Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра
(бакалавра, спеціаліста, магістра)

Студента Явтухов Артем Володимирович
(ПІБ)
академічної групи 126м-22-1
(шифр)
спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)
за освітньо-професійною програмою

«Інформаційні системи та технології»
(офіційна назва)
на тему Комплексна кваліфікаційна робота: Розробка інформаційної технології визначення об'єктів у системах машинного зору в умовах обмежених обчислювальних потужностей
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Коротенко Г.М.			
розділів:				

Рецензент	доц. Ширін А.Л.			

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро
2023

ЗАТВЕРДЖЕНО:
завідувач кафедри

___інформаційних технологій___
_____та комп'ютерної інженерії___
(повна назва)

_____ _Гнатушенко В.В._
(підпис) (прізвище, ініціали)

«_____»_____ 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня _____магістр _____
(бакалавра, спеціаліста, магістра)

студенту _____Явтухов А. В._____ **академічної групи** ___126м-22-1_ ___
(прізвище та ініціали) (шифр)

спеціальності _____126 «Інформаційні системи та технології»_____ _____

за освітньою-професійною програмою _____

«Інформаційні системи та технології»

на тему _____Комплексна кваліфікаційна робота: Розробка інформаційної технології визначення об'єктів у системах машинного зору в умовах обмежених обчислювальних потужностей_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 09.10.2021 р. № 1227-с

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз стану області рішення задачі	09.10.2023 – 20.10.2023
Розділ 2	Метрики оцінки якості алгоритмів та аналіз просторової складності описаних алгоритмів	21.10.2023 – 30.11.2023
Розділ 3	Порівняння архітектур нейромереж кластеризації в умовах обмежених обчислювальних потужностей з метою побудови відповідної ІТ	1.12.2023 – 18.12.2023

Завдання видано _____ _____Коротенко Г.М._____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____09.10.2023 р._____

Дата подання до екзаменаційної комісії ____21.12.2023 р.____

Прийнято до виконання _____ _____Явтухов А.В._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: стор. 85, рис. 8, табл. 3 додатки 4, джерела 37.

Об'єкт дослідження: інформаційна технологія визначення об’єктів на зображеннях в системах машинного зору за умов обмеженої продуктивності.

Предмет дослідження: нейромережеві методи визначення об’єктів на зображеннях в системах машинного зору з використанням процесорів штучного інтелекту.

Мета кваліфікаційної роботи: підвищення точності визначення об’єкту на зображеннях у системах машинного зору за умови використання нейромереж та спеціалізованих процесорів штучного інтелекту.

Новизна отриманих результатів полягає у розробці інформаційної технології визначення прямокутника для об’єкта з використанням нейромереж в системах машинного зору.

Пояснювальна записка містить опис аналіз теоретичних відомостей про нейромережеві алгоритми кластеризації, аналіз часової та просторової складності цих алгоритмів, опис метрик якості кластеризації та реалізацію нейромережевих методів кластеризації з результатами їх тестування за умови використання процесорів штучного інтелекту.

Список ключових слів: GNN, GCN, DeepCut, Hi-LANDER, MEA, K-means.

ABSTRACT

Explanatory note: pages 85, figures 8, tables 3, applications 4, sources 37.

Object of research: information technology for detecting objects in images in machine vision systems under conditions of limited performance.

Subject of research: neural network methods for detecting objects in images in machine vision systems using artificial intelligence processors.

Purpose of research: to improve the accuracy of object detection in images in machine vision systems by using neural networks and specialized artificial intelligence processors.

The novelty of the obtained results lies in the development of information technology for determining the rectangle for an object using neural networks in machine vision systems.

The explanatory note contains a description of the analysis of theoretical information about neural network clustering algorithms, analysis of the temporal and spatial complexity of these algorithms, description of clustering quality metrics, and implementation of neural network clustering methods with the results of their testing using artificial intelligence processors.

Keywords: GNN, GCN, DeepCut, Hi-LANDER, MEA, K-means.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GNN – графова нейронна мережа

GCN – згорткова графова нейронна мережа

DeepCut – власна назва архітектури графової нейронної мережі

Hi-LANDER – власна назва архітектури графової нейронної мережі

MEA – власна назва архітектури графової нейронної мережі

K-means – Алгоритм кластеризації, який розбиває дані на K кластерів, мінімізуючи середньоквадратичне відхилення між точками і центроїдами кластерів.

ВСТУП

Актуальність роботи. Сучасний стан розвитку систем штучного інтелекту дозволяє застосовувати нейромережеві алгоритми задля забезпечення функціонування систем машинного зору, слуху тощо. Однак класична архітектура роботизованих систем, що використовують нейронні мережі до недавнього часу здебільшого дозволяла тільки клієнт-серверну обробку. Тобто сама роботизована система мала якомога швидше та точніше зібрати інформацію та передати в обчислювальну систему, в якій працює нейронна мережа. Така процедура обумовлювалась тим, що в разі великої роздільної здатності сенсорів роботизованої системи, розмір вхідних даних не дозволяв застосовувати нейронні мережі прямого розповсюдження безпосередньо на роботах. Згорткові мережі хоча й покращили стан справ, однак повністю проблема не зникла, тож до недавнього часу використання нейронних мереж безпосередньо у роботизованих системах було обмежено.

Але останнім часом в апаратних та програмних методах штучного інтелекту відбулися суттєві зміни. З’явилися алгоритми глибокого навчання, які дозволяють використовувати одні й ті самі навчанні моделі на вхідних даних різного розміру. З’явилися нові типи нейромережевих архітектур та архітектури різного призначення для розв’язання найрізноманітніших практичних задач. А також з’явилися портативні процесори штучного інтелекту, які суттєво поступаються потужним графічним процесорам, що використовуються під час навчання нейронних мереж. Проте потужність портативних процесорів штучного інтелекту суттєво перевищує продуктивність класичних процесорів мобільних пристроїв, що надає можливість швидко та ефективно використовувати навченні моделі безпосередньо на роботах. Однією с таких задач машинного зору є виявлення об’єктів на зображенні.

Об’єктом досліджень є нейромережеві методи та архітектури нейронних мереж задля кластеризації точок на зображенні.

Предметом досліджень є часова та просторова складність нейромережевих кластеризаційних архітектур під час їх використання у системах, обладнаних портативним процесором штучного інтелекту.

Мета роботи – підвищення точності кластеризації за умови використання нейронних мереж та процесорів штучного інтелекту.

Пояснювальна записка містить пояснювальний опис архітектур нейронних мереж для кластеризації, теоретичних оцінок їх просторової та обчислювальної складності та практичних результатів тестування різних архітектур нейронних мереж на пристрої, обладнаному портативним процесором штучного інтелекту.

Постановка задачі

За результатами аналізу нейронних мереж різного типу, призначеного для кластеризації точок на зображенні, їхньої теоретичної просторової та часової складності, провести випробування практичного застосування цих алгоритмів на пристрої, обладнаним портативним процесором штучного інтелекту. За результатами практичних випробувань обґрунтувати використання певної архітектури нейронної мережі для пристроїв з цим типом процесорів штучного інтелекту.

Висновок. У даній атестаційній роботі будуть розглянуті нейромережеві методи кластеризації, їхня будова, теоретичні оцінки часової та просторової складності, будуть проведенні випробування архітектур нейронних мереж з використанням портативного процесору штучного інтелекту від компанії Texas Instruments та буде обґрунтовано використання однієї з цих архітектур.

РОЗДІЛ 1
АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧІ

1. **Стан проблема**

Завдяки розвитку мікроелектроніки та математичних методів, що пов'язані з нейронними мережами, з’явилася можливість створювати розумніші машини, що обладнані автономними системами машинного зору, слуху, прийняття рішень тощо. Завдяки таким нейронним мережам, роботи отримуються здатність досліджувати навколишній простір, визначаючи у ньому об’єкти, реагувати на зовнішні подразники та реакції об’єктів навколишнього світу. Портативні чіп, що забезпечують такі можливості, є доволі новими на ринку мікроелектроніки і завжди, коли розробник обирає апаратну основу системи та метод, що лежить в основі її програмного забезпечення, постає суттєве питання, що з чим поєднати. Різниця у внутрішній будові чіпу від різних виробників є доволі суттєвою, і одна й та сама архітектура нейронної мережі на чипах різних компаній може працювати з різною продуктивністю. Слід також пам’ятати, що портативний процесор штучного інтелекту хоч і має більшу продуктивність у порівнянні з традиційним мобільним процесором, все ж дуже сильно поступається у продуктивності традиційним системам, що використовуються у машинному та глибокому навчанні. То ж ця атестаційна робота присвячена дослідженню архітектур кластеризаційних нейронних мереж та їхнього використання на портативних процесорах від компанії Texas Instruments.

1.1. Кластеризація у неконтрольованому навчанні

Підсистема машинного зору в роботизованих системах залежно від задачі задля якої вона створена, потребує різних алгоритмів обробки зображень та відео. Найпримітивніші алгоритми здатні фіксувати рух просто оцінюючи зміну кадрів у відео потоці, зазвичай такі алгоритми є різновидами рекурентних нейронних мереж і недоліком таких алгоритмів є те, що за

наявності нерухомих об’єктів на зображеннях виявити їх неможливо. Інші підходи, які намагаються зрозуміти, що перед ними, здебільшого ґрунтуються на аналізі мапи кольорів зображення або телевізійної мапи у разі використання інфрачервоних камер, сама технологія виявлення об'єкта може бути реалізована як у вигляді пошуку прямокутного фрагмента зображення, де міститься той чи інший об’єкт, так і сегментації зображення, яка дозволяє визначити не тільки наявність об’єктів у кадрі, а й відносну віддаленість до них, тобто завдяки сегментації можна сказати, що один об’єкт є ближчим за інший. Ці дві задачі виявлення не є конкурентними, оскільки задача визначення прямокутника використовується у разі класифікації та стеження за об'єктом, а задача визначення відносної віддаленості може бути використана в алгоритмах аналізу сцен робота, якого може бути пов’язана з підтримкою прийняття рішень. Отже, ця атестаційна робота присвячений нейромережевому способу виявлення прямокутника в якому міститься об’єкт, однак деякі з описаних архітектур можуть бути застосовані й у випадку сегментації. Фактично виявлення прямокутника за допомогою нейромереж зводиться до розв’язання задачі нейромережевої кластеризації зображення з подальшим обходом отриманої мапи розподілу кластерів та визначення меж прямокутників для кожного кластеру, що може бути виконане за $O(n\hat{}2)$ часу.

1.2. Нейронні мережі для некерованого навчання

1.2.1. Hi-Lander

Представлений ієрархічний GNN використовує новий підхід для об'єднання зв'язаних компонентів, передбачених на кожному рівні ієрархії, для формування нового графа на наступному рівні. На кожному рівні GNN використовується для прогнозування зв'язності між вузлами графа. Ітерації продовжуються доти, доки GNN не припинить додавати нові ребра до графа.

У некерованій агломеративній кластеризації збіжність досягається, коли всі кластери об'єднуються в єдиний вузол або коли досягається довільний поріг довільного критерію складності моделі. У випадку Hi-Lander

збіжність визначається навчальною множиною і відбувається, коли GNN не додає більше ребер до графа.

1. Кластеризація за допомогою k-NN графа

Формально, задано набір з N зображень $D = \{I_i\}_{i=1}^{N}$та їх відповідних візуальних вкладень $F = \{f_i\}_{i=1}^{N}$, спочатку треба побудувати граф спорідненості G = {V, E}, де |V | = N, за допомогою k-найближчих сусідів визначається відносна косинусоїдальна подібність, тобто внутрішній добуток нормалізованих візуальних вбудовувань. Кожне зображення (наприклад, один зріз обличчя) тягне за собою один об'єкт до кластера і являє собою вузол на графі, характеристикою вузла є його візуальне вбудовування f_i[26]. Ребра з'єднують кожну вершину з її k сусідами. Згідно з парадигмами кластеризації, функція φ приймає на вхід граф спорідненості G та характеристики вузлів F, і виробляє реберну підмножину $E' \subset E$, тобто $E' = \varphi(G, F)$. Отриманий граф $G' = \{V, E'\}$ розбивається на зв'язні компоненти, кожна з яких відповідає кластеру вершин. Метод Hi-Lander побудовано на цій парадигмі кластеризації k-NN графів[30].

2. Ієрархічне узагальнення до Hi-LANDER

Для того, щоб врахувати природний рівень деталізації кластерів у наборі даних, пропонується ієрархічна кластеризація на основі k-NN.

Маючи набір початкових візуальних вкладень F і невелике фіксоване значення k фіксованому значенню k ітеративно генерується послідовність

графів $G_l = \{V_l, E_l\}$ та відповідних вершинних характеристик $H_i = \{h_i$ }, де i = 1 . . . $|V_l|$ та $l = 1 \ldots$, використовуючи базову кластерну функцію φ та функція агрегації ψ. Алгоритм 1 підсумовує запропонований процес ієрархічного узагальнення.

Алгоритм 1:

Вхід: N, F, k;

l <= 1;

H_1 <= F;

Доки не зійшлися робити

G_l<= k найближчий сусід (H_l, k);

$E_l^{`}$ <= φ(G_l, H_l);

$G_l^{`}$ <= зв’язанні компоненти ($E_l^{`}$);

H_{l+1}<= (H_l, $G_l^{`}$);

l <= $l + 1$;

Кінець

ID <= id-розповсюдження ($\{G_l\}$, $\{G_l^{`}\}$);

Повернути ID

Для початку треба визначити G_1 як G у k-NN кластеризації, а H1 =$\{f_i\}$. Функція φ виконує таку операцію:

$$E_l^{`} = \varphi(G_l, H_l) \quad (1.1)$$

беручи на вході характеристики вершин та k-NN графа на рівні l та отримавши вибрані підмножини ребер $E_l^{`}$. В результаті граф $G_l^{`}$= $\{V_l, E_l^{`}\}$ розбивається на множину зв'язних компонентів. Далі треба визначити множину зв'язних компонентів через $G_l^{'}$ як $\{c_i^{(l)}\}_{i=1}^{V_{l+1}}$, де $c_i^{(l)}$ і-й елемент. Для того, щоб згенерувати G_{l+1} , отримується V_{l+1}, H_{l+1} та E_{l+1} наступним чином. Спочатку визначається і-та вершина в G_{l+1}, $v_i^{(l+1)}$ І як сутність, що

представляє зв'язну компоненту $c_i^{(l)}$. Далі генеруються нові вектори ознак вузлів за допомогою функції агрегування ψ, яка виконує

$$H_{l+1} = \varphi(H_l, \grave{G_l})$$

(1.2)

Він агрегує особливості вузлів у кожному підключеному компоненті $c_i^{(l)}$ в один вектор ознак відповідно[12]. Нарешті, з цього отримується E_{l+1} шляхом пошуку k найближчих сусідів на H_{l+1}і з'єднавши кожну вершину з її k сусідами[13].

Генерація сходиться, коли більше не додається жодного нового ребра, тобто $\grave{E_l} = \emptyset$. Далі треба визначити L як довжину збіжної послідовності. Для остаточного призначення кластерів, починаючи з G_L, присвоюється ідентифікатор кластера (ID) i з'єднаній компоненті $c_i^{(l)}$ який поширює ідентифікатор i на всі свої вузли $\{v_j^{(L)} | v_j^{(L)} \in c_i^{(L)}\}$. Тоді для кожного $v_i^{(L)}$ поширює свою мітку до відповідної зв'язної компоненти $c_i^{(L-1)}$ попередньої ітерації. Цей процес розповсюдження ідентифікатора врешті-решт присвоює ідентифікатор кластера кожному вузлу у V_1, і це присвоєння використовується як остаточна прогнозована кластеризація. Надалі у цьому розділі буде описані: дизайн базової кластерної функції φ, функції агрегації ψ та загальна модель Hi-LANDER за допомогою набору метанавчання. «Ми також використовуємо назву LANDER для позначення нашої базової однорівневої моделі, подібної до однієї ітерації Hi-LANDER»[1].

3. Реалізація функції кластера φ

Щоб досягти високої точності, φ розробляється як модель, що навчається.

GNN-модель для кластеризації в контрольованому середовищі для роботи зі складними кластерними структурами, де кожен вузол v_i у V поставляється з міткою кластера y_i, але тільки у метанавчальній множині. На відміну від неконтрольованих методів кластеризації, Hi-LANDER не використовує явний критерій групування. Замість цього, він отримує критерій групування з аналізу даних.

Сучасні методи керованої кластеризації показують, що щільність і зв'язність є ефективними сигналами для навчання моделі GNN. Hi-LANDER використовує обидва ці сигнали.

Для підвищення ефективності та точності Hi-LANDER спільно прогнозує щільність і зв'язність[11], використовуючи вбудовування, створені одним кодувальником графів.

Оцінки щільності і зв'язності потім проходять через етап декодування графа для визначення зв'язності ребер і, таким чином, прогнозування кластерів.

4. Кодування графа

Для кожної вершини v_i з відповідною вхідною характеристикою h_i стек шарів Graph Attention Network (GAT) кодує кожен h_i як нову ознаку або вбудовування $h_i`$

Загалом, було виявлено, що альтернативні кодувальники (наприклад, шари згорткової мережі звичайного графа), дають подібну продуктивність.

Спільне передбачення для щільності та зв'язності Для кожного ребра (v_i, v_j) в E, треба об’єднати в характеристики вузла-джерела та вузла-приймача, отримані від кодера, у вигляді $[h_i`, h_j`]$ де [-, -] - оператор конкатенації, та подати його на багатошаровий персептрон (MLP) з

наступним перетворенням softmax для отримання ймовірностей зв'язку $p_{ij} = P(y_i = y_j)$, тобто оцінку ймовірності того, що це ребро з'єднує дві вершини з однаковими мітками. Також це значення використовується для прогнозування оцінки псевдощільності вузлів $\hat{d}_i$ яка вимірює зважену за схожістю частку однокласних вузлів у його околиці. Для цього спочатку кількісно оцінюється схожість a_{ij} між вершинами v_i та v_j як внутрішній добуток їх відповідних характеристик, тобто $a_{ij} = \langle h_i, h_j \rangle$. Згодом, обчислюються відповідні реберні коефіцієнти $\hat{e}_{ij}$ як

$$\hat{e}_{ij} = P(y_i = y_j) - P(y_i \neq y_j) \tag{1.3}$$

де j індексує k найближчих сусідів v_i. Тоді стає можливим визначити $\hat{d}_i$ як

$$\hat{d}_i = \frac{1}{k} \sum_{j=1}^{k} \hat{e}_{ij} * a_{ij} \tag{1.4}$$

Ця оцінка призначена для наближення до істинного значення псевдогустини d_i яка отримується простою заміною $\hat{e}_{ij}$ у рівнянні 1.4 на $e_{ij} = 1(y_i = y_j) - 1(y_i \neq y_j)$ з використанням мітки істинних класів, де 1 - індикаторна функція. За побудовою, d_i є великим, коли найбільш схожі сусіди мають спільні мітки; в іншому випадку вона є малою. І що важливо, апроксимуючи d_i в термінах $\hat{e}_{ij}$ через p_{ij}, отриманий механізм спільного передбачення зменшує параметри для головки передбачення під час навчання, що дозволяє обом завданням отримати вигоду одна від одної. Головка передбачення - це нейронна мережа, яка використовується для прогнозування мітки для даного вузла. Вона отримує на вхід два вектори: вектор ознак даного вузла і вектор подібності між даним вузлом і його найближчими сусідами.

5. Декодування графа

Після того, як було отримано ймовірності зв'язків і оцінки щільності вузлів, їх потрібно перетворити на остаточні кластери за допомогою процесу декодування. Попередні методи використовують аналогічний процес декодування, але Hi-LANDER адаптує цей процес до отриманих спільних оцінок щільності та зв'язку. Треба почати з $E' = \emptyset$. Враховуючи $\hat{e}_{ij}$, $\hat{d}_i$, p_{ij} та поріг зв'язності ребер p_τ , треба визначити множину ребер-кандидатів E(i) для вершини v_i як

$$E(i) = \{j|(v_i, v_j) \in E \text{ and } \hat{d}_i \leq \hat{d}_j \text{ and } p_{ij} \geq p_\tau\} \quad (1.5)$$

Для будь якого i, якщо E(i) не пусте, треба обрати

$$j = argmax\, \hat{e}_{ij} \text{ j} \in E(i) \quad (1.6)$$

і треба додати (v_i, v_j) до E`. Важливо підкреслити, що вибір порогу крайового з'єднання p_τ є гіперпараметричним процес налаштування лише на валідаційній множині, відокремленій від метанавчальної[33]. Після метанавчання параметри Hi-LANDER не змінюються. Це відрізняє Hi-LANDER від некерованих методів агломеративної кластеризації, які можуть використовувати різні параметри для різних тестових наборів. Крім того, визначення E(i) гарантує, що кожен вузол v_i з непорожнім E(i) додає рівно одне ребро до E'. З іншого боку, кожна вершина з порожнім E(i) стає вершиною без вихідних ребер. При цьому умова $d_i \leq d_j$ вносить індуктивний зсув у встановлення зв'язків. Оскільки вершини з низькою щільністю мають тенденцію бути ті, що мають околиці, які перетинаються з іншими класами, або вузли на межі між декількома класами, зв'язки з такими вершинами часто є небажаними. Після повного проходу над кожною вершиною, E' формує набір з'єднаних компонент G' які слугують так званими кластерами.

6. Реалізація функції агрегації ψ

Треба позначити $c_i^{(l)}$ i-ту зв'язну компоненту в G_l`. Щоб побудувати $G_{l+1} = \{V_{l+1}, E_{l+1}\}$, спочатку потрібно перетворити $c_i^{(l)}$ у G_l у вершину $v_i^{(l+1)}$ у V_{l+1}. Визначаються два вектори ознак для нової вершини, а саме ознаку тотожності та середню характеристику $\tilde{h}_i^{(l+1)}$ І $\overline{h_i^{(l+1)}}$ як

$$\tilde{h}_i^{(l+1)} = \tilde{h}_{mi}^{(l)} \text{ і } h_i^{(l+1)} = \frac{1}{\left|c_i^{(l)}\right|} \sum_{J \epsilon c_i^{(l)}} \tilde{h}_j^{(l)} \quad (1.7)$$

Де $m_i = argmax_{J \epsilon c_i^{(l)}} \hat{d}_j^{(l)}$ позначає вершину вузла індекс приєднаної компоненти $c_i^{(l)}$. Крім того, на першому рівні, $\tilde{h}_i^{(0)} = \overline{h_i^{(0)}} = f_i$, де f_i візуальна ознака вбудовування. Вхідна функція наступного рівня для базової кластерної функції φ вузла $v_i^{(l+1)}$ є конкатенація пікової ознаки та середнього значення, тобто $h_i^{(l+1)} = [\tilde{h}_i^{(l+1)}, \overline{h_i^{(l+1)}}]$. Тобто було емпірично виявлено, що безпосереднє використання однієї з ознак дає такі ж результати, як і конкатенація на деяких перевірочних множинах, і було вирішено залишити це як гіперпараметр. Ознака тотожності $\tilde{h}_i^{(l)}$ може бути використана для ідентифікації схожих вузлів в ієрархіях, в той час як середня ознака $\overline{h_i^{(l)}}$ надає огляд інформації для всіх вузлів у кластері.

7. Hi-LANDER Learning

Оскільки об'єднані функції для супервузлів, $\tilde{h}_i^{(l+1)}$ та $\overline{h_i^{(l+1)}}$ завжди лежать в одному візуальному просторі вкладень що й вузлові ознаки $h^{(l)}$ попереднього рівня, ті ж самі.

Параметри моделі GNN можуть використовуватися для декількох рівнів ієрархічної структури, щоб вивчити природну гранулярність кластерного розподілу метанавчальної вибірки.

8. Ієрархічна стратегія навчання

Маючи k та мітки базової істинності, тепер можливо визначити рівень L, на якому ієрархічна агломерація сходиться. Таким чином, потрібно будувати послідовність графів $\{G_l\}$ за алгоритмом, описаним в Алгоритмі 1, з тією лише різницею, що використовуються базові реберні зв'язки $\{\grave{E}_l^{gt}\}$ на всіх рівнях і, таким чином, істинні проміжні кластери $\{\grave{G}_l^{gt}\}$ для побудови графів. Далі треба ініціалізувати LANDER і навчати його на всіх проміжних графах $\{G_l\}$ За одну епоху відбувається проходження циклічно через кожен G_l виконується прохід вперед за графом $\{G_l\}$, обчислюються втрати, як буде визначено далі, а потім оновлюються параметри моделі за допомогою зворотного розповсюдження.

9. Навчальні втрати

Модель Hi-LANDER навчається за допомогою композитної функції втрат, що задається формулою

$$L = L_{conn} + L_{den} \tag{1.8}$$

Перший член L_{conn} забезпечує нагляд за прогнозуванням попарних з'єднань через середню втрату зв'язності на кожне ребро

$$L_{conn} = -\frac{1}{|E|} \sum_{(v_i, v_j) \in E} l_{ij} \tag{1.9}$$

де l_{ij} - втрати на кожне ребро у вигляді

$$l_{ij} = q_{ij} log\, p_{ij} + \left(1 - q_{ij}\right) \log log \left(1 - p_{ij}\right), \text{ якщо } d_i \leq d_j$$

В іншому випадку $l_{ij} = 0$ (1.10)

Тут мітка базової істини $q_{ij} = 1(y_i = y_j)$ вказує чи належать дві вершини, з'єднані ребром, до одного кластера, і може бути обчислена на всіх рівнях, як описано раніше (аналогічно для базової істини d_i, отриманої зі значень q_{ij}). Тим часом другий доданок L_{den} відображає середню щільність втрат по сусідству, яка визначається за формулою

$$L_{den} = \frac{1}{|V|} \sum_{i=1}^{|V|} ||d_i - \hat{d}_i||_2^2 \quad (1.11)$$

Під час навчання як L_{conn}, так і L_{den} усереднюються заданих усіх рівнів.

1.2.2. **DeepCut**

Нехай $G = (V, E)$ - неорієнтований граф, утворений зображенням. Кожна вершина представляє область зображення, а ваги w_{ij} представляють спорідненість між областями зображення i та j, $i, j = 1 \dots n$. Нехай W n x n матриця, елементами якої є w_{ij}.

Мета полягає в тому щоб розбити граф на k непересічних множин $A_1, A_2 \dots A_k$ так, щоб $\cup_i A_i = V$ і $\forall_j \neq_i A_i \cap A_j = \emptyset$. Це розбиття можна подати у вигляді бінарної матриці $S \in \{0, 1\}^{n \times k}$, де $S_{ic} = 1$, якщо $i \in A_c$.

- Нормований розріз (N-розріз)

Хороший розріз - це такий, який містить багато внутрішньо групових зв'язків і мало між групових зв'язків. Кількість між групових зв'язків можна обчислити як загальну вагу ребер, які видаляються в процесі розрізання графа:

$$cut(A, B) = \sum_{u \in A,\ v \in B} w(u, v) \quad (1.12)$$

Де A, B - частини двостороннього розбиття G. Ця задача формулюється за допомогою функції нормалізованого розрізу

$$N_{cut}(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)} \quad (1.13)$$

де $assoc(A, V) = \sum_{i \in A,\ j \in V} w_{ij}$ - повна спорідненість, що з'єднує вершини A з усіма вершинами графа. Формулу для N розрізів можна легко розширити до K > 2 відрізків. Ши та Малік також запропонували наближений розв'язок рівняння (1.13)[10] за допомогою спектральних методів, відомих як спектральна кластеризація, де спектр (власні значення) матриці Лапласа графа використовується для отримання наближеного розв'язку задачі N-розрізів.

- Кореляційна кластеризація (КК)

Якщо граф, отриманий з зображення, містить як позитивні, так і негативні зв'язки, N-розріз вже не підходить. У цьому випадку хорошим сегментом зображення може бути такий, який максимізує позитивну спорідненість всередині сегмента і мінімізує негативну між сегментами. Цю мету можна сформулювати за допомогою функціоналу кластеризації кореляційних зв'язків (КЗ). За заданою матрицею ваг W оптимальне розбиття U, яке мінімізує:

$$CC(S) = -\sum_{ij} w_{ij} \sum_{c} S_{ic} S_{jc} \quad (1.14)$$

Кореляційна кластеризація використовує внутрішньокластерні розбіжності (відштовхування)[2] для автоматичного виведення кількості кластерів k, тому й не потрібно знати k наперед[35].

Графові нейронні мережі (GNN) - це тип нейронних мереж, призначений для обробки даних, структурованих у вигляді графів. Граф - це структура даних, що складається з вузлів і ребер, які з'єднують ці вузли. Вузли можуть представляти об'єкти, а ребра - їх зв'язки.

Шар GNN складається з двох фундаментальних операцій: передачі повідомлень та агрегації. Передача повідомлень передбачає, що кожен вузол графа обмінюється інформацією зі своїми сусідами. Інформація, якою обмінюються, може включати властивості вузлів, ребер або будь-які інші дані, вбудовані в граф.

Агрегація означає, що кожен вузол об'єднує інформацію, отриману від своїх сусідів, в одне повідомлення, яке оновлює поточний стан вузла.

У представленому підході буде використано специфічний тип GNN, який називається графозгортковою мережею (GCN)[27].

GCN - це тип GNN, який використовує операцію згортки для обробки інформації в графах. Згортка - це операція, яка використовується в традиційних нейронних мережах для обробки даних у вигляді матриць. У GCN згортка використовується для обробки інформації, яка передається між вузлами графа.

GCN - це потужний інструмент для обробки даних, структурованих у вигляді графів. Вони були успішно використані для вирішення різноманітних задач, таких як класифікація, сегментація та рекомендації.

У шарі GCN l кожен вузол h_v обробляється наступним чином[29]:

$$h_v^{(l+1)} = \sum_{u \in N(v)} \Theta * \frac{h_u^{(l)}}{|N(v)|} \quad (1.15)$$

де Θ - матриця параметрів, що навчаються. $N(v)$ позначає сусідів вершини h_v, а $|N(v)|$ - кількість сусідів. Агрегація GCN виконується за допомогою оператора підсумовування, причому доданок всередині суми представляє операцію передачі повідомлень. Метою навчання є оптимізація параметрів мережі Θ таким чином, щоб генерувати змістовні повідомлення для передачі між вузлами, оптимізуючи функцію втрат[21].

- Метод DeepCut

У представленому підході зображення обробляються за допомогою попередньо навченої мережі для виділення глибоких ознак. Ці ознаки використовуються для побудови зваженого графа, де вузли представляють пікселі зображення, а ребра - зв'язку між ними. Полегшена графова нейронна мережа (GNN) використовується для оптимізації функції втрат при розбитті графа (LNCut або LCC) для кожного графа зображення. Ця методологія застосовується для досягнення неконтрольованої локалізації, сегментації та семантичної сегментації об'єктів.

- Від глибоких ознак до графів

Маючи зображення M розміром m × n і d каналів, треба пропустити його через трансформер T. Трансформер ділить зображення на $\frac{mn}{p^2}$ ділянки, де p - розмір ділянки трансформера T. Для вилучення внутрішнього представлення трансформера для кожної ділянки буде використано ключ токен з останнього шару, оскільки він продемонстрував достатньо високу продуктивність у різних задачах. Результатом є вектор функцій $f(\frac{mn}{p^2} \times c)$, де c - розмірність вбудовування токенів, що містить всі вилучені ознаки з різних патчів. Розглянемо зважений граф $G = (V, E)$ з W як ваговою матрицею. Треба побудувати кореляційну матрицю по патчах з ознак, отриманих за допомогою трансформера:

$$W = ff^T \in R^{\frac{mn}{p^2} \times \frac{mn}{p^2}} \quad (1.16)$$

При кореляційній кластеризації від'ємні ваги (що представляють оцінки) несуть цінну інформацію, яка впливає на результат кластеризації. Однак результат нормалізованого зрізу приймає лише позитивні ваги (формула 1.17):

$$W = ff^T * (ff^T > 0) \in R^{\frac{mn}{p^2} \times \frac{mn}{p^2}} \quad (1.17)$$

Для адаптації процесу вибору кластерів у кореляційній кластеризації вводиться гіперпараметр k-чутливості, який позначається α. Оскільки кількість кластерів не може бути обрана безпосередньо при кореляційній кластеризації, цей параметр дозволяє контролювати чутливість процесу, де більше значення α відповідає більшій кількості кластерів. Це налаштування здійснюється наступним чином:

$$W = ff^T - \frac{max\left(ff^T\right)}{\alpha} \quad (1.18)$$

де $\alpha \in [1, +\infty)$. Менше значення α відповідає більшим силам відштовхування між вузлами (від'ємні ваги у W) і, таким чином, більшій кількості кластерів, оскільки кореляційна кластеризація max імітує від'ємну спорідненість між сегментами.

- Кластеризація графових нейронних мереж

Нехай $\hat{N}$ - матриця ознак вузлів, отримана шляхом застосування одного або декількох шарів згортки GNN до графа G з матрицею суміжності W. У даному випадку використовується одношарова GNN і будується граф, використовуючи матрицю кореляції з отриманих ознак ViT (Рівняння (1.16), (1.17), (1.18)). Нехай S - вихід багатошарового сприйняття (MLP) з функцією softmax, застосованою до

$$\hat{N} = GNN(N, W; \Theta_{GNN}),$$

$$S = MLP(\hat{N}; \Theta_{MLP}) \quad (1.19)$$

де Θ_{MLP} і Θ_{GNN} - параметри, що навчаються. Вихід GNN S - це матриця розподілу кластерів, яка містить вектори, що представляють ймовірність приналежності вузла до певного кластера. Оптимізація GNN здійснюється за допомогою релаксації з нормалізованим зрізом або за допомогою нового методу з кореляційною кластеризацією як функцією втрат. Це дозволяє

зберегти основні властивості графа, наприклад зв'язність. Функція втрат у випадку нормалізованого зрізу має вигляд:

$$L_{NCuts} = \frac{Tr\left(S^T WS\right)}{Tr\left(S^T DS\right)} + \left|\left|\frac{S^T S}{||S^T S||_F} - \frac{IK}{\sqrt{K}}\right|\right| \quad (1.20)$$

де $D = diag(\sum_j w_{ij})$ - матриця суми діагоналей W. K - це кількість кластерів, на які потрібно розбити граф. IK - матриця ідентичності. Перший доданок цільової функції сприяє об'єднанню вузлів, які сильно пов'язані між собою. Другий доданок сприяє тому, щоб кластерні розподіли були ортогональними і мали подібні розміри. Функція втрат для кореляційної кластеризації має вигляд:

$$L_{CC} = -Tr(WSS^T) \quad (1.21)$$

Цей член сприяє тому, щоб вузли всередині одного кластера були схожими один на одного, а вузли з різних кластерів були різними. W визначається з рівнянь (1.17) і (1.18) для втрат при N-розрізі і CC, відповідно. Для того, щоб процес кластеризації для CC був більш чутливим до невеликих відмінностей між кластерами, використовується значення k-чутливості.

- Сегментація графової нейронної мережі

Раніше було запропоновано побудувати граф з глибинних ознак, витягнутих з навчених без нагляду ViT. Кожен вузол графа представляє ділянку вихідного зображення. Потім вузли графа кластеризують в розрізнені множини, які відповідають різним сегментам зображення.

Для цього процесу кластеризації можна використовувати кластеризацію графових нейронних мереж. Вона може використовуватися з двома типами втрат: CC або N-розрізні.

На відміну від попередніх підходів, які використовували втрати N-зрізу, які визначені суворо для додатних ваг, у цьому підході рекомендується використовувати функціонал кореляційної кластеризації як втрати.

Кореляційна кластеризація дозволяє використовувати від'ємні ваги для побудови графа. Це полегшує кластеризацію без попереднього визначення кількості кластерів.

Крім того, у цьому підході глибинні ознаки використовуються як вузлові ознаки для побудови графа. Це відрізняється від інших методів, які використовували лише кореляційну матрицю глибинних ознак.

В результаті, запропонований метод дозволяє класифікувати ознаки під час кластеризації та неявно полегшує сегментацію семантичної частини без необхідності подальшої обробки.

Даний підхід є універсальним і може бути застосований до різних завдань, пов'язаних із розбиттям зображень.

Локалізація об'єктів - це процес виявлення і визначення меж основного об'єкта на зображенні. Для цього потрібно виконати наступні кроки:

1. Треба використати запропонований метод кластеризації GCN з k = 2.
2. Далі досліджуються межі кластеризованого зображення та визначається кластер, який з'являється на більш ніж двох межах, як фоновий, тоді як інший кластер стає основним об'єктом.
3. Потім треба нанести обмежувальну рамку навколо визначеного основного об'єкта.

Сегментація об'єктів

Сегментація об'єктів - це завдання, яке полягає в тому, щоб розділити зображення на окремі об'єкти. Один із способів зробити це - це сегментація переднього і заднього планів. Цей метод ділить зображення на два класи: об'єкт на передньому плані та фон. Метод сегментації об'єктів, описаний у тексті, ділить зображення на два класи: об'єкт на передньому плані та фон. Потім він застосовує обмежувальну рамку до об'єкта на передньому плані, щоб покращити точність сегментації. Семантична сегментація DeepCut досягає семантичної сегментації за допомогою парадигми оптимізації під час тестування. Це означає, що модель навчається на зображенні, яке вона сегментує. Це усуває необхідність у спільній сегментації або етапах

постобробки. Семантична сегментація DeepCut має ряд переваг порівняно з іншими методами. Вона не потребує попереднього навчання на всіх зображеннях, що робить її більш ефективною. Крім того, вона не покладається виключно на кореляції, що дозволяє їй отримувати більш точні результати. Процес сегментації DeepCut складається з двох етапів:

- Сегментація переднього і заднього планів (k=2);
- Сегментація семантичної частини на об'єкті переднього плану (k=4).

Двоетапний процес усуває зсув функцій кластеризації в бік більших кластерів (наприклад, фон-передній план), що обмежує рівень деталізації сегментації об'єктів на передньому плані. Цей самий процес також можна застосувати для покращення сегментації фону.

1.2.3. **Мережа Кохонена**

Карта, що самоорганізується, була вперше розроблена в 1982 році фінським професором і дослідником доктором Теуво Кохоненом і є моделлю некерованого навчання, призначеною для додатків, в яких підтримка топології між вхідним і вихідним просторами має важливе значення. SOM починається з випадкового розподілу вузлів у двовимірному просторі. Потім, поступово, кожному вузлу присвоюється значення, яке представляє його позицію у високовимірному просторі. Це відбувається шляхом навчання SOM на наборі даних високовимірних векторів.

SOM підтримує топологію між вхідним і вихідним просторами, оскільки вузли, які є близькими у високовимірному просторі, також відображаються на сусідні вузли у двовимірному просторі. Це означає, що близькі точки в багатовимірному просторі будуть відображені на близькі точки у двовимірному просторі. Це корисно, коли потрібно візуалізувати складні багатовимірні дані. SOM також можна використовувати для зменшення розмірності, оскільки він відображає вхідні дані високої розмірності у низьковимірне (зазвичай двомірне) дискретизоване

представлення. Це корисно, коли потрібно зберегти основну структуру вхідного простору, але потрібно працювати з меншими даними.

SOM має широкий спектр застосувань, включаючи візуалізацію даних, класифікацію даних та розпізнавання образів. SOM можна використовувати для візуалізації даних зображення. Наприклад, можна використовувати SOM для відображення кольорів або текстур на зображенні. Це може бути корисно для розуміння структури зображення або для виявлення аномалій[31].

SOM також можна використовувати для класифікації даних зображення. Наприклад, можна використовувати SOM для класифікації зображень людей і тварин. Це може бути корисно для розпізнавання об'єктів на зображеннях або для створення систем безпеки.

Вузли сітки SOM пов'язані безпосередньо з вхідним вектором, але не один з одним. Це означає, що вузли не знають значень своїх сусідів, а лише оновлюють вагу своїх зв'язків як функцію від вхідних даних.

SOM організується на кожній ітерації як функція вхідних даних[18]. Це означає, що вузли сітки SOM поступово групуються таким чином, щоб вузли, які отримують схожі вхідні вектори, розташовувалися поблизу один одного. Ці відмінності в архітектурі та алгоритмічній поведінці роблять SOM корисним для різних завдань, таких як візуалізація даних, класифікація та кластеризація.

Таким чином, після кластеризації кожен вузол має власну (i, j) координату, що дозволяє обчислити евклідову відстань між 2 вузлами за допомогою теореми Піфагора.

При конкурентному навчанні на кожній ітерації активується лише один вузол, який є найкращим відповідником (BMU) для вхідних даних. BMU вибирається на основі подібності між вхідними даними та всіма вузлами в сітці. Вузол з найменшою евклідовою різницею між вхідними даними та всіма вузлами вибирається разом з сусідніми вузлами в певному радіусі. Ваги сусідніх вузлів трохи коригуються для відповідності вхідному вектору[23].

Цей процес повторюється для всіх вузлів у сітці. Зрештою, вся сітка збігається з повним набором вхідних даних, причому схожі вузли згруповані в одну область, а несхожі – рознесені.

Це дозволяє SOM зберігати топологію вхідного простору, що робить її корисною для таких завдань, як візуалізація даних та класифікаціях[15].

Змінні

t - поточна ітерація

n - межа ітерації, тобто загальна кількість ітерацій, яку може пройти мережа

λ - постійна часу, яка використовується для зменшення радіуса та швидкості навчання

i - координата рядка сітки вузлів

j - координата стовпчика сітки вузлів

d - відстань між вузлом та BMU

w - вектор ваги

$w_{ij}(t)$ - вага зв'язку між вузлами i,j у сітці та екземпляром вхідного вектора на ітерації t

x - вхідний вектор

x(t) - екземпляр вхідного вектора на ітерації t

α(t) - швидкість навчання, яка зменшується з часом в інтервалі [0,1], щоб забезпечити збіжність мережі.

$\beta_{ij}(t)$ - функція сусідства, яка монотонно спадає і представляє відстань вузла $i,\ j$ від BMU та вплив, який він має на навчання на кроці t.

$\sigma(t)$ - радіус функції сусідства, який визначає, наскільки далеко розглядаються сусідні вузли у 2D сітці при оновленні векторів. Він поступово зменшується з часом.

Алгоритм

- Треба ініціалізувати вагу кожної вершини w_{ij} випадковою величиною.
- Обирається випадковий вхідний вектор x_k.

Потрібно повторити пункти 4 і 5 для всіх вершин на карті:

- Обчислити евклідову відстань між вхідним вектором x(t) та вектором ваги w_{ij}, пов'язаним з першою вершиною, де $t,\ i,\ j\ =\ 0$.
- Відстежити вершину, яка дає найменшу відстань t.
- Знайти загальну найкращу відповідність (BMU), тобто вершину з найменшою відстанню від усіх обчислених.
- Визначити топологічну околицю $\beta_{ij}(t)$ та її радіус $\sigma(t)$ BMU на карті Кохонена
- Повторювати для всіх вершин в околі BMU: Оновлювати вектор ваги w_{ij} першого вузла в околиці BMU, додаючи частину різниці між вхідним вектором $x(t)$ та вагою $w(t)$ нейрона.
- Повторювати всю цю ітерацію до тих пір, поки не буде досягнуто обраної межі ітерації $t\ =\ n$

  Крок 1 - це фаза ініціалізації, а кроки 2-8 - фаза навчання.

Формули

Оновлення та зміни змінних відбуваються за наступними формулами:

Ваги в межах околиці оновлюються за формулою 1.22:

$$w_{ij}(t\ +\ 1) = w_{ij}(t) + a_i(t)\left[x(t) - w_{ij}(t)\right],\ \ or$$

$$w_{ij}(t\ +\ 1) = w_{ij}(t) + a_i(t)\beta_{ij}(t)[x(t) - w_{ij}(t)] \qquad (1.22)$$

Перше рівняння показує, що нова оновлена вага w_{ij}(t + 1) для вузла i, j дорівнює сумі старої ваги $w_{ij}(t)$ і частини різниці між старою вагою і вхідним вектором x(t). Іншими словами, ваговий вектор "зсувається" ближче до вхідного вектора. Ще одним важливим елементом є те, що оновлена вага буде пропорційною двовимірній відстані між вузлами в радіусі околиці та BMU.

Крім того, те саме рівняння 1.22 не враховує вплив навчання, яке є пропорційним до відстані між вершиною та BMU. Оновлена вага повинна враховувати той фактор, що ефект навчання близький до нуля на крайніх

точках околиці, оскільки кількість навчання повинна зменшуватися з відстанню. Тому друге рівняння додає додатковий коефіцієнт функції околиці $\beta_{ij}(t)$ і є більш точним поглибленим.

Радіус і швидкість навчання з часом зменшуються як аналогічно, так і експоненціально.

$$\sigma(t) = \sigma_0 * \exp exp\left(\frac{-t}{\lambda}\right), \text{ де } t = 1, 2, 3 \ldots n$$

$$a(t) = a_0 * \exp exp\left(\frac{-t}{\lambda}\right), \quad \text{де } t = 1, 2, 3 \ldots n \quad (1.23)$$

Вплив функції околиці $B_{ij}(t)$ обчислюється за формулою:

$$B_{ij}(t) = \exp exp\left(-\frac{d^2}{2\sigma^2(t)}\right), \text{ де } t = 1, 2, 3, \ldots, n \quad (1.24)$$

Евклідова відстань між вектором ваги вершини та поточним вхідним екземпляром обчислюється як сума квадратів відстаней від кожної координати вектора ваги до відповідної координати вхідного екземпляра[20].

$$\left\|\vec{x} - \vec{w}_{ij}\right\| = \sqrt{\sum_{t=0}^{n} \vec{x}(t) - \vec{w}_{ij}(t)} \quad (1.25)$$

З усіх розрахованих відстаней до вузла вибирається BMU з найменшою відстанню.

$$d = \left(\left\|\vec{x} - \vec{w}_{ij}\right\|\right) = min(\sqrt{\sum_{t=0}^{n} [\vec{x}(t) - \vec{w}_{ij}(t)]^2}) \quad (1.26)$$

1.2.4. **MEA (Multi-layer Evolving Architecture)**

MEA - це нейронна мережа, яка використовується для кластеризації даних. Вона заснована на самоорганізованих картах (SOM), але використовує локальне навчання, а не глобальне.

Локальне навчання означає, що ваги нейронів оновлюються лише в межах невеликого локального вікна навколо нейрона, який найкраще відповідає поточному даному. Це робить MEA більш ефективним, ніж інші алгоритми SOM, які використовують глобальне навчання, при якому ваги нейронів оновлюються для всієї карти.

На першому кроці алгоритму ваги нейронів ініціалізуються випадковим чином. Це можна зробити, наприклад, розподіливши ваги нейронів рівномірно в деякому діапазоні значень.

- Обробка точок даних

На кожному кроці алгоритму обробляється одна точка даних x. Для цієї точки даних алгоритм виконує наступні кроки:

Знаходження найближчого нейрона y:

$$y = argmin_i\, d\left(x,\, w_i(t)\right) \quad (1.27)$$

де $d(x,\, w_i(t))$ - це відстань між точкою даних x і нейроном $w_i(t)$ в момент часу t.

Оновлення ваг нейрона y і всіх його сусідів:

$$w_i(t\, +\, 1) = w_i(t) +\, a\, *\, h(d(x, y))\, *\, \left(x\, -\, w_i(t)\right) \quad (1.28)$$

де $w_i(t\, +\, 1)$- це вага нейрона $w_i(t)$ в момент часу t + 1, α - це коефіцієнт навчання, h(d(x, y)) - це функція спаду, а x - це точка даних, яка обробляється[3].

Функція спаду h(d(x, y)) зменшує вплив відстані між нейроном і точкою даних на величину оновлення ваги. Зазвичай використовується наступна функція спаду:

$$h(d(x,\, y)) =\, 1/(1\, +\, d\left(x,\, y\right)^2\Big) \quad (1.29)$$

Зменшення коефіцієнта навчання:

$$\alpha\, =\, \alpha\, /\, (1\, +\, t\, /\, T)$$

де T - це максимальне число ітерацій алгоритму.

Алгоритм зупиняється, коли досягнуто заданого числа ітерацій або коли ваги нейронів перестають значно змінюватися.

Додаткові деталі:

- MEA може використовуватися з будь-якою формою карти, але квадратна або прямокутна форма є найбільш популярною.
- Розмір карти повинен бути достатнім для того, щоб утворити кластери для всіх точок даних, але не надто великим, щоб не знижувати ефективність алгоритму.
- Кількість нейронів на карті повинна бути достатньою для того, щоб утворити кластери для всіх точок даних, але не надто великою, щоб не ускладнювати формування кластерів.
- Коефіцієнт навчання визначає, як швидко ваги нейронів оновлюються. Зазвичай використовується значення $\alpha = 0{,}1$.
- Функція спаду визначає, як швидко ваги нейронів оновлюються з часом. Зазвичай використовується функція спаду експоненціального спаду.:

$$h(d(x,\ y))\ =\ 1\ /\ (1\ +\ d\left(x,\ y\right)^{2}\Big) \qquad (1.30)$$

Щоб використовувати алгоритм MEA для кластеризації зображень, спочатку необхідно перетворити зображення в набори чисел, які характеризують його частотні компоненти. Це можна зробити, наприклад, за допомогою методу DCT.

Метод DCT перетворює зображення з просторової форми в частотну форму. Це дозволяє представити зображення як набір чисел, які характеризують його частотний спектр.

- Навчання алгоритму MEA

Форма карти визначає, як нейрони на карті будуть розташовані один щодо одного. Кількість нейронів на карті визначає, скільки кластерів буде

створено. Коефіцієнт навчання визначає, наскільки швидко ваги нейронів будуть змінюватися під час навчання[34]. Функція спаду визначає, як швидко ваги нейронів будуть змінюватися з часом. Алгоритм MEA навчається за допомогою наступного алгоритму:

Для кожної точки даних:

- Треба знайти нейрон, який найбільше наближений до цієї точки даних.
- Оновити ваги цього нейрона та всіх його сусідів.
- Цей алгоритм повторюється до тих пір, поки не буде досягнуто заданого числа ітерацій або поки ваги нейронів перестануть значно змінюватися.

- Кластеризація нових зображень

Після навчання алгоритм MEA можна використовувати для кластеризації нових зображень[19]. Для цього нові зображення також повинні бути перетворені в набори чисел.

Потім алгоритм MEA може бути використаний для визначення того, до якого кластера належить кожне нове зображення.

Алгоритм MEA має ряд переваг перед іншими алгоритмами кластеризації зображень. Нижче представленні деякі з них:

- Ефективність: Алгоритм MEA є ефективним, оскільки він використовує локальне навчання. Це означає, що ваги нейронів оновлюються лише в межах невеликого локального вікна навколо нейрона, який найбільше наближений до поточної точки даних.
- Універсальність: Алгоритм MEA може використовуватися для кластеризації різних типів зображень.
- Можливість кластеризації даних високої розмірності: Алгоритм MEA може використовуватися для кластеризації даних високої розмірності.

Недоліки використання алгоритму MEA для кластеризації зображень:

- Чуттєвість до початкових значень ваги нейронів: Алгоритм MEA може бути чутливим до початкових значень ваги нейронів. Це означає, що результати кластеризації можуть залежати від того, як ваги нейронів ініціалізуються.
- Чуттєвість до розміру локального вікна: Результати кластеризації можуть також залежати від розміру локального вікна.

Практичні поради[4]:

- Для кластеризації даних високої розмірності рекомендується використовувати невелику форму карти і невелику кількість нейронів.
- Для кластеризації даних з нерівномірним розподілом рекомендується використовувати функцію спаду з більш високою швидкістю зближення.

Розширення алгоритму

Алгоритм MEA можна розширити, додавши додаткові шари нейронів або адаптивні функції спаду.

Додаткові шари нейронів можуть допомогти поліпшити якість кластеризації, додавши більше складності моделі. Адаптивна функція спаду може допомогти алгоритму MEA швидше знайти оптимальне рішення, адаптуючись до змін у даних.

1.3. Висновок до розділу

Розвиток науки про дані обумовив появу різних підходів до кластеризації даних. Їх можна умовно розділити на традиційні та нейромережеві. Класичною нейронною мережею для кластеризації є мережа Кохонена. Ця архітектура є простою, за правильної побудови здатна вчитися протягом існування, але має недоліки при масштабуванні та афінних перетвореннях кластерів. На противагу їй існує кілька графових нейронних мереж кластеризації. Вони використовують механізм обміну повідомленнями та механізм машинної уваги під час роботи. Але їхня точність та швидкість залежить від вхідних даних, тож треба вивчити теоретичні відомості про

складність алгоритмів з метою визначення кращої архітектури для конкретного набору даних та конкретного пристрою, де експлуатується нейронна мережа.

РОЗДІЛ 2
МЕТРИКИ ОЦІНКИ ЯКОСТІ АЛГОРИТМІВ ТА АНАЛІЗ ЧАСОВОЇ Й ПРОСТОРОВОЇ СКЛАДНОСТІ ОПИСАНИХ АЛГОРИТМІВ

Аналіз часової та просторової складності алгоритму є важливим для оцінки його ефективності та придатності для конкретних завдань.

Часова складність алгоритму визначає кількість часу, необхідного для його виконання. Вона може бути описана у вигляді асимптотичної функції, яка залежить від розміру вхідних даних.

Просторова складність алгоритму визначає кількість пам'яті, яка необхідна для його виконання. Вона також може бути описана у вигляді асимптотичної функції, яка залежить від розміру вхідних даних. Метрики якості результату алгоритму дозволяють оцінити його відповідність вимогам користувача.

1.1. Оцінка просторової та часової складності описаних алгоритмів

1.1.1. Аналіз часової та просторової складності алгоритму Hi-LANDER

Часова Складність:

Одиночна ітерація алгоритму:

- Операції на ребрах: O (кількість ребер) - лінійна залежність від кількості ребер у графі.
- Операції на вершинах: O(кількість вершин) - лінійна залежність від кількості вершин у графі.
- Агрегація та декодування: O(кількість вершин) - лінійна залежність від кількості вершин у графі.

Ітерації на рівні ієрархії:

- Глибина ієрархії: O(глибина ієрархії) - лінійна залежність від глибини ієрархії.
- Загальна часова складність: O(кількість рівнів ієрархії * (кількість ребер + кількість вершин))

Просторова Складність:

Одиночна ітерація алгоритму:

- Часова складність алгоритму збереження ознак вершин залежить від кількості вершин у графі. Вона є лінійною, тобто зростає прямо пропорційно до кількості вершин.
- Часова складність алгоритму ітерацій на рівні ієрархії залежить від кількості вершин і ребер у кожному рівні ієрархії. Вона також є лінійною, тобто зростає прямо пропорційно до загальної кількості вершин і ребер у всіх рівнях ієрархії.

Ці оцінки часової та просторової складності є асимптотичними, тобто вони дають загальний огляд відносних обчислювальних витрат алгоритму. Вони не враховують конкретних оптимізацій чи деталей реалізації, які можуть впливати на реальні значення.

2.1.2. Аналіз часової та просторової складності алгоритму DeepCut

Часова складність:

- Створення зображення та отримання глибинних ознак: $O(m * n * d)$
- Створення зваженого графа: $O((\frac{mn}{p^2})\text{^}2)$
- Оптимізація GNN та функції втрат: Залежить від розмірності вбудовування токенів та кількості шарів у GNN.

Просторова складність:

- Створення зображення та отримання глибинних ознак: O(1)

- Створення зваженого графа: $O(\left(\frac{mn}{p^2}\right)^2)$
- Оптимізація GNN та функції втрат: Залежить від розмірності вбудовування токенів та кількості шарів у GNN.

До переваг цього метода можна віднести такі:

- Універсальність: метод може застосовуватися до різних завдань, але це може ускладнити його налаштування та вимоги до ресурсів.
- Спрощення: метод не вимагає постобробки, що може бути корисним у деяких випадках. Однак, це може залежати від вхідних даних та конкретної задачі.
- Точність та швидкодія: метод має високу продуктивність у порівнянні з багатьма іншими методами. Однак, важливо оцінити вплив великих розмірів зображення на швидкодію.

Але метод має і певні недоліки такі як:

- Кореляційна кластеризація з від'ємними вагами - це нетиповий підхід, який може бути менш ефективним або вимагати додаткових обчислень.
- Використання графових нейронних мереж (GNN) може збільшити складність обчислень та навчання[5].
- Етапи локалізації та сегментації у двох етапах можуть призвести до додаткової обчислювальної та часової складності.

2.1.3. Аналіз часової та просторової складності Карт Кохонена

Часова складність:

- Ініціалізація: Часова складність ініціалізації SOM зазвичай становить $O(d * m * n)$, де d - кількість вхідних ознак, а m та n - розмірність сітки МНК.

SOM - це метод, який використовується для обчислення центрів кластерів. Він передбачає, що центри кластерів знаходяться в точках, де функція подібності між точками і кластерами досягає максимуму[36].

- Етап навчання: Часова складність фази навчання часто виражається через кількість ітерацій (T), кількістю точок даних (N) та розмірами сітки SOM (m*n)[7]. Типова часова складність становить O(T * N * m * n * d), де d - кількість вхідних ознак.

Просторова складність:

- Вагові вектори: Просторова складність зберігання вагових векторів, пов'язаних з кожним вузлом у сітці SOM, становить O(m * n * d), де m та n - розміри сітки, а d - кількість вхідних ознак.
- Вхідні дані: Складність простору для зберігання вхідних даних становить O(N * d), де N - кількість точок даних, а d - кількість вхідних ознак.
- Функція сусідства: Деякі реалізації можуть потребувати додаткового простору для зберігання інформації, пов'язаної з функцією сусідства, під час навчання.

Практичні міркування:

- Час виконання етапу навчання SOM залежить від кількості ітерацій, критеріїв збіжності та розміру сітки SOM.
- Розмір сітки SOM (m∗n) та графік швидкості навчання можуть впливати на часову та просторову складність.
- SOM можуть бути чутливими до ініціалізації вагових векторів та параметрів, таких як швидкість навчання.
- SOM можуть бути ефективними для певних типів даних, але можуть погано масштабуватися для дуже великих наборів даних або даних високої розмірності.

Важливо пам'ятати, що навчання SOM може бути обчислювально дорогим, особливо для великих наборів даних або великих сіток. Вибір параметрів, таких як швидкість навчання та функція сусідства, також може впливати на продуктивність алгоритму. Як і для будь-якого алгоритму кластеризації,

рекомендується експериментувати з різними налаштуваннями та оцінювати поведінку алгоритму на конкретних наборах даних.

2.1.4. Аналіз часової та просторової складності алгоритма MEA

Часова складність алгоритму MEA залежить від наступних факторів:

- Часова складність алгоритму MEA пропорційна кількості точок даних, які необхідно кластеризувати.
- Часова складність алгоритму MEA пропорційна розміру карти.
- Часова складність алгоритму MEA пропорційна числу ітерацій, які виконує алгоритм.

Загальна оцінка часової складності алгоритму MEA може бути представлена наступним чином:

$$O(n * m * t)$$

де:

n - кількість точок даних

m - розмір карти

t - число ітерацій

Просторова складність

Просторова складність алгоритму MEA залежить від наступних факторів:

- Просторова складність алгоритму MEA пропорційна кількості точок даних, які необхідно кластеризувати.
- Просторова складність алгоритму MEA пропорційна розміру карти.
- Загальна оцінка просторової складності алгоритму MEA може бути представлена наступним чином:

$$O(n * m)$$

де:

- n - кількість точок даних

- m - розмір карти

Таким чином, алгоритм MEA є ефективним алгоритмом кластеризації, оскільки його часова складність пропорційна кількості точок даних, які необхідно кластеризувати. Це означає, що алгоритм може масштабуватися для великих наборів даних.

2.2. Метрики оцінки якості алгоритмів кластеризації

Силует (Silhouette Score): це метрика, яка оцінює, наскільки об'єкти всередині одного кластера схожі один на одного, порівняно з об'єктами в інших кластерах. Коефіцієнт силуету або оцінка силуету - це метрика, яка використовується для оцінки ефективності та якості методу кластеризації. Його значення коливається від -1 до 1.

1: Означає, що кластери добре відокремлені один від одного і чітко розрізняються.

0: Означає, що кластери індиферентні, або можна сказати, що відстань між кластерами не є значущою.

-1: Кластери розподілені неправильно.

$$\text{Оцінка силуету} = (b - a)/max(a, b) \quad (2.1)$$

де

- a= середня внутрішньокластерна відстань, тобто середня відстань між кожною точкою всередині кластера.
- b= середня міжкластерна відстань, тобто середня відстань між усіма кластерами.

Індекс силового поля (Dunn Index): Індекс Данна (DI) (введений Дж. К. Данном у 1974 році), це метрика для оцінки ефективності алгоритмів кластеризації. Він вимірює, наскільки добре кластери розділені один від одного та наскільки компактними вони є.

Чим вище значення індексу Данна, тим кращою є кластеризація. Якщо значення індексу близьке до нуля, це означає, що кластери погано розділені один від одного. Якщо значення індексу близьке до нескінченності, це означає, що кластери є дуже компактними, але не розділені один від одного.

Чим вище значення індексу Данна, тим кращою є кластеризація. Кількість кластерів, яка максимізує індекс Данна, приймається за оптимальну кількість кластерів k. Вона також має певні недоліки. Зі збільшенням кількості кластерів та розмірності даних зростають і обчислювальні витрати.

Індекс Данна для с кількості кластерів визначається як:

$$Dunn\ Index(U) = min_{1\le i\le c}\{\left\{\frac{\delta(x_i,\ x_j)}{min_{1\le k\le c}\{\Delta(x_k)\}}\right\}\} \quad (2.2)$$

$\delta(x_i,\ x_j)$ між кластерна відстань, тобто відстань між кластером x_i, і x_j

$\Delta(x_k)$ внутрішньокластерна відстань кластера x_k, тобто відстань всередині кластера x_k.

DB-index: Індекс Девіса-Болдіна (DBI) (введений Девідом Л. Девісом і Дональдом В. Боулдіном у 1979 році), це метрика для оцінки ефективності алгоритмів кластеризації. Він вимірює, наскільки добре кластери розділені один від одного.

Чим менше значення індексу DBI, тим кращою є кластеризація. Однак, навіть якщо значення індексу DBI є низьким, це не гарантує найкращого пошуку інформації. Індекс Девіса-Болдуіна є внутрішньою схемою оцінки, тобто він базується лише на самих кластеризованих даних.

Індекс ДБ для k кластерів визначається як :

$$DB\ index(U) = \frac{1}{k}\sum_{i=1}^{k}\Delta\left(x_i\right) + \Delta(x_j)/\delta(x_i,\ x_j) \quad (2.3)$$

$\delta(x_i,\ x_j)$ міжкластерна відстань, тобто відстань між кластером x_i, і x_j

$\Delta(x_k)$ внутрішньокластерна відстань кластера x_k, тобто відстань всередині кластера x_k.

ARI (Adjusted Rand Index) - це метрика, яка оцінює схожість між кластеризаціями та істинними мітками даних. ARI приймає значення від -1 до 1, де високі значення вказують на хорошу схожість між істинною структурою і кластеризацією.

NMI (Normalized Mutual Information) - це метрика, яка оцінює схожість між кластеризацією та істинними мітками даних, враховуючи збіг інформації між ними. NMI приймає значення від 0 до 1, де 1 вказує на ідеальний збіг.

Формула розрахунку NMI:

$$NMI\ (\Omega,\ C) = \frac{I(\Omega;C)}{[H(\Omega)+H(C)]/2} \quad (2.4)$$

Purity: Purity є простим і прозорим мірилом оцінки. Нормалізована взаємна інформація може бути теоретично інтерпретована. Індекс Ренда карає як хибнопозитивні, так і хибнонегативні рішення під час кластеризації. Міра F додатково підтримує диференційоване зважування цих двох типів помилок.

Для обчислення purity кожному кластеру присвоюється клас, який найчастіше зустрічається в кластері, а потім вимірюється точність цього присвоєння шляхом підрахунку кількості правильно віднесених данних і ділення на N. Формально:

$$purity(\Omega, C) = \frac{1}{N}\sum_{k} max_{j}|\omega_{k} \cap C_{j}| \quad (2.5)$$

Adjusted Mutual Information (AMI): це метрика, яка вимірює ступінь взаємозалежності між двома кластеризаціями, при цьому враховуються випадкові відповіді. AMI приймає значення від 0 до 1, де 0 вказує на відсутність схожості, а 1 - на повну взаємозалежність.

Variation of Information (VI): це метрика, яка вимірює відстань між двома кластеризаціями, використовуючи концепції інформаційної теорії. Чим менше значення VI, тим більш схожі кластеризації. Однак, значення VI залежить від обраного розбиття ентропії.

Fowlkes-Mallows Index (FMI): це метрика, яка вимірює схожість між двома кластеризаціями, порівнюючи кількість пар точок у різних кластерах та однакових кластерах. FMI приймає значення від 0 до 1, де 1 вказує на повну схожість.

Jaccard Coefficient: це міра подібності між двома кластеризаціями, яка обчислюється як відношення кількості точок, що належать одному та тому ж кластеру, до загальної кількості точок у двох кластерах. Коефіцієнт Jaccard приймає значення від 0 до 1, де 1 вказує на повну схожість[28].

Pairwise F1 Score: це метрика, яка використовується для вимірювання точності при виявленні пар кластерів між двома кластеризаціями. Pairwise F1 Score приймає значення від 0 до 1, де 1 вказує на повну точність.

Hubert Index: це міра подібності між двома кластеризаціями, яка вимірює ступінь схожості між об'єктами, які належать одному кластеру в одній кластеризації, та об'єктами, які належать одному кластеру в іншій кластеризації. Hubert Index приймає значення від -1 до 1, де 1 вказує на повну схожість, а -1 - на повну розбіжність.

Cophenetic Correlation Coefficient: це метрика, яка використовується для оцінки ступеня, до якого дендрограма зберігає відстані між вихідними точками. Cophenetic Correlation Coefficient приймає значення від -1 до 1, де 1 вказує на повну збереженість відстаней, а -1 - на їх повну втрату.

2.3. Підготовка зображень до кластеризації

Для кластеризації зображень більшість алгоритмів вимагає попередньої обробки зображення перед кластеризацією. У цьому розділі будуть описані методи, які можна використовувати для підготовки зображення під кожен алгоритм кластеризації, описаний у розділі 1.

2.3.2. Підготовка зображень для Hi-LANDER

Для того, щоб використовувати алгоритм Hi-LANDER для кластеризації зображень, необхідно попередньо підготувати зображення. Підготовка зображень до кластеризації включає наступні етапи:

1. Перетворення зображень у числову форму

Для обробки зображень алгоритмом Hi-LANDER необхідно перетворити їх у числову форму. Числова форма зображення представляє його як набір чисел, які характеризують його частотні компоненти.

Для перетворення зображень у числову форму можна використовувати різні методи, такі як:

- Дискретно-косинусне перетворення (DCT) - це лінійний метод перетворення, який розкладає зображення на суму косинусних хвиль.
- Дискретно-вейвлет-перетворення (DWT) - це нелінійний метод перетворення, який розкладає зображення на суму вейвлет-функцій.
- Метод головних компонентів (PCA) - це статистичний метод, який виділяє найважливіші компоненти зображення.

2. Відбір ознак

Після того, як зображення представлені у числовій формі, необхідно вибрати ознаки, які будуть використовуватися для кластеризації. Ознаки повинні бути достатньо інформативними, щоб дозволити алгоритму Hi-LANDER правильно кластеризувати зображення.

Однією з найпоширеніших методик відбору ознак є метод головних компонент (PCA).

3. Нормування ознак

Нормування ознак дозволяє порівнювати ознаки з різних зображень. Для нормування ознак можна використовувати різні методи, такі як стандартизація, центрування або нормування до одиниці.

4. Визначення параметрів алгоритму

Алгоритм Hi-LANDER має ряд параметрів, які необхідно визначити перед його використанням[24]. До цих параметрів відносяться:

- Розмір кластера визначає, скільки точок даних буде входити до одного кластера.
- Параметр щільності визначає, наскільки щільно розташовані точки даних в кластері.
- Кількість ітерацій визначає, скільки разів алгоритм Hi-LANDER буде переглядати зображення.

Параметри алгоритму Hi-LANDER зазвичай вибираються експериментально.

2.3.3. Підготовка зображень для алгоритму DeepCut

Алгоритм DeepCut - це алгоритмом сегментації зображень, заснований на методі машинного навчання. Алгоритм DeepCut працює, навчаючись на наборі прикладів, в яких кожне зображення розділено на сегменти.

Для використання алгоритму DeepCut для сегментації зображень зображення необхідно попередньо обробити.. Підготовка зображень до сегментації за допомогою алгоритму DeepCut включає наступні етапи:

1. Вирівнювання зображень

Вирівнювання зображень є важливим кроком, оскільки воно дозволяє алгоритму DeepCut краще вчитися на наборі прикладів. Вирівнювання зображень можна виконати за допомогою таких методів:

- Центрування: Цей метод полягає в тому, що центри всіх зображень в наборі прикладів переносяться в одну точку.
- Нормування: Цей метод полягає в тому, що всі зображення в наборі прикладів масштабуються до одного розміру.
- Корекція освітленості: Цей метод полягає в тому, що всі зображення в наборі прикладів приводяться до одного рівня освітленості.

2. Відбір ознак

Після того, як зображення вирівняні, необхідно вибрати ознаки, які будуть використовуватися для сегментації. Ознаки повинні бути достатньо інформативними, щоб алгоритм DeepCut міг правильно сегментувати зображення[32].

Однією з найпоширеніших методик відбору ознак є метод головних компонент (PCA).

.

3. Нормування ознак

Нормування ознак дозволяє порівняти ознаки з різних зображень, використовуючи єдину шкалу. Для цього можна використовувати різні методи, такі як стандартизація, центрування або нормування до одиниці.

4. Визначення параметрів алгоритму

Алгоритм DeepCut має ряд параметрів, які необхідно визначити перед його використанням. До цих параметрів відносяться:

- Розмір кластера визначає, скільки пікселів буде входити до одного сегменту.
- Параметр щільності визначає, наскільки щільно розташовані пікселі в кластері.
- Кількість ітерацій визначає, скільки разів алгоритм DeepCut буде переглядати зображення.

Параметри алгоритму DeepCut зазвичай вибираються експериментально.

5. Сегментація зображень

Після того, як зображення підготовлені, їх можна сегментувати за допомогою алгоритму DeepCut. Алгоритм DeepCut сегментує зображення, поступово об'єднуючи пікселі в кластери[17].

Нижче приведені рекомендації щодо підготовки зображень до сегментації за допомогою алгоритму DeepCut:

- Потрібно використати метод PCA для відбору ознак. Метод PCA дозволяє вибрати такі ознаки, які найкраще відображають основні характеристики зображень.
- Нормувати ознаки методом стандартизації. Нормування ознак дозволяє порівнювати ознаки з різних зображень.

Для підготовки зображень для використання алгоритму DeepCut рекомендується дотримуватися наступних рекомендацій:

- Використовувати зображення розміром не менше 256x256 пікселів. Це дозволить алгоритму DeepCut краще навчитися на наборі прикладів.
- Використовувати зображення з високою роздільною здатністю. Це дозволить алгоритму DeepCut краще розпізнавати межі сегментів.
- Використовувати зображення з рівномірною освітленістю. Це дозволить алгоритму DeepCut краще розпізнавати об'єкти на зображенні.
- Використовувати зображення без шуму. Шум може негативно вплинути на точність сегментації.

При дотриманні цих рекомендацій буде можливо отримати кращі результати сегментації зображень за допомогою алгоритму DeepCut.

2.3.4. Підготовка зображень для алгоритму Карти Кохонена

Підготовка зображень до сегментації є важливим етапом процесу сегментації зображень. Важливо, щоб зображення були належним чином відрегульовані, щоб алгоритм сегментації міг правильно сегментувати їх. Непідготовлені зображення можуть призвести до наступних проблем:

- Неточна сегментація
- Повільне навчання алгоритму
- Нестабільні результати сегментації

Етапи підготовки зображень

Підготовка зображень до сегментації за допомогою алгоритму Карти Кохонена включає наступні етапи:

1. Вирівнювання зображень

Вирівнювання зображень є важливим кроком, оскільки воно дозволяє алгоритму краще навчитися на наборі прикладів. Вирівнювання зображень можна виконати за допомогою таких методів:

- Центрування полягає в тому, що центри всіх зображень в наборі прикладів розташовуються в одній точці. Це можна зробити, вирахувавши середні значення x і y для всіх пікселів кожного зображення. Потім, для кожного зображення, можна відняти середні значення з кожної точки зображення.
- Нормування полягає в тому, що всі зображення в наборі прикладів масштабуються до однакових розмірів. Це можна зробити, використовуючи коефіцієнти масштабування, які можна обчислити, вимірявши ширину і висоту кожного зображення.
- Корекція освітленості полягає в тому, що всі зображення в наборі прикладів приводяться до одного рівня освітленості. Це можна зробити, використовуючи функцію для коригування яскравості і контрастності кожного зображення.

2. Відбір ознак

Ознаки - це характеристики зображень, які використовуються алгоритмом Карти Кохонена для визначення кластерів. Ознаки повинні бути достатньо інформативними для того, щоб дозволити алгоритму Карти Кохонена правильно сегментувати зображення.

Однією з найпоширеніших методик відбору ознак є метод головних компонент (PCA). Метод PCA дозволяє вибрати ознаки, які найкраще відображають основні характеристики зображень. Цей метод є одним з найпоширеніших методів відбору ознак для алгоритму Карти Кохонена.

3. Нормування ознак

Нормування ознак дозволяє порівнювати ознаки з різних зображень, які можуть мати різні масштаби, значення і діапазони. Для нормування ознак можна використовувати різні методи, такі як стандартизація, центрування або нормування до одиниці.

4. Визначення параметрів алгоритму

Алгоритм Карти Кохонена[22] має ряд параметрів, які необхідно визначити перед його використанням. До цих параметрів відносяться:

- Розмір карти визначає, скільки кластерів буде створено алгоритмом.
- Швидкість навчання визначає, як швидко алгоритм буде навчатися.
- Кількість ітерацій визначає, скільки разів алгоритм буде виконувати один цикл навчання.

Параметри алгоритму Карти Кохонена зазвичай вибираються експериментально.

5. Сегментація зображень

Після того, як зображення підготовлені, їх можна сегментувати за допомогою алгоритму Карти Кохонена. Алгоритм Карти Кохонена сегментує зображення, поступово розподіляючи пікселі зображень між кластерами.

Нижче приведені рекомендації щодо підготовки зображень до сегментації за допомогою алгоритму Карти Кохонена[14]:

- Нормувати ознаки методом стандартизації. Це дозволяє порівнювати ознаки з різних зображень.

- Визначати параметри алгоритму експериментально. Параметри алгоритму Карти Кохонена зазвичай вибираються експериментально.

Нижче приведені деякі специфічні рекомендації щодо підготовки зображень для використання алгоритму Карти Кохонена:

- Використовувати зображення розміром не менше 256x256 пікселів. Це дозволить алгоритму Карти Кохонена краще навчитися на наборі прикладів.
- Використовувати зображення з високою роздільною здатністю. Це дозволить алгоритму Карти Кохонена краще розпізнавати межі сегментів.

2.3.5. Підготовка зображень для алгоритму MEA

Загалом підготовка зображень для алгоритму MEA мало відрізняється від підготовки для Карт Кохонена[16], окрім таких моментів як:

1. Розбиття зображень на рівні

Розбиття зображень на рівні дозволяє алгоритму MEA краще розпізнавати межі сегментів. Зображення можна розбити на рівні за допомогою таких методів, як:

- Квадратична декомпозиція полягає в тому, що зображення розбивається на квадратні блоки. Цей метод є простим і ефективним, але він може не бути оптимальним для зображень з високою роздільною здатністю.
- Фільтр Гаусса[9] полягає в тому, що зображення розбивається на блоки, які мають приблизно однаковий рівень шуму. Цей метод є більш складним, ніж квадратична декомпозиція, але він може забезпечити більш точне розбиття для зображень з високою роздільною здатністю.

2. Обчислення щільності пікселів

Щільність пікселів - це характеристика зображення, яка визначає, скільки пікселів знаходиться в певній області. Щільність пікселів можна обчислити за допомогою таких методів, як:

- Обернена відстань полягає в тому, що щільність пікселів визначається оберненою відстанню від пікселя до найближчого кордону сегмента. Цей метод є простим і ефективним, але він може не бути точним для зображень з високою роздільною здатністю[6].
- Інтеграл щільності полягає в тому, що щільність пікселів визначається шляхом інтегрування функції щільності розподілу пікселів. Цей метод є більш складним, ніж обернена відстань, але він може забезпечити більш точну оцінку щільності для зображень з високою роздільною здатністю.

3. Сегментація зображень

Після того, як зображення підготовлені, їх можна сегментувати за допомогою алгоритму MEA. Алгоритм MEA сегментує зображення, поступово об'єднуючи блоки з однаковою щільністю.

Нижче приведені рекомендації щодо підготовки зображень до сегментації за допомогою алгоритму MEA[37]:

- Використовувати метод квадратичної декомпозиції для розбиття зображень на рівні. Цей метод є ефективним для більшості типів зображень.
- Використовувати метод оберненої відстані для обчислення щільності пікселів. Цей метод є простим у реалізації і забезпечує точні результати[25].

Параметри алгоритму MEA зазвичай вибираються експериментально.

Нижче приведені додаткові рекомендації щодо підготовки зображень для використання алгоритму MEA:

- Використовувати зображення розміром не менше 256x256 пікселів. Це дозволить алгоритму MEA краще розпізнавати межі між сегментами.
- Використовувати зображення з високою роздільною здатністю. Це дозволить алгоритму MEA краще розпізнавати межі між сегментами.
- Використовувати зображення без шуму. Шум може негативно вплинути на точність сегментації.

2.4. **Висновок до розділу**

Час роботи мережі Кохонена має бути меншим, тож у випадку, якщо не потрібно масштабувати або повертати кластери, суто теоретично її використання виглядає бажанішим. Однак у разі наявності відмінностей між схожими кластерами, ефективнішим має бути використання графових нейронних мереж кластеризації, теоретично найповільнішою має бути MEA, однак реальну швидкість та точність треба перевіряти на певних вхідних даних.

РОЗДІЛ 3 ПОРІВНЯННЯ АРХІТЕКТУР НЕЙРОМЕРЕЖ КЛАСТЕРИЗАЦІЇ В УМОВАХ ОБМЕЖЕНИХ ОБЧИСЛЮВАЛЬНИХ ПОТУЖНОСТЕЙ З МЕТОЮ ПОБУДОВИ ВІДПОВІДНОЇ ІТ

3.1. Умови експерименту

Як зазначалося вище, експлуатація нейронних мереж на звичайних мобільних пристроях без спеціальних вдосконалень є вкрай повільною. Повільним є процес прогнозування, а процес навчання на більш менш великому наборі даних є взагалі неприйнятно довгим. Це пов’язано з чималою кількістю обчислень дійсних чисел. Цю проблему на стаціонарних комп’ютерах зазвичай розв’язують за допомогою чипів відеокарт або спеціалізованих процесорів для дійсних обчислень. Однак з суто технічної точки зору, встановлення потужної відеокарти на портативний пристрій не є можливим через чималу електричну потужність, яку споживає відеокарта, площу плати, яку вона займає, кількість тепла, яку вона виділяє під час роботи тощо. До недавнього часу вжиток нейронних мереж загального призначення на звичайних побутових пристроях був доволі обмеженим. Виняток становили лише системи голосового керування або машинного зору, які дозволяли використання деяких нейромереж прямого розповсюдження та однієї єдиної моделі, навченої для розв’язання одного окремого різновиду задач.

Однак нещодавно ситуація на ринку профільної електроніки змінилася. З’явилися кілька одноплатних комп’ютерів з вбудованими контролерами для обробки нейромереж. Ці контролери мають апаратну реалізацію деяких типових шарів, що часто використовуються у нейромережевих архітектурах, зокрема у системах машинного зору. До числа таких одноплатних комп’ютерів належать Nvidia Jetson Nano, ATI Kria KV260 Vision AI та Texas BeagleBone AI-64. З наведених плат була обрана Texas BeagleBone через порівняно нижчу вартість. При цьому технічно цей пристрій не поступається

виробу компанії Nvidia та перевершує виріб ATI. У таблиці 3.1 наведені основні технічні характеристики цього пристрою.

Таблиця 3.1

Основні технічні характеристики Texas BeagleBone AI-64

Характеристика	Значення
Тип процесора	Dual 64-bit Arm Cortex-A72 microprocessor subsystem at up to 2.0 GHz
Тип обчислювача дійсних чисел	C7x floating point, vector DSP, up to 1.0 GHz, 80 GFLOPS, 256 GOPS[38]
Модуль штучного інтелекту	Deep-learning matrix multiply accelerator (MMA), up to 8 TOPS (8b) at 1.0 GHz [38] Vision Processing Accelerators (VPAC) with Image Signal Processor (ISP) and multiple vision assist accelerators Depth and Motion Processing Accelerators (DMPAC)
Додаткові засоби обробки дійсних чисел	Two C66x floating point DSP, up to 1.35 GHz, 40 GFLOPS, 160 GOPS[38]
GPU	3D GPU PowerVR Rogue 8XE GE8430, up to 750 MHz, 96 GFLOPS, 6 Gpix/sec

Наведені характеристики чітко вказують на те, що цей тип процесора не є досконалим і суттєво поступається в продуктивності традиційним потужним відеокартам. Однак слід зауважити, що у порівнянні з традиційними процесорами, у порівнянні з системою на базі традиційного мобільного процесору, цей пристрій є достатньо швидким аби нейронна мережа встигала опрацювати достатньо великий обсяг даних у псевдореальному часі. Тож можна говорити, що у порівнянні з традиційними системами призначеними для роботи штучних нейронних мереж, одноплатний комп’ютер Texas Instruments має обмежену продуктивність.

3.2. Реалізація DeepCut

Для перевірки можливостей архітектури DeepCut, була збудована нейронна мережа наступної архітектури, яка зображена на рисунку 3.1:

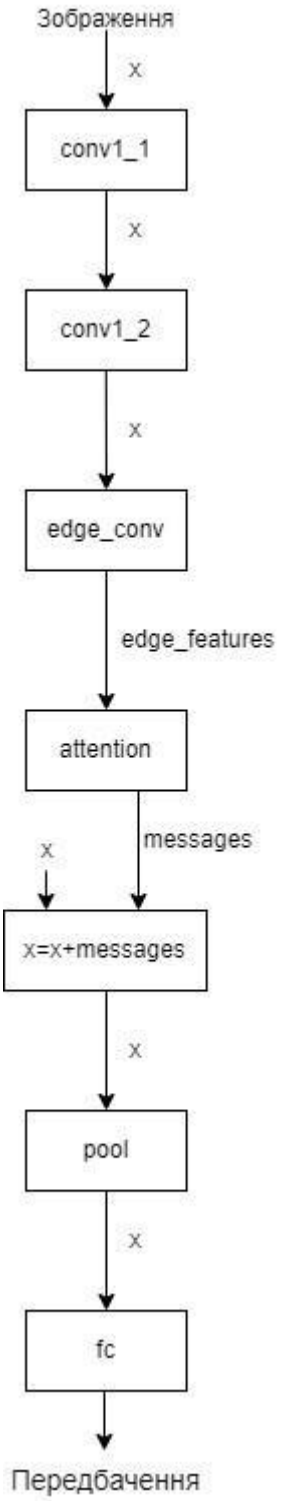

Рис. 3.1. Збудована архітектура DeepCut

Де,

conv1_1: Шар згортки 2D з 32 фільтрами розміром 3x3.

conv1_2: Шар згортки 2D з 32 фільтрами розміром 3x3.

edge_conv: Шар згортки 2D з 32 фільтрами розміром 1.

attention: Шар уваги.

pool: Шар агрегації повідомлень.

fc: Вихідний шар.

Передбачення: У якості передбачення мережа повертає вектор кластеризації.

Збудована архітектура має два згорткові шари з ядром 3x3 та кроком 1 та один шар з ядром 1 та кроком 1 для виділення властивостей зв’язків у графі. Після цього результати згортки попадають в шар attention, який формує повідомлення для оновлення вузлів графа. Результати оновлення графу подаються в шар агрегації повідомлень, а з нього потрапляють на повнозв’язний шар, який формує результати кластеризації.

Нижче наведений код збудованої архітектури, написаний мовою Python з використанням пакета PyTorch:

Лістинг 3.1:

```
import torch
import torch.nn as nn

class DeepCutGCN(nn.Module):

  def __init__(self, num_clusters):
    super(DeepCutGCN, self).__init__()

    # Представлення елементів графу
    self.conv1_1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
    self.conv1_2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)

    # Представлення зв'язків графу
    self.edge_conv = nn.Conv2d(32, 32, kernel_size=1, stride=1, padding=0)

    # Механізм оновлення повідомлень
```

```
        self.attention = nn.Softmax(dim=1)

        # Функція агрегації повідомлень
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Вихідний шар
        self.fc = nn.Linear(32 * 7 * 7, num_clusters)

    def forward(self, x, edge_index):
        x = F.relu(self.conv1_1(x))
        x = F.relu(self.conv1_2(x))

        # Обчислюємо представлення зв'язків
        edge_features = self.edge_conv(x)

        # Обчислюємо повідомлення
        messages = self.attention(edge_features) * x

        # Оновлюємо представлення елементів
        x = x + messages

        # Агрегуємо представлення елементів
        x = self.pool(x)

        # Виводимо значення ознак
        x = x.view(-1, 32 * 7 * 7)
        x = self.fc(x)

        return x
```

3.3. Реалізація карт Кохонена

Карти Кохонена не є мережею глибокого навчання. В ній використовується всього один повнозв’язний шар двовимірної матриці нейронів, як показано на рисунку 3.2:

Рис. 3.2. Збудована архітектура карт Кохонена

Де fc – повнозв’язний шар.

Нижче наведений код збудованої архітектури, написаний мовою Python з використанням пакета PyTorch:

Лістинг 3.2:

```
class KohonenMap(nn.Module):
    def __init__(self, num_clusters):
        super(KohonenMap, self).__init__()

        self.num_clusters = num_clusters

        self.weights = nn.Parameter(torch.randn(num_clusters, 28 * 28))

    def forward(self, x):
        x = x.view(1, -1)

        distances = (x - self.weights).pow(2).sum(-1)
        winner = distances.argmin(-1)

return winner
```

3.4. Реалізація Hi-LANDER

Для перевірки можливостей архітектури Hi-LANDER, була збудована нейронна мережа, архітектура якої зображена на рисунку 3.3:

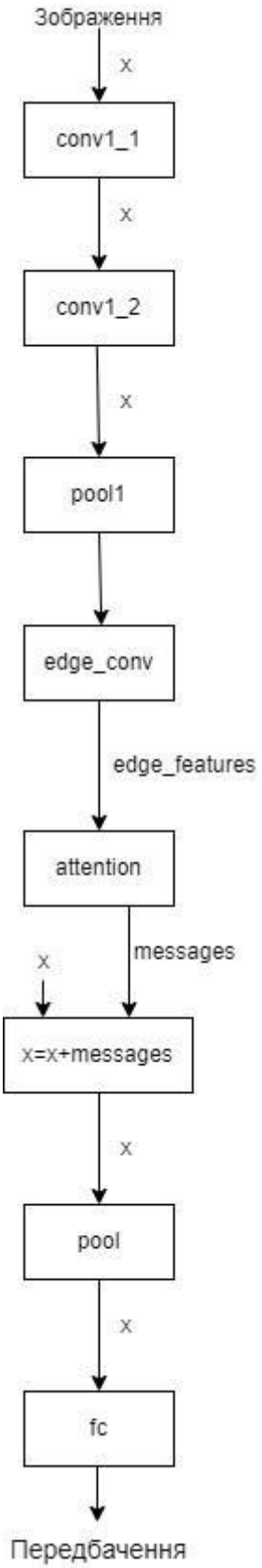

Рис. 3.3. Збудована архітектура HI-LANDER

Де,

conv1_1: Шар згортки 2D з 32 фільтрами розміром 3x3.

conv1_2: Шар згортки 2D з 32 фільтрами розміром 3x3.

pool1: Шар агрегації результатів згортки.

edge_conv: Шар згортки 2D з 32 фільтрами розміром 1.

attention: Шар уваги.

pool: Шар агрегації повідомлень.

fc: Вихідний шар.

Передбачення: У якості передбачення мережа повертає вектор кластеризації.

Збудована архітектура має два згорткові шари з ядром 3x3 та кроком 1, шар агрегації результатів згортки та один шар з ядром 1 та кроком 1 для виділення властивостей зв'язків у графі. Після цього результати згортки попадають в шар attention, який формує повідомлення для оновлення вузлів графа. Результати оновлення графу подаються в шар агрегації повідомлень, а з нього потрапляють на повнозв'язний шар, який формує результати кластеризації.

Нижче наведений код збудованої архітектури, написаний мовою Python з використанням пакета PyTorch:

Лістинг 3.3:

```
import torch
import torch.nn as nn

class HiLANDERGCN(nn.Module):

  def __init__(self, num_clusters):
    super(HiLANDERGCN, self).__init__()

    # Представлення елементів графу
    self.conv1_1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
```

```
        self.conv1_2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Представлення зв'язків графу
        self.edge_conv = nn.Conv2d(32, 32, kernel_size=1, stride=1, padding=0)

        # Механізм оновлення повідомлень
        self.attention = nn.Softmax(dim=1)

        # Функція агрегації повідомлень
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Вихідний шар
        self.fc = nn.Linear(32 * 7 * 7, num_clusters)

    def forward(self, x, edge_index):
        x = F.relu(self.conv1_1(x))
        x = F.relu(self.conv1_2(x))

        # Обчислюємо представлення зв'язків
        edge_features = self.edge_conv(x)

        # Обчислюємо повідомлення
        messages = self.attention(edge_features, edge_features, edge_features)

        # Оновлюємо представлення елементів
        x = x + messages

        # Агрегуємо представлення елементів
        x = self.pool(x)
```

```
# Виводимо значення ознак
x = x.view(-1, 32 * 7 * 7)
x = self.fc(x)

return x
```

3.5. Реалізація MEA

Для перевірки можливостей архітектури MEA, була збудована нейронна мережа, архітектура якої зображена на рисунку 3.4:

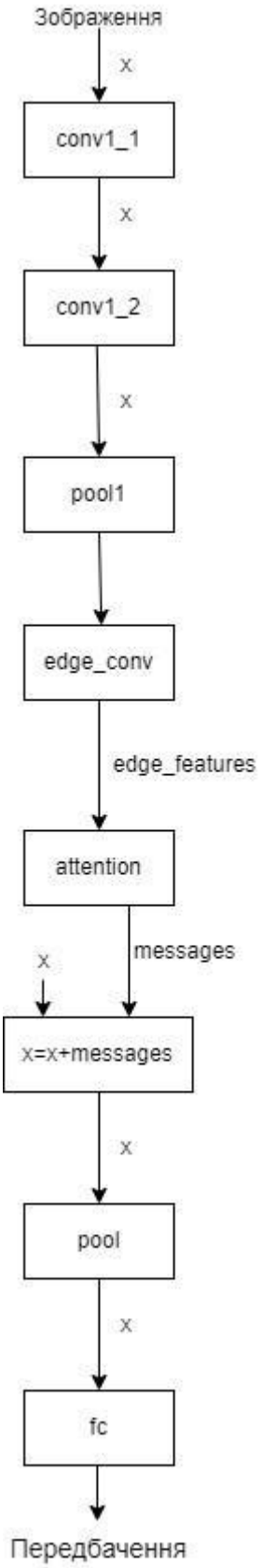

Рис. 3.4. Збудована архітектура MEA

Де,

conv1_1: Шар згортки 2D з 32 фільтрами розміром 3x3.

conv1_2: Шар згортки 2D з 32 фільтрами розміром 3x3.

pool1: Шар агрегації результатів згортки.

edge_conv: Шар згортки 2D з 32 фільтрами розміром 1.

attention: Шар уваги.

pool: Шар агрегації повідомлень.

fc: Вихідний шар.

Передбачення: У якості передбачення мережа повертає вектор кластеризації.

Збудована архітектура має два згорткові шари з ядром 3x3 та кроком 1, шар агрегації результатів згортки та один шар з ядром 1 та кроком 1 для виділення властивостей зв’язків у графі. Після цього результати згортки попадають в шар attention, який формує повідомлення для оновлення вузлів графа. Результати оновлення графу подаються в шар агрегації повідомлень, а з нього потрапляють на повнозв’язний шар, який формує результати кластеризації. Основна відмінність між ними полягає в тому, що MEA GNN використовує MEA (Multi-Edge Attention) для визначення важливості кожного зв'язку в графі, а Hi-LANDER GNN використовує звичайну увагу.

Нижче наведений код збудованої архітектури, написаний мовою Python з використанням пакета PyTorch:

Лістинг 3.4:

```
import torch
import torch.nn as nn

class MEA_GCN(nn.Module):

    def __init__(self, num_clusters):
        super(MEA_GCN, self).__init__()

        # Представлення елементів графу
        self.conv1_1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv1_2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
```

```
    # Представлення зв'язків графу
    self.edge_conv = nn.Conv2d(32, 32, kernel_size=1, stride=1, padding=0)

    # Механізм оновлення повідомлень
    self.attention = nn.MultiHeadAttention(heads=8)

    # Функція агрегації повідомлень
    self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

    # Вихідний шар
    self.fc = nn.Linear(32 * 7 * 7, num_clusters)

def forward(self, x, edge_index):
    x = F.relu(self.conv1_1(x))
    x = F.relu(self.conv1_2(x))

    # Обчислюємо представлення зв'язків
    edge_features = self.edge_conv(x)

    # Обчислюємо MEA
    messages = self.attention(edge_features, edge_features, edge_features)

    # Оновлюємо представлення елементів
    x = x + messages

    # Агрегуємо представлення елементів
    x = self.pool(x)

    # Виводимо значення ознак
    x = x.view(-1, 32 * 7 * 7)
    x = self.fc(x)

    return x
```

3.6. Навчання та перевірка

У якості вхідного набору даних були використані зображення геометричних фігур. Датасет складався зі 150 зображень. Навчання мережі Кохонена не схоже на навчання решти архітектур. Для неї не сформульовано поняття функції втрат, тому під час її навчання через неї були просто пропущені вхідні зображення з навчального набору даних.

Решта архітектур використовують класичний метод зворотнього розповсюдження помилки[8]. Для їх навчання було обрано 10 епох. На рисунках 3.5, 3.6 та 3.7 наведені графіки функції втрат для DeepCut, Hi-LANDER та MEA відповідно.

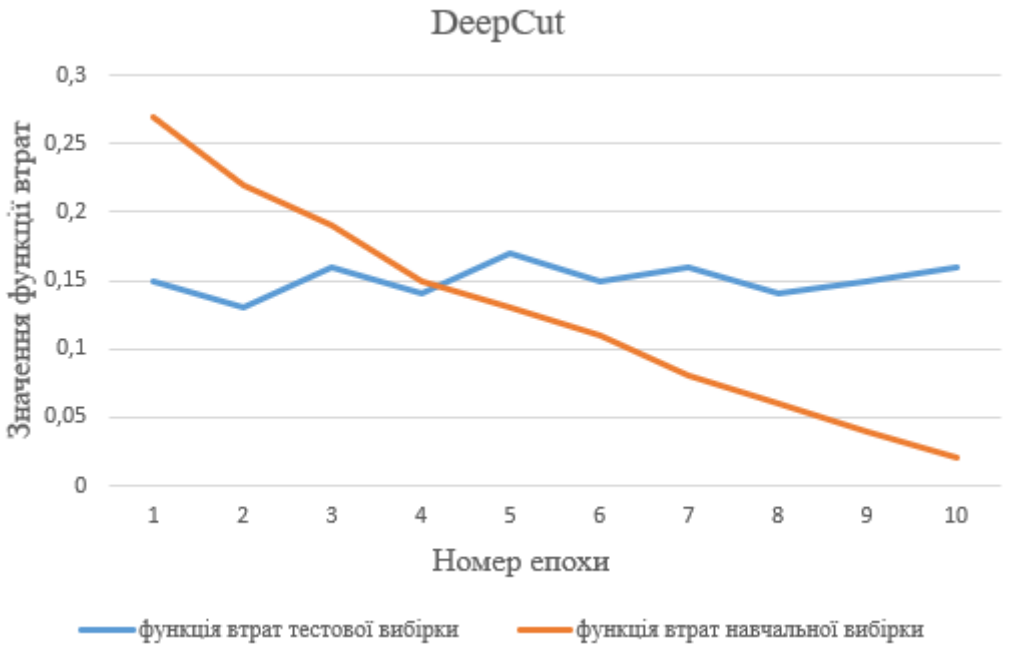

Рис. 3.5. Функція втрат DeepCut

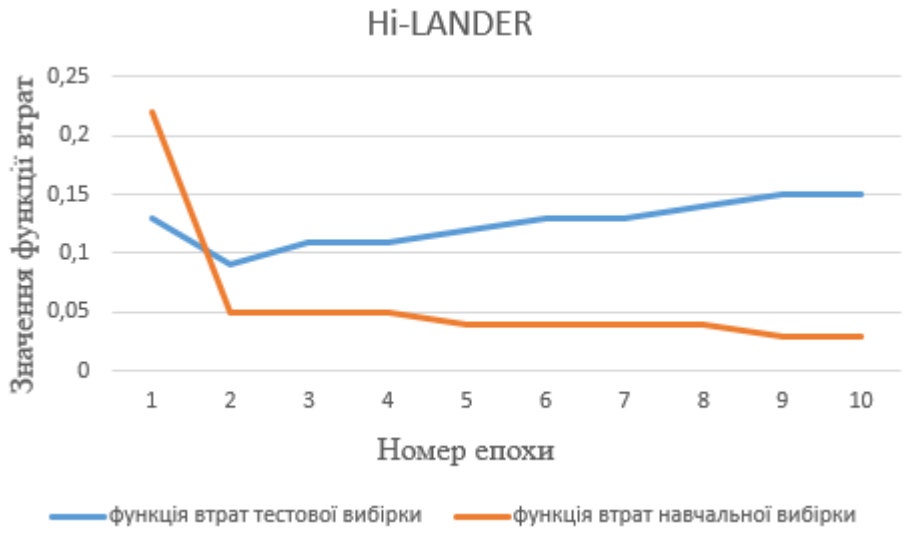

Рис. 3.6. Функція втрат Hi-LANDER

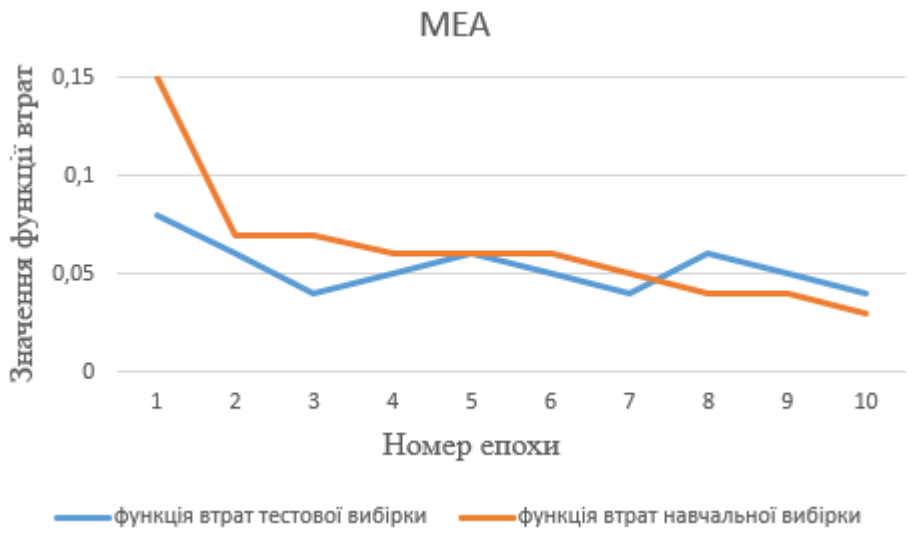

Рис. 3.7. Функція втрат MEA

Навчання глибоких нейронних мереж відбувалося на навчальній вибірці 128 зображень, а тестування відбувалося на вибірці розміром 22 зображення. З результатів видно, що архітектура Hi-LANDER перенавчилася. Скоріш за все це пов’язано з малим набором даних. Ефект перенавчання також частково спостерігається в архітектурі MEA, хоча виражений не так яскраво.

Архітектура DeepCut не показує перенавчання, хоча навчається повільно. В таблиці 3.2 наведений час прогнозування для всіх чотирьох мереж для зображення розміром 1980 на 1020.

Таблиця 3.2

Результати випробувань збудованих архітектур

Архітектура	Кількість випробувань	Середня точність	Час прогнозу
DeepCut	10	0,92	0,3 с
Карти Кохонена	10	0,83	0,1 с
Hi-LANDER	10	0,86	0,4 с
MEA	10	0,95	0,4 с

3.7. **Результати кластеризації**

Після навчання та тестування мережі, було виконано обчислення метрики силуету для мережі Кохонена, а також для мереж DeepCut, Hi-LANDER та MEA разом з класичним методом кластеризації k-means. Крім того, було обраховано значення цієї метрики для алгоритму k-means. Нижче наведено код функції, яка виконує відповідний розрахунок:

Лістинг 3.5:

```
def silhouette(data, labels, with_gcn=True):
    if with_gcn:
        gcn = DeepCut(data.shape[1])
        gcn.fit(data)
        embeds = gcn.predict(data)
    else:
        embeds = data
    clusters = torch.unique(labels)
    n_clusters = len(clusters)
    cluster_boundaries = []
    for cluster in clusters:
```

```
            cluster_boundaries.append(embeds[labels == cluster].mean(0))
        silhouettes = []
        for i, label in enumerate(labels):
            cluster_center = cluster_boundaries[label]
            within_cluster_distance = torch.min(torch.sum((embeds[i] - cluster_center)**2, 1))
            between_cluster_distances = []
            for other_label in clusters:
                if other_label == label:
                    continue
                other_cluster_center = cluster_boundaries[other_label]
                                between_cluster_distance = torch.min(torch.sum((embeds[i] -
other_cluster_center)**2, 1))
                between_cluster_distances.append(between_cluster_distance)
                    silhouette = (within_cluster_distance - torch.min(between_cluster_distances)) /
(torch.max(between_cluster_distances) - torch.min(between_cluster_distances))
            silhouettes.append(silhouette)

        return torch.min(silhouettes)
```

Результати обчислень наведено в таблиці нижче.

Таблиця 3.3

Результати обчислень

Назва мережі	Метрика силуету
Карти Кохонена	0,28
DeepCut	0,42
Hi-LANDER	0,36
MEA	0.4
K-means	0.21

За результатом аналізу таблиці 3.3 в якості основи для розробленої інформаційної технології була обрана графова нейронна мережа DeepCut у сполучені з методом кластеризації K-means та методом виділення прямокутника для об’єкта, що був розроблений в рамках цієї комплексної

кваліфікаційної роботи. Результат виділення прямокутників для об’єктів наведено на рисунку 3.8.

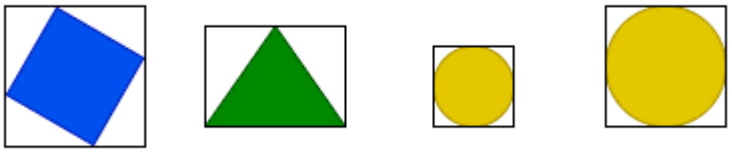

Рис. 3.8. Результати роботи розробленої інформаційної технології

3.8. Висновок до розділу

Експеримент показав, що мережа Кохонена спрацювала швидше. Мережі MEA та Hi-LANDER швидко перенавчилися, але мережа DeepCut показала кращий час та кращу поведінку попри малий обсяг даних для навчання. Обчислення метрики силуету показало, що сполучення нейромережі DeepCut та алгоритму K-means дає кращій за інші способи кластеризації результат. Таким чином мету кваліфікаційної роботи досягнуто. Запропоновано інформаційну технологію, яка використовує сполучення мережі DeepCut, алгоритму K-means та методу виділення прямокутників для точнішого виявлення у системах машинного зору в умовах обмеженої обчислювальної потужності.

ВИСНОВОК

В ході виконання атестаційної роботи був виконаний аналіз способу визначення об’єктів на зображенні в системах машинного зору, досліджені нейромережеві методи кластеризації точок на зображенні, метрики кластеризації, як спосіб оцінки результатів її роботи, були проаналізовані теоретичні часові та просторові оцінки різних нейромережевих архітектур кластеризації та виконано побудову та перевірку кластеризаційних нейромереж у системі машинного зору, що має обмежену обчислювальну продуктивність. Був розроблений метод перевірки метрики силуету за результатами кластеризації для оцінки точності побудованих архітектур в умовах обмежених обчислювальних потужностей. За результатами виконання роботи запропонована інформаційна технологія, що має включати в себе нейронну мережу архітектури DeepCut, метод кластеризації K-Means та метод виділення прямокутників для об’єктів на кластеризованому зображені. Результати застосування розробленої технології можуть бути використані у класифікаційних алгоритмах та рекурентних нейронних мережах для стеження за об’єктами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. “Learning Hierarchical Graph Neural Networks for Image Clustering” Yifan Xing* Tong He* Tianjun Xiao Yongxin Wang Yuanjun Xiong Wei Xia David Wipf Paul Zheng Zhang Stefano Soatto
2. “DeepCut: Unsupervised Segmentation using Graph Neural Networks Clustering” Amit Aflalo , Shai Bagon , Tamar Kashti , and Yonina Eldar
3. "Self-Organizing Maps with Local Learning" авторов G. M. Hirata и H. Ishibuchi.
4. "Self-Organizing Maps: Fundamentals and Applications" авторов T. Kohonen, S. Kaski, K. Mäkisara и R. Lagus.
5. "DeepCut: A Deeply-Learned Image Segmentation Model for Natural Images"
6. "Multi-scale Edge Adaptive Image Segmentation Using Local Density"
7. "Self-Organizing Maps"
8. "Preprocessing for Image Segmentation"
9. "Evaluation of Clustering Algorithms"
10. Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. IEEE Transactions on pattern analysis and machine intelligence, 22(8):888–905, 2000
11. Mahmoud Assran, Mathilde Caron, Ishan Misra, Piotr Bojanowski, Armand Joulin, Nicolas Ballas, and Michael Rabbat. Semi-supervised learning of visual features by nonparametrically predicting view assignments with support samples. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 8443–8452, 2021.
12. Shai Bagon and Meirav Galun.
13. Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In International Conference on Machine Learning, pages 874–883. PMLR, 2020.
14. https://medium.com/@a01740285/clustering-of-images-with-self-organizing-maps-9563910df35f

15. P. Dostál, P. Pokorný. Cluster analysis and neural network. In: Technical Computing Prague 2009, 17th Annual Conference Proceedings. Prague, Czech Republic, 2008.
16. L. Meza, L. Neto. Modelling with Self-Organising Maps and Data Envelopment Analysis: A Case Study in Educational Evaluation. In: book Self Organizing Maps - Applications and Novel Algorithm Design, Published by InTech, Rijeka, Croatia, january 2011, p. 33, ISBN 978-953-307- 546-4
17. Edo Collins, Radhakrishna Achanta, and Sabine Susstrunk. Deep feature factorization for concept discovery. In Proceedings of the European Conference on Computer Vision (ECCV), pages 336–352, 2018.
18. Zixuan Huang and Yin Li. Interpretable and accurate finegrained recognition via region grouping. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8662–8672, 2020.
19. Gorokhovatskyi O., Gorokhovatskyi V., Peredrii O.: Analysis of Application of Cluster Descriptions in Space of Characteristic Image Features. Data. 3(4), 52 (2018). doi:10.3390/data3040052
20. Kumar, V., Namboodiri, A., Jawahar, C.V.: Semi-supervised annotation of faces in image collection. Signal, Image and Video Processing. 12(1), pp. 141–149 (2018). doi:10.1007/s11760-017-1140-5
21. He, T., et al.: Bag of tricks for image classification with convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 558–567 (2019). doi:10.1109/CVPR.2019.00065
22. da Silva, I.N., et al.: Self-organizing Kohonen networks. Artificial Neural Networks. pp. 157–172 (2017). doi:10.1007/978-3-319-43162-8_8
23. Hu, Z., Bodyanskiy, Y., Tyshchenko, O.: Kohonen Maps and Their Ensembles for Fuzzy Clustering Tasks'. Self-Learning and Adaptive

Algorithms for Business Applications. pp. 51–77 (2019). doi:10.1108/978-1-83867-171-620191004

24. «End-to-End Hierarchical Clustering with Graph Neural Networks» Nicholas Choma A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science at New York University 2019.
25. Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jorg Sander. Optics: ordering points to identify the clus- ¨ tering structure. ACM Sigmod record, 28(2):49–60, 1999
26. Thomas Bonald, Bertrand Charpentier, Alexis Galland, and Alexandre Hollocou. Hierarchical graph clustering using node pair sampling. arXiv preprint arXiv:1806.01664, 2018.
27. Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. arXiv preprint arXiv:1801.10247, 2018.
28. Michael Defferrard, Xavier Bresson, and Pierre Van- ¨ dergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in neural information processing systems, pages 3844–3852, 2016.
29. Alex Lipov and Pietro Lio. A multiscale graph convolutional network using hierarchical clustering. arXiv preprint arXiv:2006.12542, 2020
30. Li Mi and Zhenzhong Chen. Hierarchical graph attention network for visual relationship detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 13886–13895, 2020
31. Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems, pages 8026–8037, 2019

32. Ulrike Von Luxburg. A tutorial on spectral clustering. Statistics and computing, 17(4):395–416, 2007
33. Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In Advances in neural information processing systems, pages 4800–4810, 2018.
34. «Image Clustering Method Based on Self Organization Mapping: SOM Derived Density Maps and Its Application for Landsat Thematic Mapper Image Clustering» Kohei Arai Graduate School of Science and Engineering Saga University.
35. Kohei Arai, Fundamental Theory for Image Processing Algorithms, Gakujutu-Tosho-Publishing Co. Ltd., 1999.
36. Kohei Arai, Learning processes of image clustering method with density maps derived from Self-Organizing Mapping(SOM), Journal of Japan Photogrammetry and Remote Sensing, 43, 5, 62-67, 2004.
37. Jouko Lampinen and Timo Kostiainen, Generative probability density model in the Self-Organizing Map. In U. Seiffert and L. Jain, editors, Self-organizing neural networks: Recent advances and applications. pages 75-94. Physica Verlag, 2002.
38. https://www.ti.com/

ДОДАТОК А

Відомість матеріалів кваліфікаційної роботи

		Позначення	Найменування	Кільк. аркушів	Примітка
	1				
	2		Документація		
	3				
	4	**ІТКІ.ДП 18.01.ДА.ПЗ**	Пояснювальна записка	85	
	5				
	6		Диск CD-R з презентацією	1	

					ІТКІ.ДП 18.01.ДА.ПЗ			
Зм	Лист	№ докум.	Підпис	Дата				
Розроб.		Явтухов			**Матеріали кваліфі-каційної роботи**	Літ.	Аркуш	Аркушів
Керівник		Коротенко					1	1
Рецензент		Ширін				НТУ «ДП» 8; 126м-22-1		
Н.контр.		Коротенко						
Зав. каф.		Гнатушенко						

ДОДАТОК Б

КОД ПРОГРАМИ

networks.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from magicdrawer import draw_diagram
from sklearn.cluster import KMeans
import numpy as np

class DeepCutGCN(nn.Module):

    def __init__(self, num_clusters):
        super(DeepCutGCN, self).__init__()

        # Представлення елементів графу
        self.conv1_1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv1_2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)

        # Представлення зв'язків графу
        self.edge_conv = nn.Conv2d(32, 32, kernel_size=1, stride=1, padding=0)

        # Механізм оновлення повідомлень
        self.attention = nn.Softmax(dim=1)

        # Функція агрегації повідомлень
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Вихідний шар
        self.fc = nn.Linear(32 * 7 * 7, num_clusters)

    def forward(self, x, edge_index):
        x = F.relu(self.conv1_1(x))
        x = F.relu(self.conv1_2(x))

        # Обчислюємо представлення зв'язків
        edge_features = self.edge_conv(x)

        # Обчислюємо повідомлення
        messages = self.attention(edge_features) * x

        # Оновлюємо представлення елементів
        x = x + messages

        # Агрегуємо представлення елементів
        x = self.pool(x)

        # Виводимо значення ознак
        x = x.view(-1, 32 * 7 * 7)
        x = self.fc(x)
```

```
        return x

class HiLANDERGCN(nn.Module):

    def __init__(self, num_clusters):
        super(HiLANDERGCN, self).__init__()

        # Представлення елементів графу
        self.conv1_1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv1_2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Представлення зв'язків графу
        self.edge_conv = nn.Conv2d(32, 32, kernel_size=1, stride=1, padding=0)

        # Механізм оновлення повідомлень
        self.attention = nn.Softmax(dim=1)

        # Функція агрегації повідомлень
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Вихідний шар
        self.fc = nn.Linear(32 * 7 * 7, num_clusters)

    def forward(self, x, edge_index):
        x = F.relu(self.conv1_1(x))
        x = F.relu(self.conv1_2(x))

        # Обчислюємо представлення зв'язків
        edge_features = self.edge_conv(x)

        # Обчислюємо повідомлення
        messages = self.attention(edge_features, edge_features, edge_features)

        # Оновлюємо представлення елементів
        x = x + messages

        # Агрегуємо представлення елементів
        x = self.pool(x)

        # Виводимо значення ознак
        x = x.view(-1, 32 * 7 * 7)
        x = self.fc(x)

        return x

class MEA_GCN(nn.Module):

    def __init__(self, num_clusters):
        super(MEA_GCN, self).__init__()
```

```
        # Представлення елементів графу
        self.conv1_1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.conv1_2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Представлення зв'язків графу
        self.edge_conv = nn.Conv2d(32, 32, kernel_size=1, stride=1, padding=0)

        # Механізм оновлення повідомлень
        self.attention = nn.MultiHeadAttention(heads=8)

        # Функція агрегації повідомлень
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Вихідний шар
        self.fc = nn.Linear(32 * 7 * 7, num_clusters)

    def forward(self, x, edge_index):
        x = F.relu(self.conv1_1(x))
        x = F.relu(self.conv1_2(x))

        # Обчислюємо представлення зв'язків
        edge_features = self.edge_conv(x)

        # Обчислюємо МЕА
        messages = self.attention(edge_features, edge_features, edge_features)

        # Оновлюємо представлення елементів
        x = x + messages

        # Агрегуємо представлення елементів
        x = self.pool(x)

        # Виводимо значення ознак
        x = x.view(-1, 32 * 7 * 7)
        x = self.fc(x)

        return x

class KohonenMap(nn.Module):
    def __init__(self, num_clusters):
        super(KohonenMap, self).__init__()

        self.num_clusters = num_clusters

        self.weights = nn.Parameter(torch.randn(num_clusters, 28 * 28))

    def forward(self, x):
        x = x.view(1, -1)

        distances = (x - self.weights).pow(2).sum(-1)
        winner = distances.argmin(-1)
```

```
            return winner

        def silhouette(data, labels, with_gcn=True):
            if with_gcn:
                gcn = DeepCut(data.shape[1])
                gcn.fit(data)
                embeds = gcn.predict(data)
            else:
                embeds = data
            clusters = torch.unique(labels)
            n_clusters = len(clusters)
            cluster_boundaries = []
            for cluster in clusters:
                cluster_boundaries.append(embeds[labels == cluster].mean(0))
            silhouettes = []
            for i, label in enumerate(labels):
                cluster_center = cluster_boundaries[label]
                within_cluster_distance = torch.min(torch.sum((embeds[i] - cluster_center)**2, 1))
                between_cluster_distances = []
                for other_label in clusters:
                    if other_label == label:
                        continue
                    other_cluster_center = cluster_boundaries[other_label]
                    between_cluster_distance = torch.min(torch.sum((embeds[i] - other_cluster_center)**2, 1))
                    between_cluster_distances.append(between_cluster_distance)
                        silhouette = (within_cluster_distance - torch.min(between_cluster_distances)) / (torch.max(between_cluster_distances) -
torch.min(between_cluster_distances))
                silhouettes.append(silhouette)

            return torch.min(silhouettes)




        def main():
            deepcut_net = DeepCutGCN(4)
            hilander_net = HiLANDERGCN(4)
            mea_net = MEA_GCN(4)
            kohonen_net = KohonenMap(4)

            transform = transforms.Compose([
                transforms.Resize(224),
                transforms.ToTensor(),
                transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
            ])
            dataset = datasets.ImageFolder(root='./images', transform=transform)
            dataloader = torch.utils.data.DataLoader(dataset, batch_size=128, shuffle=True)

            optimizer = torch.optim.Adam(net.parameters(), lr=0.0001)

            deepcut_loses = []
```

```
hilander_loses = []
mea_losses = []
for epoch in range(10):
    for i, (images, labels) in enumerate(dataloader):
        images = images.to(torch.float)
        labels = labels.to(torch.long)

        outputs = deepcut_net(images)
        loss = F.cross_entropy(outputs, labels)
        deepcut_loses.append(loss)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        outputs = hilander_net(images)
        loss = F.cross_entropy(outputs, labels)
        hilander_losses.append(loss)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        outputs = mea_net(images)
        loss = F.cross_entropy(outputs, labels)
        mea_losses.append(loss)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
torch.save(deepcut_net.state_dict(), './model.pth')
torch.save(hilander_net.state_dict(), './model.pth')
torch.save(mea_net.state_dict(), './model.pth')

dataset = datasets.ImageFolder(root='./test_images', transform=transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=128, shuffle=True)

test_deepcut_loses = []
test_hilander_loses = []
test_mea_losses = []
deepcut_sil = []
hilander_sil = []
mea_sil = []
kmeans = KMeans(n_clusters=4)

for epoch in range(10):
    for i, (images, labels) in enumerate(dataloader):
        images = images.to(torch.float)
        labels = labels.to(torch.long)

        outputs = deepcut_net(images)
        loss = F.cross_entropy(outputs, labels)
        test_deepcut_loses.append(loss)
        labels = kmeans.fit_predict(outputs)
        deepcut_sil.append(silhouette(images, labels))
```

```
        outputs = hilander_net(images)
        loss = F.cross_entropy(outputs, labels)
        hilander_losses.append(loss)
        labels = kmeans.fit_predict(outputs)
        hilander_sil.append(silhouette(images, labels))

        outputs = mea_net(images)
        loss = F.cross_entropy(outputs, labels)
        mea_losses.append(loss)
        labels = kmeans.fit_predict(outputs)
        mea_sil.append(silhouette(images, labels))

draw_diagram(deepcut_losses, test_deepcut_losses)
draw_diagram(hilander_losses, test_hilander_losses)
draw_diagram(mea_losses, test_mea_losses)
print(np.average(deepcut_sil), np.average(hilander), np.average(mea_sil))

dataset = datasets.MNIST(root='./test_images', train=False, transform=transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=128, shuffle=False)
sil = []
for images, labels in dataloader:
    labels = labels.view(-1)
    predictions = net(images)
    correct = (predictions == labels).sum().item()
    sil.append(silhouette(images, labels))
print(np.average(sil))
```

ДОДАТОК В

ВІДГУК
на комплексну кваліфікаційну роботу рівня магістра «Розробка інформаційної технології визначення об’єктів у системах машинного зору в умовах обмежених обчислювальних потужностей» студента групи 126м-22-1 Явтухова Артема Володимировича

1 Мета даної кваліфікаційної роботи – підвищення точності визначення об’єкту на зображеннях у системах машинного зору за умови обмеженої продуктивності.

2 Обрана тема актуальна тому, що під час аналізу інформації що надходить від системи машинного зору є ідентифікація та класифікація об’єктів навколишнього світу, враховуючи те що роботизовані системи часто мають обмежений обчислювальний ресурс і використання навіть традиційних алгоритмів іноді може бути недоречним через погану часову складність цих алгоритмів. Дані, що надходять у реальному масштабі часу можуть бути обробленні невчасно або взагалі загубленні. Таким чином постає питання: який з алгоритмів традиційного виділення точок, що належать зображенню певного об’єкта обрати. Добір має давати достатню точність за умови кращої часової і бажано просторової складності алгоритму. Також потрібно зважати на особливості використання даних алгоритмів за умови застосування процесорів штучного інтелекту.

3 Тема кваліфікаційної роботи відповідного рівня безпосередньо пов'язана з об'єктом діяльності магістра спеціальності 126 «Інформаційні системи та технології» галузі знань 12 «Інформаційні технології» – створення інформаційних технологій різного призначення і застосування.

4 Явища і процеси, що досліджуються в даній кваліфікаційній роботі і обрані для моделювання, оцінювання та реалізації – віднесені в освітньо-кваліфікаційній характеристиці магістрів до класу дослідних та евристичних, рішення яких заснована на знаково-понятійних уміннях.

5 Робота складається з трьох розділів. Перший розділ присвячений аналізу теми дослідження та постановці задачі. У другому розділі наведено проектну складову вирішення завдання. Третій розділ присвячено порівнянню методів кластерізації зображень в умовах обмежених обчислювальних потужностей. Оригінальність отриманих в роботі наукових результатів та їх наукова новизна полягають у наступному:

- досліджено і обгрунтовано застосування відповідних методів кластерізації складних даних;
- детально пророблено алгоритмічні компоненти методів кластерізації;
- порівняно обрані алгоритми кластерізації;
- розроблені та реалізовані алгоритми обчислення метрик кластерізації для оцінки точності отримання результатів при застосуванні обраних методів в умовах обмежених обчислювальних потужностей.

- обрано сполучення нейромережі DeepCut та алгоритму K-means, які відповідають вимогам точності та швидкості.

6 Практичне значення результатів роботи полягає в обранні потрібних для вирішення поставленої задачі алгоритмів на основі порівняння групи найбільш відомих.

7 Практичні результати кваліфікаційної роботи отримані із застосуванням відповідних технічних і програмних засобів, мови Python, а також програмних продуктів MS Word і MS PowerPoint на інформаційно-технологічній платформі Windows.

8 Оформлення графічних матеріалів до кваліфікаційної роботи рівня магістр виконано на сучасному рівні і відповідає вимогам, що пред'являються до рівня виконання робіт даної кваліфікації.

9 Ступінь самостійності виконання кваліфікаційної роботи достатньо висока.

10 Деякі дискусійні положення та недоліки, які мають місце в роботі:

а) недостатньо повно визначено кінцеву функціональну придатність даної розробки;

б) дещо обмежене застосування розробленої інформаційної технології;

в) у роботі мають місце поодинокі орфографічні та стилістичні помилки.

Незважаючи на вищевказані зауваження, кваліфікаційна робота в цілому заслуговує оцінки «___відмінно (95 балів)_____» та присвоєння здобувачу відповідної кваліфікації.

Керівник кваліфікаційної роботи,
проф. кафедри ІТКІ, д.т.н. Г.М. Коротенко

ДОДАТОК Г

РЕЦЕНЗІЯ
на комплексну кваліфікаційну роботу рівня магістра
«Розробка інформаційної технології визначення об’єктів у системах машинного зору в умовах обмежених обчислювальних потужностей»
студента групи 126м-22-1 Явтухова Артема Володимировича

Розглянута робота присвячена підвищення точності визначення об’єкту на зображеннях у системах машинного зору за умови обмеженої продуктивності.

Завдання і зміст кваліфікаційної роботи відповідає головній цілі - перевірці знань і ступеня підготовленості студента за фахом 126 «Інформаційні системи та технології» галузі знань 12 «Інформаційні технології».

Зміст пояснювальної записки кваліфікаційної роботи відповідає необхідним критеріям та затвердженій темі.

Актуальність обраної теми обумовлена тим, що дослідження ефективності застосування інформаційних технологій і відповідних програмних засобів продовжують розвиватися на базі розширення їхнього спектру.

Повнота і глибина вирішення задач, поставлених в завданні на кваліфікаційну роботу є достатньою.

Оформлення пояснювальної записки кваліфікаційної роботи виконано в повній відповідності з діючими стандартами і нормативними вимогами.

Наукова новизна результатів кваліфікаційної роботи визначається тим, що підведено базу під розробку інформаційної технології, що призначена для визначення об’єктів у системах машинного зору в умовах обмежених обчислювальних потужностей.

Практичне значення результатів роботи полягає у порівнянні та обранні найбільш дійових алгоритмів для визначення об’єктів у системах машинного зору в умовах обмежених обчислювальних потужностей.

До числа загальних зауважень і недоліків роботи слід віднести:

1) не до кінця розкриті функціональні особливості реалізації та результатів практичного застосування розробленої технології;

2) дещо спрощений опис кінцевих результатів роботи.

Однак, зазначені зауваження не здійснюють істотного впливу на підсумкові результати кваліфікаційної роботи і не знижують її безумовну практичну та наукову цінність.

Таким чином, слід зробити висновок, що кваліфікаційна робота в цілому

заслуговує оцінки «_____», а її виконавець присвоєння відповідної кваліфікації.

Рецензент, доцент кафедри програмного
забезпечення комп’ютерних систем НТУ
«ДП», к.т.н.. А.Л. Ширін