: Оцінити вплив версій бібліотеки TensorFlow на швидкодію систем комп'ютерного зору, яка використовує різні графи розрахунків.

Системи комп'ютерного зору (Computer Vision) - це системи, які використовують технології обробки зображень та аналізу даних для автоматичного виявлення, розпізнавання та інтерпретації образів. Їх метою є забезпечення автоматизації процесів обробки великих об'ємів даних, що містять зображення. [1]

Актуальність теми "Аналіз систем комп'ютерного зору з різними графами розрахунків" базується на розширенні використання комп'ютерного зору в різних сферах життя. Системи комп'ютерного зору здатні виявляти об'єкти, розпізнавати образи, аналізувати великі обсяги даних і приймати рішення на основі зібраної інформації.

Стан проблеми полягає в пошуку оптимальних методів та алгоритмів, які б дозволяли ефективно використовувати графи розрахунків у системах комп'ютерного зору. Це включає вивчення різних типів графів, їхнього впливу на продуктивність обчислень та розробку ефективних алгоритмів обробки даних.

Ступінь дослідженості обраного напрямку в українському науковому просторі свідчить про активну роботу вчених та дослідників в цій області. Українські наукові групи та лабораторії займаються розробкою та вдосконаленням методів комп'ютерного зору з використанням різних графів

розрахунків. Вони працюють над створенням нових алгоритмів, моделей та інструментів для поліпшення продуктивності та точності систем комп'ютерного зору. Відомі в Україні такі наукові установи як Інститут кібернетики ім. В.М. Глушкова НАН України, Інститут проблем штучного інтелекту НАН України, Національний технічний університет України "Київський політехнічний інститут" та інші, які займаються дослідженнями у сфері комп'ютерного зору. [2]

Однак, варто зазначити, що ця тема також має активне дослідження за межами України. Багато світових університетів, дослідницьких центрів та компаній займаються вивченням та розробкою систем комп'ютерного зору з використанням різних графів розрахунків. Прикладами є Массачусетський технологічний інститут (MIT), Стенфордський університет, Каліфорнійський університет у Берклі, Google Research, Facebook AI Research та інші великі наукові установи. Вони внесли значний вклад у розвиток систем комп'ютерного зору та дослідження графів розрахунків. [3]

Протиріччя можуть існувати в різних доробках вчених та дослідників щодо вибору оптимальних графів розрахунків для систем комп'ютерного зору. Вони можуть породжувати дискусії щодо ефективності та придатності певних графових розрахунків у системах комп'ютерного зору. Крім того, можуть існувати розбіжності щодо вибору конкретних бібліотек та фреймворків для реалізації систем комп'ютерного зору з використанням графів розрахунків. Одні дослідники можуть надавати перевагу TensorFlow і вважати його найбільш відповідним рішенням, тоді як інші можуть висловлювати зауваження до його продуктивності та шукати альтернативні бібліотеки або власні розробки. У нашому випадку, аналізується швидкодія бібліотеки TensorFlow у двох версіях. Це дозволить встановити, яка версія бібліотеки забезпечує кращу продуктивність та ефективність системи комп'ютерного зору з різними графами розрахунків. У цьому контексті дослідження з аналізу систем комп'ютерного зору з різними графами розрахунків дозволять оцінити

ефективність та переваги різних версій TensorFlow і внести свій внесок в розуміння проблеми та її вирішення. [1]

РОЗДІЛ 1. ОГЛЯД РІЗНИХ МЕТОДІВ АНАЛІЗУ ЗОБРАЖЕНЬ ТА ЇХ ЗАСТОСУВАННЯ В СИСТЕМАХ КОМП'ЮТЕРНОГО ЗОРУ

1.1 Класифікація систем комп'ютерного зору

Комп'ютерний зір - це галузь науки, що займається розробкою алгоритмів та технологій, які дозволяють машинам виявляти, відстежувати та класифікувати об'єкти на зображеннях або відео. Головна мета комп'ютерного зору полягає в наданні комп'ютерам здатності "бачити" і розуміти світ через використання візуальної інформації. Він складається з різноманітних завдань, таких як: виявлення об'єктів, класифікація, сегментація, розпізнавання облич, вимірювання розмірів та форм об'єктів, визначення руху, аналіз тексту, ідентифікація, трекінг об'єктів та багато інших. Для вирішення цих завдань використовуються різні методи та алгоритми, включаючи статистичний аналіз, машинне навчання, нейронні мережі, глибинне навчання та інші. [1]

Комп'ютерний зір, як наукова дисципліна, спрямований на вивчення теорії та розробку технологій для створення штучних систем, які опрацьовують інформацію у вигляді зображень. Відеодані можуть бути подані у різних формах, таких як відеопослідовності, зображення з різних камер або тривимірні дані з медичних сканерів. [4]

Як технологічна дисципліна, він прагне використовувати свої теорії та моделі для розробки систем комп'ютерного зору. На основі вивчення відповідної літератури були виділені наступні основні класи систем комп'ютерного зору:

1. Системи керування процесами, що включають промислові роботи та автономні транспортні засоби.
2. Системи відеоспостереження.
3. Системи організації інформації, які можуть використовуватися для індексації баз даних зображень.

4. Системи моделювання об'єктів або оточуючого середовища, такі як аналіз медичних зображень або топографічне моделювання.

5. Системи взаємодії, які включають пристрої введення для людино-машинної взаємодії. [4]

Ба більше, можна сказати, що комп'ютерний зір взаємодіє з біологічним зором і може бути розглянутий, до прикладу, як його доповнення. У біологічній науці вивчається сприйняття зором людини та різних тварин, і на основі цього створюються моделі роботи таких зорових систем, які пояснюють фізіологічні процеси. З іншого боку, комп'ютерний зір досліджує та описує системи комп'ютерного зору, які реалізовані апаратно або програмно. Взаємодія між біологічним та комп'ютерним зором виявилась досить продуктивною для обох наукових галузей, і цей міждисциплінарний обмін сприяє розвитку обох дисциплін.

1.2 Методи аналізу зображень в системах комп'ютерного зору

Аналіз зображень є ключовим компонентом систем комп'ютерного зору, що здатні автоматично розуміти та інтерпретувати зображення. Розглянемо основні методи обробки зображень, сегментації та розпізнавання об'єктів, які використовуються в системах комп'ютерного зору.

Методи обробки зображень включають різні техніки, які використовуються для поліпшення, зміни або підготовки зображень перед подальшим аналізом. Деякі з найпоширеніших методів обробки зображень включають фільтри, згладжування і контрастування.

- Фільтри використовуються для покращення якості зображення шляхом видалення шуму або підсилення певних характеристик. Наприклад, фільтр розмивання (blur filter) видаляє високочастотні деталі, тоді як фільтр різкості (sharpen filter) підсилює різкість контурів об'єктів.

- Згладжування (smoothing) використовується для зменшення шуму або видалення непотрібних деталей на зображенні. Це досягається шляхом

розмиття значень пікселів в околі кожного пікселя. Згладжування може бути корисним, коли потрібно знизити шум для подальшого аналізу зображень.

- Контрастування (contrast enhancement) використовується для підвищення різкості і виразності зображення шляхом розширення діапазону яскравості. Це може бути досягнуто шляхом підсилення різниці між яскравими і темними областями зображення, що поліпшує його візуальну якість та сприяє подальшому аналізу. [5]

Методи сегментації зображень використовуються для виділення та визначення окремих об'єктів або регіонів на зображенні. Основні методи сегментації включають бінаризацію, порогову обробку та сегментацію за кольором.

- Бінаризація (binarization) використовується для перетворення зображення в двокольорове, де кожен піксель класифікується як об'єкт або фон. Зазвичай, для бінаризації використовується порогове значення, яке розділяє пікселі на дві категорії в залежності від їх яскравості або колірної характеристики.

- Порогова обробка (thresholding) використовується для розділення пікселів на дві категорії на основі порогового значення. Пікселі зі значеннями, які перевищують поріг, вважаються об'єктами, тоді як пікселі з меншими значеннями вважаються фоном. Цей метод є простим і ефективним для сегментації зображень з чіткою різницею між об'єктами та фоном.

- Сегментація за кольором (color segmentation) використовується для виділення об'єктів на зображенні на основі їх кольору або колірних характеристик. Цей метод дозволяє виявити об'єкти з різними кольорами або відокремити їх від фону, що особливо корисно для аналізу зображень з багатьма об'єктами різних кольорів. [5]

Методи розпізнавання об'єктів на зображеннях використовуються для автоматичного визначення і класифікації об'єктів на основі їх характеристик і

особливостей зображення. Деякі з найпоширеніших методів розпізнавання об'єктів включають класифікацію, нейромережі і глибинне навчання.

- Класифікація використовується для призначення об'єктів на зображенні до певних категорій або класів на основі їх характеристик або навчених моделей. Цей метод вимагає побудови набору ознак, які описують об'єкти, і розробки алгоритму для призначення класів на основі цих ознак. Класифікація може використовувати різні алгоритми, такі як метод опорних векторів (Support Vector Machines, SVM) або наївний байєсівський класифікатор.

- Нейромережі використовуються для розпізнавання об'єктів шляхом моделювання мозкових нейронних мереж. Цей підхід базується на використанні штучних нейронних мереж, які навчаються розпізнавати певні образи або характеристики. Нейромережі можуть бути навчені на великих наборах зображень для розпізнавання об'єктів з високою точністю.

- Глибинне навчання (deep learning) є підмножиною машинного навчання, яка використовує нейромережі з багатьма шаровими архітектурами. Цей метод дозволяє автоматично вивчити характеристики і представлення зображень на різних рівнях абстракції. Глибинне навчання здатне досягати вражаючих результатів у розпізнаванні об'єктів на зображеннях, зокрема завдяки використанню згорткових нейронних мереж. [5]

Ці методи розпізнавання об'єктів на зображеннях використовуються в різних застосуваннях, таких як комп'ютерний зір, медична діагностика, автоматичне розпізнавання образів, безпека та багато інших областей, де важлива автоматизація і аналіз великих обсягів візуальних даних.

1.3 Штучні нейронні мережі (ШНМ)

Штучні нейронні мережі - це обчислювальні системи, створені на основі біологічних нейронних мереж у мозку тварин. Вони складаються з штучних нейронів, які взаємопов'язані між собою. Штучні нейрони здатні вчитися на

прикладів і адаптувати свої ваги, щоб здійснювати бажані обчислення та розв'язувати задачі. В результаті цього штучні нейронні мережі можуть застосовуватись для розпізнавання зображень, класифікації даних, прогнозування та вирішення різних завдань у сфері штучного інтелекту. [7]

Нейрон є складовою частиною програмного коду, що утворює нейронну мережу. Кожен нейрон отримує вхідну інформацію, обробляє її та передає до інших нейронів за допомогою синапсів. Якщо узагальнювати, то можна сказати, що нейрон є базовою та основною одиницею штучного інтелекту, а нейронна мережа - це комп'ютерна реалізація мозку. [6]

Розвиток всесвітньої мережі та процеси глобалізації призвели до того, що з'явилась величезна кількість інформації, яку людина фізично не здатна обробити без допомоги комп'ютера. Нейронні мережі використовуються для наступних цілей:

- аналізу та класифікації даних згідно з певними критеріями.
- створення аналітичних прогнозів, заснованих на вхідних даних.
- порівняння та розпізнавання однакових даних.

Існує кілька типів нейронних зв'язків, серед яких найбільш поширеними є синусоїдальний та ReLU. В першому випадку нейронна мережа використовує дані в діапазоні від -1 до 1 (що фактично відповідає -100% до 100%). В другому вхідні дані передаються через значення 0 та \inf (інформація будь-якого характеру). [6]

Для пояснення системного аналізу краще підходить синусоїдальна функція, оскільки обмежений діапазон вхідної інформації дозволяє отримати більш наочне уявлення. Алгоритм обчислення:

- дані надходять на нейрон;
- вираховується їх вага;
- результати обчислень передаються до наступного нейрону;
- алгоритм повторюється. [6]

Кількість обчислень задається шляхом встановлення кількості шарів. Сучасні нейронні мережі мають десятки, а іноді навіть сотні шарів обчислення.

Перед початком машинного навчання вхідні дані для нейронної мережі потрібно привести до встановленого формату. Наприклад, якщо ми аналізуємо динаміку ринку акцій, ціни можуть мати великі значення. Тому ми можемо нормалізувати дані, перетворивши їх на відсоткові зміни цін, що дозволить отримати значення в діапазоні від -1 до 1.

Цей процес нормалізації вхідних даних є першим і ключовим кроком перед початком машинного навчання. Система повинна отримувати дані у форматі, який вона може обробити.

Наступним кроком є отримання першого результату обчислень. Часто він може відрізнятись від очікуваного результату, оскільки нейронна мережа не має достатньо інформації для точного аналізу, а значить, ймовірно, не має вагомості розподілення.

На цьому етапі створюється тренувальний сет, який є набором параметрів обробки вхідних даних і допомагає нейронній мережі правильно оцінювати вагу. Залежно від складності задачі може бути використано від кількох до сотень формул.

Кожна ітерація проходження через операнди називається епохою. Початкова епоха, коли мережа створена, має номер 0. Після першої епохи настає епоха 1 і так далі. З кожною епохою помилка обчислень зменшується. Коли цей показник досягає невеликого значення (наприклад, кілька відсотків), мережу вважають навченою і придатною для розв'язання реальних задач. [7]

Початково штучні нейронні мережі були розроблені з метою розв'язання задач таким же способом, як робить це людський мозок. Проте з часом увага переключилася на відповідність певним когнітивним здібностям, що призвело до відхилень від біології. ШНМ знайшли застосування у різних задачах, включаючи комп'ютерне зорове сприйняття, розпізнавання мови, машинний

переклад, соціально-мережеве фільтрування, настільні та відеоігри, а також медичну діагностику. [7]

РОЗДІЛ 2. МАТЕМАТИЧНІ МЕТОДИ АНАЛІЗУ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ

2.1 Математичні основи систем комп'ютерного зору

Першим важливим аспектом систем комп'ютерного зору є використання матриць і векторів. Вони дозволяють ефективно представляти та обробляти зображення, а саме:

1. Представлення зображень у векторній формі.

Зображення можна розглядати як двовимірний масив пікселів, де кожен піксель має свої координати і значення яскравості. Для зручності обробки, зображення можна перетворити у вектор, де кожен піксель стає елементом вектора. Наприклад, якщо маємо зображення розміром 100×100 пікселів, то його можна представити у вигляді вектора розміром 10000 (100×100).

2. Операції над матрицями та векторами.

Лінійні алгебраїчні операції використовуються для обробки зображень, наприклад, для зсуву, масштабування, обертання та інших операцій. Їх можна виразити у вигляді математичних формул.

Приклади операцій:

- Зсув - додавання або віднімання вектора зображення на константу:

$$X_{new} = X + b, \quad (2.1)$$

де X_{new} - новий вектор зображення, X - початковий вектор зображення, b - вектор зсуву.

- Масштабування - множення вектора зображення на константу:

$$X_{new} = a \times X, \quad (2.2)$$

де X_{new} - новий вектор зображення, X - початковий вектор зображення, a - коефіцієнт масштабування.

- Обертання - перетворення вектора зображення за допомогою матриці обертання:

$$X_{new} = R \times X, \quad (2.3)$$

де X_{new} - новий вектор зображення, X - початковий вектор зображення, R - матриця обертання.

- Інші операції: Додавання, віднімання, множення матриць або векторів, елементарні операції (наприклад, взяття модуля, піднесення до степеня і т.д.).

3. Матриці та вектори у фільтрації та згладжуванні зображень:

Матриці використовуються для визначення фільтрів, які застосовуються до зображень. Застосування фільтрів до зображення може виконуватися шляхом згортки зображення з фільтром за такою формулою:

$$X_{new}(i, j) = \sum \sum X(i + m, j + n) \times H(m, n), \quad (2.4)$$

де X_{new} - нове зображення, X - початкове зображення, H - матриця фільтра, (m, n) - індекси зсуву.

Наступний аспект - це операції лінійної алгебри, що використовуються для обробки зображень у системах комп'ютерного зору. Вони дозволяють виконувати різноманітні операції над зображеннями, такі як зміна контрастності, розмірів, поворот, перетворення кольорів та інші. Розглянемо деякі основні операції лінійної алгебри для обробки зображень:

1. Масштабування зображення.

Масштабування зображення виконується шляхом множення його піксельних значень на певний коефіцієнт:

$$X_{new}(i, j) = a \times X(i, j), \quad (2.5)$$

де X_{new} - нове зображення, X - початкове зображення, a - коефіцієнт масштабування.

2. Зсув зображення.

Зсув зображення виконується шляхом додавання або віднімання певного значення з його піксельних значень:

$$X_{new}(i, j) = X(i, j) + b, \quad (2.6)$$

де X_{new} - нове зображення, X - початкове зображення, b - значення зсуву.

3. Обертання зображення.

Обертання зображення виконується шляхом застосування матриці обертання до його піксельних значень:

$$X_{new}(i, j) = R \times X(i, j), \quad (2.7)$$

Де X_{new} - нове зображення, X - початкове зображення, R - матриця обертання.

4. Лінійне перетворення кольорів.

Лінійне перетворення кольорів виконується шляхом множення піксельних значень зображення на матрицю перетворення кольорів:

$$X_{new}(i, j) = M \times X(i, j), \quad (2.8)$$

де X_{new} - нове зображення, X - початкове зображення, M - матриця перетворення кольорів. [9]

Підсумовуючи, можна сказати, що використання матриць та векторів у системах комп'ютерного зору дозволяє ефективно обробляти, аналізувати та вирішувати завдання, пов'язані з зображеннями. Ці дані надають основні інструменти для розуміння, обробки та використання зображень у широкому спектрі додатків, включаючи розпізнавання об'єктів, візуальний пошук, розробку комп'ютерного зору та багато інших.

2.2 Теорія ймовірностей та статистика у системах комп'ютерного зору

Теорія ймовірностей та статистика грають важливу роль у системах комп'ютерного зору, де метою є розуміння та обробка зображень за допомогою комп'ютера.

Статистичні методи сегментації та класифікації зображень використовуються для виділення об'єктів на зображенні (сегментація) та їх подальшої класифікації на основі статистичних ознак. Ці методи базуються на

аналізі статистичних характеристик пікселів або областей зображення та використанні математичних моделей для опису цих характеристик.

Основні статистичні методи:

1. Середнє значення - середнє арифметичне всіх піксельних значень на зображенні:

$$\mu = \frac{\sum x}{N}, \quad (2.9)$$

де μ - середнє значення (середнє арифметичне) на зображенні, $\sum x$ - сума всіх піксельних значень на зображенні, N - кількість пікселів на зображенні.

2. Дисперсія - міра розсіювання піксельних значень відносно середнього значення:

$$\sigma^2 = \frac{1}{N} \times \sum (X(i, j) - \mu)^2, \quad (2.10)$$

де N - кількість пікселів у зображенні, $X(i, j)$ - піксельне значення, μ - середнє значення.

3. Кореляція - ступінь залежності між піксельними значеннями двох зображень або регіонів зображення:

$$r = \frac{\sum((X - \mu_X) \times (Y - \mu_Y))}{N \times \sigma_X \times \sigma_Y}, \quad (2.11)$$

де: X і Y - піксельні значення, μ_X і μ_Y - середні значення піксельних значень X і Y відповідно, σ_X і σ_Y - стандартні відхилення піксельних значень X і Y відповідно, N - загальна кількість пікселів на зображенні.[4]

4. Сегментація - виділення об'єкта на зображенні.

Одним з методів сегментації є пороговий аналіз. Він базується на використанні порогового значення, яке відокремлює об'єкти від фону на зображенні. Прикладом порогового аналізу є метод Оцу (Otsu's method). Цей метод шукає порогове значення, яке мінімізує дисперсію пікселів усередині класів і максимізує дисперсію пікселів між класами:

$$\sigma_W^2 = w_0 \times \sigma_0^2 + w_1 \times \sigma_1^2, \quad (2.12)$$

$$\sigma_B^2 = w_0 \times w_1 \times (\mu_0 - \mu_1)^2, \quad (2.13)$$

$$\sigma_T^2 = \sigma_W^2 + \sigma_B^2, \quad (2.14)$$

$$\eta = \frac{\sigma_B^2}{\sigma_T^2}, \quad (2.15)$$

де w_0 та w_1 - ймовірності фону та об'єкту, σ_0^2 та σ_1^2 - дисперсії фону та об'єкту, μ_0 та μ_1 - середні значення фону та об'єкту, σ_W^2 - внутрішньокласова дисперсія, σ_B^2 - міжкласова дисперсія, σ_T^2 - загальна дисперсія, η - коефіцієнт Оцу.

Іншим методом сегментації є метод активних контурів, наприклад, метод зміюк (snake method). Він використовує контур, що може змінювати свою форму, для виділення об'єктів. Контур притягується до країв об'єктів на зображенні шляхом мінімізації функціоналу енергії, який включає в себе статистичні характеристики пікселів вздовж контуру.[9]

У класифікації зображень статистичні методи використовуються для визначення класу або категорії об'єкта на зображенні на основі його статистичних властивостей. Один з найпоширеніших методів - метод k-найближчих сусідів (k-nearest neighbors, k-NN). Він використовує набір тренувальних зображень з відомими класами для навчання моделі. При класифікації нового зображення, модель порівнює його статистичні ознаки (наприклад, середнє значення, дисперсія, гістограма) з тренувальними зображеннями і визначає його клас на основі k найближчих тренувальних зображень.[4]

Моделі ймовірностей для розпізнавання об'єктів використовуються для визначення ймовірності того, що деякий об'єкт на зображенні належить до певного класу або категорії. Ці моделі базуються на статистичних властивостях об'єктів та фону і використовуються для порівняння ознак об'єкта зі знаннями про класи.

Ймовірність визначається як відношення кількості сприятливих випадків до кількості всіх можливих випадків і приймає значення від 0 до 1:

$$P(A) = \frac{N(A)}{N(S)}, \quad (2.16)$$

де $P(A)$ - ймовірність події A , $N(A)$ - кількість сприятливих випадків, $N(S)$ - кількість всіх можливих випадків.

Однією з найпоширеніших моделей ймовірності є гаусівська модель. Вона використовується для моделювання ознак об'єктів як гаусівських розподілів. Гаусівський розподіл (або нормальний розподіл) описується такою формулою:

$$p(x) = \frac{1}{\sqrt{2 \times \pi \times \sigma^2}} \times \exp\left(-\frac{(x-\mu)^2}{2 \times \sigma^2}\right), \quad (2.17)$$

де $p(x)$ - ймовірність значення x , μ - середнє значення розподілу, σ^2 - дисперсія розподілу.

Гаусівська модель використовується для моделювання ознак, таких як яскравість пікселів, текстурні ознаки тощо. Щоб використати дану модель для розпізнавання об'єктів, спочатку вивчається гаусівський розподіл для кожного класу на основі тренувальних зображень, визначаються їх середні значення та дисперсії. Потім, для нового зображення, порівнюються його ознаки з гаусівськими розподілами кожного класу, і визначається ймовірність належності до кожного класу.[10]

Загалом, теорія ймовірностей та статистика є важливими інструментами у системах комп'ютерного зору. Вони дозволяють моделювати статистичні властивості зображень та об'єктів, виконувати класифікацію, оцінювати параметри моделей та здійснювати статистичний аналіз даних.

2.3 Геометричні методи аналізу зображень

Геометричні методи аналізу зображень використовують геометричні властивості зображень для виявлення, опису та вимірювання об'єктів на зображенні. Ці методи базуються на розпізнаванні форм, розташуванні об'єктів, вимірюванні відстаней та кутів, а також на використанні геометричних перетворень для вирішення завдань обробки зображень.

Перетворення координат та геометрична ректифікація зображень є важливими процедурами у обробці та аналізі зображень. Вони дозволяють

змінити форму, положення або перспективу зображення з метою поліпшення якості, виправлення деформацій або підгонки до заданих стандартів.

Перетворення координат дозволяє перетворити точки на зображенні з однієї системи координат в іншу. Це корисно, коли необхідно порівняти або обробити дані з різних джерел або в різних форматах. Одним із найпоширеніших видів перетворень координат є афінне перетворення.

Афінне перетворення задається матрицею з 2×2 блоком, що відповідає за лінійне перетворення, та вектором зсуву:

$$x' = ax + by + c, \quad (2.18)$$

$$y' = dx + ey + f, \quad (2.19)$$

де (x', y') - нові координати точки, (a, b, c, d, e, f) - параметри афінного перетворення.

Геометрична ректифікація зображень використовується для виправлення геометричних спотворень, які виникають при зйомці або обробці зображень. Цей процес дозволяє змінити положення, перспективу, розмір або форму об'єктів на зображенні.

Одним з найпоширеніших методів геометричної ректифікації є афінна трансформація. Застосування афінної трансформації до зображення включає два основних кроки:

1. Вибір точок відповідності.

Обираються відповідні точки на вихідному та цільовому зображеннях. Ці точки визначають залежність між двома зображеннями та потрібну трансформацію.

2. Обчислення матриці афінної трансформації: За допомогою обраних точок відповідності обчислюється матриця афінної трансформації. Ця матриця задає параметри перетворення, яке буде застосовуватися до всього зображення:

$$[x' \ y' \ 1] = [x \ y \ 1] \times T, \quad (2.20)$$

де $[x \ y \ 1]$ - координати вихідної точки на зображенні, $[x' \ y' \ 1]$ - нові координати точки після трансформації, T - матриця афінної трансформації.

Отже, перетворення координат та геометрична ректифікація зображень використовуються для зміни форми, положення або перспективи зображень з метою поліпшення якості аналізу та обробки зображень. Ці процедури включають математичні формули та методи, що дозволяють точно обчислити нові координати пікселів на зображенні після перетворень.

Визначення геометричних властивостей об'єктів на зображеннях є важливою задачею в області аналізу зображень. Ці властивості включають розташування, форму, розмір, орієнтацію та інші характеристики об'єктів, які можуть бути використані для класифікації, розпізнавання або відслідковування об'єктів на зображеннях.

Основні геометричні властивості об'єктів на зображеннях можна обчислити за допомогою різних методів, як наприклад:

1. Центроїд - геометричний центр об'єкта, обчислений як середнє арифметичне координат пікселів, що належать об'єкту:

$$O = (x_c, y_c) = \left(\frac{\sum(x)}{N}, \frac{\sum(y)}{N} \right), \quad (2.21)$$

де (x, y) - координати кожного пікселя об'єкта, N - кількість пікселів, що належать об'єкту, а \sum позначає суму значень.

2. Прямокутник обмеження - найменший прямокутник, що обведе всі пікселі об'єкта. Він визначається мінімальними і максимальними значеннями координат пікселів об'єкта:

$$x_{min} = \min(x), \quad (2.22)$$

$$y_{min} = \min(y), \quad (2.23)$$

$$x_{max} = \max(x), \quad (2.24)$$

$$y_{max} = \max(y), \quad (2.25)$$

де (x, y) - координати кожного пікселя об'єкта.

3. Площа об'єкта - кількість пікселів, що належать об'єкту. Простий спосіб обчислення площі полягає в підрахунку кількості пікселів, які мають значення, що відповідають об'єкту.

4. Орієнтація об'єкта - властивість, що вказує на нахил або поворот об'єкта. Вона може бути визначена, наприклад, як кут між віссю X і віссю, яка проходить через центр об'єкта. [11]

Дані методи дозволяють отримати числові значення, які відображають розташування, форму та інші атрибути об'єктів, що можуть бути використані для подальшого аналізу та обробки зображень.

Використання геометричних методів аналізу зображень дозволяє отримати детальну інформацію про об'єкти на зображеннях і забезпечує широкий спектр можливостей для застосування.

РОЗДІЛ 3. РОЗГЛЯД ПРИКЛАДІВ РЕАЛІЗАЦІЇ СИСТЕМ КОМП'ЮТЕРНОГО ЗОРУ З РІЗНИМИ ГРАФАМИ РОЗРАХУНКІВ

3.1 Бібліотека TensorFlow

Для реалізації систем комп'ютерного зору є популярними бібліотеки глибокого навчання, серед яких TensorFlow. Дана бібліотека надає різноманітні інструменти та функції, що спрощують розробку та навчання моделей машинного навчання. Вона має графову структуру, де операції представлені як вузли, а дані - як ребра. Це дозволяє ефективно обчислювати графи, розподіляти обчислення на різні пристрої.

Архітектура TensorFlow має дві складові:

- Бібліотека: для визначення обчислювальних графів.
- Виконавче середовище: для виконання таких графів на різних апаратних платформах.

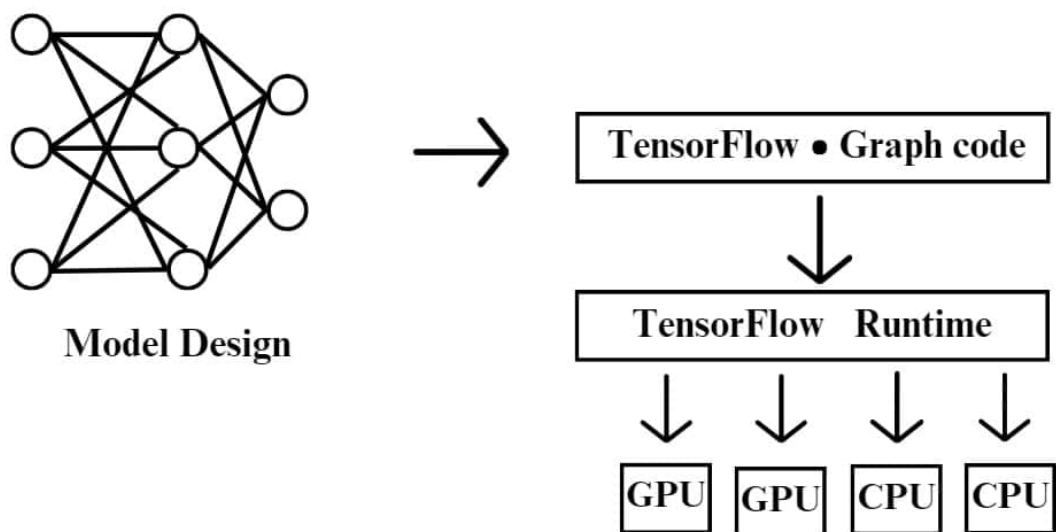


Рисунок 3.1 - Архітектура TensorFlow

Будуючи алгоритми машинного навчання з використанням TensorFlow вся робота полягає в перекладі моделей машинного навчання в обчислювальні графи TensorFlow та їх виконанні у виконавчому середовищі. Потім виконавче середовище бере на себе все інше.[12]

У TensorFlow алгоритми машинного навчання представлені у вигляді обчислювальних графів. Обчислювальний граф - це тип направленого графа, де вузли описують операції, а ребра представляють дані (тензори), що протікають між цими операціями.

Розглянемо основні складові графів:

1. Тензор - це опис багатовимірного масиву. Тензори можуть мати форму та тип даних, але не мають фактичних значень. Форми тензорів зазвичай можна вивести з графа.

2. Операція може мати нуль або більше вхідних значень та виробляти нуль або більше вихідних значень. Таким чином, операція може представляти математичне рівняння, змінну, константу або директиву потоку управління.

3. Всі навчальні параметри моделей машинного навчання є змінними `tf.Variable`. Змінна визначається своїм ім'ям, типом, формою та процедурою ініціалізації.

Слова "Tensor" і "Flow" означають, що тензори (дані) протікають через обчислювальний граф. Обчислювальні графи не є прив'язаними до конкретного TensorFlow - інші фреймворки для машинного навчання також використовують їх.

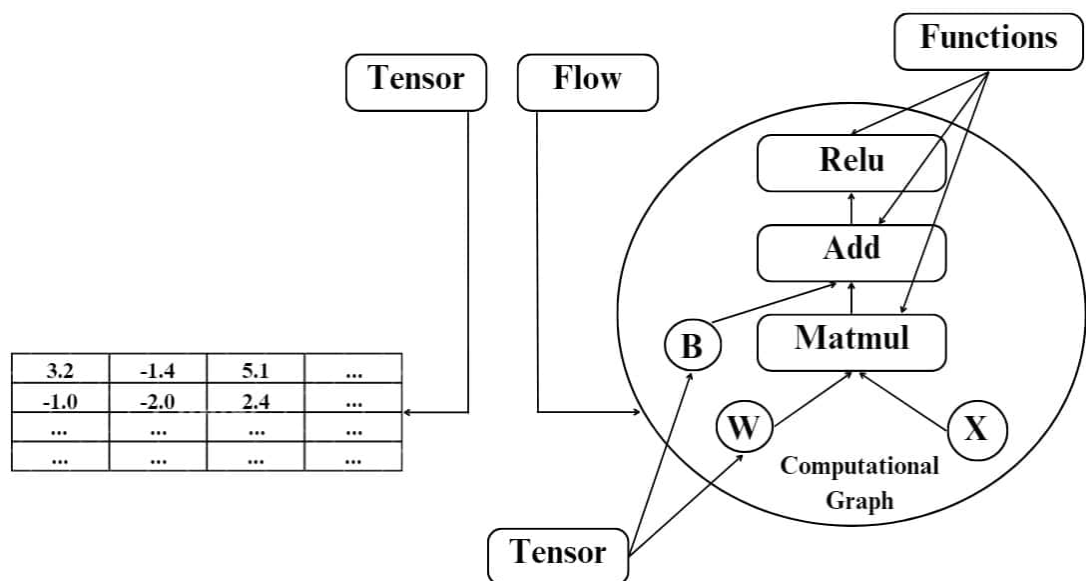


Рисунок 3.2 – Склад TensorFlow

Можна виділити деякі переваги використання обчислювальних графів:

1. Планування залежностей.

Зокрема, залежності даних, які вказують порядок виконання. Операції, які не залежать одна від одної, можуть бути заплановані паралельно.

2. Оптимізація графів.

Наприклад, усунення загальних підграфів.

3. Автоматичне диференціювання.

Описуючи обчислення у вигляді графів, можна легко обчислювати градієнти. Якщо будуть відомі градієнти кожного виходу операції відносно прямих входів, то за допомогою правила ланцюга можна знайти градієнти для будь-якого тензора відносно будь-якого іншого. Цей процес відомий як зворотне автоматичне диференціювання і дозволяє обчислити градієнт вузла графа відносно всіх інших в одному проході. [13]

3.2 Бібліотека Tensor Flow 1.0 та її особливості

TensorFlow 1.0 є однією з перших версій бібліотеки глибокого навчання TensorFlow, та входить в топ найпопулярніших бібліотек для побудови і навчання нейронних мереж.

Основні особливості TensorFlow 1.0:

1. Статичний режим побудови графа.

У TensorFlow 1.0 граф обчислень визначається перед запуском сесії. Граф складається з вузлів, які представляють операції, і ребер, що представляють потоки даних між вузлами. Це дозволяє використовувати оптимізації, такі як автоматичне розподілення обчислень на графічні процесори та розподілені системи.

2. Компіляція графу.

Перед виконанням, граф обчислень може бути оптимізований та скомпільований для покращення продуктивності. TensorFlow використовує

Just-in-Time (JIT) компіляцію, щоб перетворити граф в швидкодіючий код для виконання.

3. Гнучка розробка моделей.

Tensor Flow 1.0 надає велику гнучкість при розробці моделей глибокого навчання. Можна визначати власні вузли та операції, використовувати різні функції активації, втрачати та регуляризувати ваги, налаштовувати параметри оптимізаторів та багато іншого.

4. Розширені можливості.

TensorFlow 1.0 підтримує різні типи нейронних мереж, включаючи згорткові нейронні мережі (Convolutional Neural Networks), рекурентні нейронні мережі (Recurrent Neural Networks), а також високорівневі API для високорівневого моделювання.

При визначенні графа використовується набір функцій бібліотеки TensorFlow, для визначення обчислень, такий як `tf.Graph`. За допомогою `tf.Session` здійснюються фактичні обчислення, а `session.run` виконує граф і повертає значення тензора. Коли викликається `session.run()`, TensorFlow ідентифікує та виконує найменший набір вузлів, які необхідно обчислити для отримання запитаних тензорів.[14]

Узагальнюючи, можна зазначити, що в TensorFlow 1.0 графи є надійними та ефективними. Статичний граф та розширені можливості дозволяють реалізувати різні завдання комп'ютерного зору та обробки зображень з високою продуктивністю та гнучкістю. Проте дана версія складніше у використанні та вимагає від розробників додаткових знань та навиків.

3.3 Бібліотека Tensor Flow 2.0 та її особливості

TensorFlow 2.0 - оновлена версія відкритої бібліотеки машинного навчання TensorFlow. Вона пропонує багато нових функцій і покращень порівняно з попередньою версією.

Особливості TensorFlow 2.0:

1. Динамічний режим побудови графа.

TensorFlow 2.0 має більш простий та зрозумілий API порівняно з попередніми версіями. Він базується на концепції "eager execution" (негайне виконання), що дозволяє виконувати операції та отримувати результати без необхідності створення та запуску графа обчислень.

2. Удосконалене визначення моделей.

TensorFlow 2.0 надає вбудовану підтримку для Keras, високорівневої бібліотеки нейронних мереж. Keras став стандартом в TensorFlow 2.0 і дозволяє легко визначати та навчати моделі шляхом використання високорівневих API.

3. Покращена гнучкість.

TensorFlow 2.0 надає більшу гнучкість у виборі рівня абстракції. Можна використовувати високорівневий Keras API для швидкого створення моделей, а також мати доступ до низькорівневих можливостей TensorFlow для більшої контролю над обчисленнями.

4. Ефективність та швидкість.

TensorFlow 2.0 має оптимізовану реалізацію, що прискорює обчислення на багатьох типах пристроїв, включаючи графічні процесори (GPU) та тензорні процесори (TPU). Вона використовує автоматичне розподілення ресурсів для максимальної ефективності.[14]

Завдяки вбудованому режиму негайного виконання (Eager Execution), за замовчуванням, TensorFlow 2.0 полегшує деяку складність використання бібліотеки TensorFlow.

Негайне виконання - це гнучка платформа машинного навчання для досліджень та експериментів, яка надає:

- Інтуїтивний інтерфейс - дозволяє структурувати свій код природним чином та використовувати структури даних Python.

- Простота відлагодження - дає можливість викликати операторів безпосередньо для перевірки роботи моделей та тестування змін.
- Природний потік керування - відкриває можливість використовувати потік керування Python замість потоку керування графом.

Завдяки негайному виконанню (Eager Execution), не потрібно використовувати `tf.Session` для запуску коду. Код працює аналогічно до звичайного коду Python і дозволяє використовувати стандартні конструкції, такі як `if`, `while` і `for`, безпосередньо в потоці виконання. Значення тензорів обчислюються при зустрічі з ними в коді. Однак, оскільки немає графів у режимі негайного виконання, автоматична диференціація не може бути використана, і, відповідно, треба розраховувати на `tf.GradientType` для реєстрації операцій.

TensorFlow 2.0 також представляє AutoGraph - функціональність, яка перетворює код Python у більш ефективний графічний вигляд. AutoGraph автоматично виконує перетворення коду на основі певних правил та анотацій.

AutoGraph був розроблений для допомоги у використанні операторів управління потоком Python (`if`, `for`, `while`) з негайним виконанням. Ця можливість дозволяє писати моделі, використовуючи звичайний код Python з інструкціями потоку керування, і AutoGraph автоматично перетворює його на відповідний графічний вигляд. Таким чином, можна залишатись у зручному програмному середовищі, одночасно отримуючи ефективність та швидкість графічних обчислень, що пропонує TensorFlow.[13]

Підсумовуючи можна сказати, що версія TensorFlow 2.0 націлена на полегшення процесу розробки та навчання моделей машинного навчання, підвищення продуктивності та прискорення обчислень. Впровадження негайного режиму (Eager Execution) спрощує написання коду та робить його зрозумілим для користувачів, а додаток AutoGraph допомагає досягти оптимальної продуктивності та оптимізації. Варто враховувати, що хоч ці

нововведення вирішують багато проблем, вони впроваджують зміни в існуючу модель програмування, тим самим ускладнюючи її.

3.4 Порівняння реалізації бібліотек Tensor Flow 1.0 і Tensor Flow 2.0

TensorFlow 1.0 та TensorFlow 2.0 є двома важливими версіями бібліотеки машинного навчання TensorFlow, проте кожна з них має свої переваги та недоліки. Проаналізувавши їх особливості можна зрозуміти, що основна відмінність полягає у реалізації, а саме у тому, що TensorFlow 1.0 використовує статичний режим побудови графа, а TensorFlow 2.0 - динамічний.

Порівняння реалізації бібліотек буде здійснено за допомогою TensorFlow 2.0, оскільки в даній версії присутня можливість написання коду як за допомогою режиму негайного виконання (Eager Execution), так і графовим режимом, який притаманний саме TensorFlow 1.0.

За замовчуванням операції TensorFlow у версії 2.0 виконуються в режимі негайного виконання. Наприклад, `tf.matmul` (матричне множення) повертає результат без затримки у вигляді `tf.Tensor`. Натомість у графовому режимі `tf.matmul` повертає символічний посил на вузол у обчислювальному графі, а виконання множення відкладається. Якщо розбиратися детальніше, то у графовому режимі `tf.matmul` додає обчислювальний вузол (`tf.Operation`) до обчислювального графа (`tf.Graph`). Щоб скомпілювати та виконати обчислювальний граф, у версії TensorFlow 1.0, `session.run` викликається пізніше. Це відкладене виконання дозволяє автоматично запускати TensorFlow Grappler на задньому плані. Він застосовує довгий список оптимізацій графа для покращення продуктивності виконання. Наприклад, операції вузлів можна комбінувати або видаляти для забезпечення ефективності. Щоб скористатися цією оптимізацією у версії 2.0, потрібно виконувати код в графовому режимі, а не в режимі негайного виконання.

Внутрішнє вимірювання продуктивності TensorFlow показує, що в середньому використання TensorFlow 2.0 призводить до покращення продуктивності на 15%. Однак, ця різниця залежить від конкретної моделі, оскільки вона здатна зростати при наявності багатьох невеликих операцій і зменшується, коли використовуються менші кількості дороговартісних операцій, таких як згорткові нейронні мережі (CNN).[15]

Розглянемо реалізацію обчислення дисперсії піксельних значень відносно середнього значення з використанням TensorFlow.

Перш за все створюємо зображення за допомогою масиву, оскільки він є зручною структурою даних для зберігання та обробки піксельних значень зображення. Масив дозволяє нам організувати ці значення в структуровану форму, де кожен елемент масиву відповідає конкретному пікселю зображення. Щоб порівняння швидкодії було коректним на розгляд беремо два масиви різної розмірності. Перший масив 5×5 , другий - 10000×10000 .

Наступним кроком прописуємо визначення вхідних змінних використовуючи `tf.constant` для створення тензора зображення. Цей тензор забезпечить обчислення середнього значення та дисперсії.

Оскільки TensorFlow 1.0 використовує статичний режим побудови графа, а TensorFlow 2.0 - динамічний, для порівняння швидкодії, будуємо обидва види графів відносно однієї задачі. Для статичної побудови користуємось допомогою декоратора `tf.function`, який огортає функцію `compute_variance_static`, щоб вона використовувала статичну побудову графа TensorFlow. У цій функції застосовуємо тензорні операції для обчислення середнього значення і дисперсії зображення. Натомість динамічну побудову графа забезпечить функція `compute_variance_dynamic`, яка є просто звичайною функцією Python, що використовує тензорні операції для обчислення середнього значення і дисперсії зображення.

Важливий момент в коді - це запуск обчислень та вимірювання часу, оскільки саме завдяки цим даним буде змога порівняти швидкодію бібліотек.

Для статичної та динамічної побудови графа обчислення запускаються окремо, вимірюючи час виконання для кожного підходу. Для реалізації даних обчислень ми використовували `time.time()`, щоб отримати початковий час перед запуском обчислень і кінцевий час після завершення обчислень. Різниця між кінцевим і початковим часом рівна часу обчислення.

Коли всі ці пункти є прописані в кодї з'являється змога вивести результати середнього значення і дисперсії для статичної та динамічної побудови графа, а також час обчислення для кожного підходу.

Розглянемо результати отримані при запуску програми:

1. Для масиву 5×5 :

```
Середнє значення (статична побудова графа): 13.0
Дисперсія (статична побудова графа): 52.0
Час обчислення (статична побудова графа): 0.22523736953735352 секунд
Середнє значення (динамічна побудова графа): 13.0
Дисперсія (динамічна побудова графа): 52.0
Час обчислення (динамічна побудова графа): 0.001138448715209961 секунд
```

Рисунок 3.3 – Результати виведення програми з Додатку А

2. Для масиву 10000×10000 :

```
Середнє значення (статична побудова графа): 5000.5
Дисперсія (статична побудова графа): 8333333.0
Час обчислення (статична побудова графа): 0.40494656562805176 секунд
Середнє значення (динамічна побудова графа): 5000.5
Дисперсія (динамічна побудова графа): 8333333.0
Час обчислення (динамічна побудова графа): 0.5843138694763184 секунд
```

Рисунок 3.4 – Результати виведення програми з Додатку Б

Отже, швидкість виконання коду в TensorFlow залежить від різних факторів, включаючи розмір графа, операцій, обсяг даних та обладнання, на якому виконується код.

У деяких випадках, коли граф обчислень має велику кількість повторюваних операцій, статична побудова використанням `tf.function` може призвести до покращення швидкодії, оскільки граф може бути оптимізований і повторно використаний для різних вхідних даних. Це особливо корисно, коли

обчислення мають бути виконані багатократно, наприклад, при навчанні моделі на великих наборах даних.

Однак, в більшості випадків, динамічна побудова графа у режимі негайного виконання може бути швидшою та більш зручною для розробки і тестування моделей. В даному режимі кожна операція обчислюється та виконується безпосередньо, що зменшує накладні витрати на побудову та виконання графа.

Оцінивши отримані результати, конкретно у нашому прикладі, можна зробити висновок, що для невеликих масивів доречніше використовувати саме динамічну побудову графа. Цей підхід не тільки простіший у використанні, але й дає швидші результати. Проте, при більш масштабних масивах краще працює саме статичний граф.

ВИСНОВОК

В ході виконання дипломної роботи проаналізовано системи комп'ютерного зору, методи їх аналізу та розгляд прикладів реалізації з порівнянням графів двох видів.

В першому розділі дипломної роботи було розкрито питання, що таке комп'ютерний зір та розглянуто класифікацію його систем. Досліджено різні методи аналізу зображень та їх застосування в системах комп'ютерного зору, а саме методи: обробки зображень, сегментації зображень та розпізнавання об'єктів. А також детально описано штучні нейронні мережі.

Другий розділ присвячено математичним методам аналізу систем комп'ютерного зору. Досліджено математичні основи, які дали розуміння, що використання матриць та векторів у системах комп'ютерного зору дозволяє ефективно обробляти, аналізувати та вирішувати завдання, пов'язані з зображеннями. Розглянуто важливість теорії ймовірності та статистики, оскільки дані методи дозволяють моделювати статистичні властивості зображень та об'єктів, виконувати класифікацію, оцінювати параметри моделей та здійснювати статистичний аналіз даних. Визначено геометричні методи аналізу зображень, завдяки яким можна отримувати детальну інформацію про об'єкти на зображеннях.

Третій розділ розглядає приклади реалізації систем комп'ютерного зору з різними графами розрахунків. Дослідивши бібліотеку TensorFlow та відмінності між TensorFlow 1.0 та TensorFlow 2.0 було реалізовано обчислення дисперсії піксельних значень відносно середнього значення з використанням статичного та динамічного графів розрахунку для порівняння швидкодії. Отриманий результат було оцінено та зроблено висновки, що динамічна побудова графів успішно виконує обробку масивів з малою кількістю даних, проте з їх збільшенням час обробки зростає стрімкіше ніж при статичній

побудові графу, тому для масштабних масивів варту використовувати саме статичний граф. Ці дані підсумували розкриття теми дипломної роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Галузинський М.П., Коробов С.І. Комп'ютерний зір: теорія та практика. Київ: Видавничий дім ДМК, 2015. 408с.
2. Макушкин А.М., Заяц Д.В., Кириченко М.Ю. Обробка зображень на платформі OpenCV. Київ: Видавництво НТУУ "КПІ", 2012. 286с.
3. Forsyth, D., & Ponce, J. (2012). Computer Vision: A Modern Approach. Pearson Education. 802p.
4. Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer. 812p.
5. Чернов В.В., Сидоренко В.А. Методи та алгоритми аналізу зображень. Навчальний посібник. Київ: Видавництво Інституту кібернетики ім. В.М. Глушкова НАН України, 2010.
6. Michael Nielsen (2015). Neural Networks and Deep Learning. Determination Press. 475p.
7. Ian Goodfellow, Yoshua Bengio, Aaron Courville (2016). Deep Learning. MIT Press. 800p.
8. Christopher M. Bishop (2006). Pattern Recognition and Machine Learning. Springer . 738p.
9. О. В. Дідковський, В. А. Кардаш. Математичні основи комп'ютерного зору. Ліга-Прес, 2016. 396с.
10. What is Gaussian Distribution | Deepchecks. Deepchecks. URL: <https://deepchecks.com/glossary/gaussian-distribution/> (date of access: 11.06.2023).
11. Zisserman A., Hartley R. Multiple View Geometry in Computer Vision. 2nd ed. Cambridge University Press, 2004. 672 p.
12. Bisong E. TensorFlow. dvdabisong.github.io. URL: <https://ekababisong.org/gcp-ml-seminar/tensorflow/> (date of access: 11.06.2023).

13. Learn tensorflow 2.1.0 step by step. Kaggle: Your Machine Learning and Data Science Community. URL: <https://www.kaggle.com/code/kareem3egm/learn-tensorflow-2-1-0-step-by-step> (date of access: 11.06.2023).
14. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Incorporated, 2022.
15. Eager Execution vs. Graph Execution in TensorFlow: Which is Better? URL: <https://towardsdatascience.com/eager-execution-vs-graph-execution-which-is-better-38162ea4dbf6> (date of access: 11.06.2023).

ДОДАТКИ

Додаток А. Масив 5×5

```

import tensorflow as tf
import numpy as np
import time

image = np.array([
    [1, 2, 3, 4, 5],
    [6, 7, 8, 9, 10],
    [11, 12, 13, 14, 15],
    [16, 17, 18, 19, 20],
    [21, 22, 23, 24, 25]
])

image_tensor = tf.constant(image, dtype=tf.float32)

@tf.function
def compute_variance_static(image_tensor):
    mean_value_static = tf.reduce_mean(image_tensor)
    variance_static = tf.reduce_mean(tf.square(image_tensor -
mean_value_static))
    return mean_value_static, variance_static

def compute_variance_dynamic(image_tensor):
    mean_value_dynamic = tf.reduce_mean(image_tensor)
    variance_dynamic = tf.reduce_mean(tf.square(image_tensor -
mean_value_dynamic))
    return mean_value_dynamic, variance_dynamic

start_time_static = time.time()
mean_static, variance_static = compute_variance_static(image_tensor)
end_time_static = time.time()
computation_time_static = end_time_static - start_time_static

start_time_dynamic = time.time()
mean_dynamic, variance_dynamic = compute_variance_dynamic(image_tensor)
end_time_dynamic = time.time()
computation_time_dynamic = end_time_dynamic - start_time_dynamic

print("Середнє значення (статична побудова графа):", mean_static.numpy())
print("Дисперсія (статична побудова графа):", variance_static.numpy())
print("Час обчислення (статична побудова графа):",
computation_time_static, "секунд")

print("Середнє значення (динамічна побудова графа):",
mean_dynamic.numpy())
print("Дисперсія (динамічна побудова графа):", variance_dynamic.numpy())
print("Час обчислення (динамічна побудова графа):",
computation_time_dynamic, "секунд")

```

Додаток Б. Масив 10000×10000

```

import tensorflow as tf
import numpy as np
import time

image = np.array([[i for i in range(1, 10001)] for _ in range(10000)])

image_tensor = tf.constant(image, dtype=tf.float32)

@tf.function
def compute_variance_static(image_tensor):
    mean_value_static = tf.reduce_mean(image_tensor)
    variance_static = tf.reduce_mean(tf.square(image_tensor -
mean_value_static))
    return mean_value_static, variance_static

def compute_variance_dynamic(image_tensor):
    mean_value_dynamic = tf.reduce_mean(image_tensor)
    variance_dynamic = tf.reduce_mean(tf.square(image_tensor -
mean_value_dynamic))
    return mean_value_dynamic, variance_dynamic

start_time_static = time.time()
mean_static, variance_static = compute_variance_static(image_tensor)
end_time_static = time.time()
computation_time_static = end_time_static - start_time_static

start_time_dynamic = time.time()
mean_dynamic, variance_dynamic = compute_variance_dynamic(image_tensor)
end_time_dynamic = time.time()
computation_time_dynamic = end_time_dynamic - start_time_dynamic

print("Середнє значення (статична побудова графа):", mean_static.numpy())
print("Дисперсія (статична побудова графа):", variance_static.numpy())
print("Час обчислення (статична побудова графа):",
computation_time_static, "секунд")

print("Середнє значення (динамічна побудова графа):",
mean_dynamic.numpy())
print("Дисперсія (динамічна побудова графа):", variance_dynamic.numpy())
print("Час обчислення (динамічна побудова графа):",
computation_time_dynamic, "секунд")

```

Ознайомитись з реалізацією програм також можна за посиланням:

<https://github.com/m-skorochkina/student-applications/tree/develop/2022-2023/PMP-42/coursework/Maria%20Skorochkina>