Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів аналізу графів для вирішення задач комп'ютерного зору
(тема)

Виконав:
студент 2 курсу, групи СШІм-21-2
Кошурников Г.О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту
(повна назва спеціалізації)

Керівник проф. Єрохін А.Л.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кошурникову Глібу Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів аналізу графів для вирішення задач комп'ютерного зору

затверджена наказом університету від 31 березня 20 23 р. № 306Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 20 23 р.

3. Вихідні дані до роботи Науково-технічні публікації, дані Інтернет-джерел щодо методів використання графових алгоритмів та структур в задачах комп'ютерного зору

4. Перелік питань, що потрібно опрацювати в роботі

1) Аналіз предметної галузі

2) Сегментація зображень нейронними мережами та графовими алгоритмами

3) Обґрунтування прийнятих технологій

4) Проведені експерименти та програмна реалізація

5) Додатковий функціонал

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	03.04.2023	Виконано
2	Аналіз предметної області	04.04.2023-05.04.2023	Виконано
3	Постановка завдання та узгодження з керівником	05.04.2023-13.04.2023	Виконано
4	Аналіз методів сигментації зображень	13.04.2023-23.04.2023	Виконано
5	Дослідження технологій	23.04.2023-25.04.2023	Виконано
6	Написання програмного додатку	25.04.2023-29.04.2023	Виконано
7	Експериментальні дослідження	29.04.2023-01.05.2023	Виконано
8	Написання пояснювальної записки	01.05.2023-08.05.2023	Виконано
9	Попередній захист	10.05.2023	Виконано
10	Захист перед ЕК	18.05.2023	

Дата видачі завдання 3 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Єрохін А.Л.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 68 с., 14 рис., 1 дод., 19 джерел.

АНАЛІЗ ГРАФІВ, АНАЛІЗ ЗОБРАЖЕНЬ, ГЛИБОКА НЕЙРОННА МЕРЕЖА, ГРАФОВІ АЛГОРИТМИ, СЕГМЕНТАЦІЯ.

Об'єкт дослідження – архітектура та навчання глибоких штучних нейронних мереж та алгоритмів машинного навчання.

Предмет дослідження – методи поєжнання графових алгоритмів та моделей машинного та глибокого навчання.

Мета роботи – застосування графових алгоритмів для покращення сегментації зображень.

Методи дослідження – аналіз існуючих нейромережевих підходів для сегментації зображень, розробка алгоритму сегментації зображень на основи графових алгоритмів, програмна реалізація та проведення комп’ютерного експерименту, обробка та аналіз отриманих результатів. Під час виконання атестаційної роботи проведений теоретичний аналіз літературних джерел щодо методів використання графових аглротимів у задачах сегментації зображень, наукових публікацій щодо розробки комбінованих алгоритмів машинного навччання для задач сегментації.

ABSTRACT

Explanatory note: 68 p., 14 fig., 1 ann., 19 sources.

DEEP NEURAL NETWORK, GRAPH ALGORITHMS, GRAPH ANALYSIS, IMAGE ANALYSIS, SEGMENTATION.

The object of research is the architecture and training of deep artificial neural networks and machine learning algorithms.

The subject of research is methods of combining graph algorithms and machine and deep learning models.

The purpose of the work is to use graph algorithms to improve image segmentation.

Research methods – analysis of existing neural network approaches for image segmentation, development of an image segmentation algorithm based on graph algorithms, software implementation and computer experiment, processing and analysis of the obtained results. During the attestation work, a theoretical analysis of literary sources on methods of using graph algorithms in image segmentation tasks, scientific publications on the development of combined machine learning algorithms for segmentation tasks was carried out.

ЗМІСТ

Вступ 8
1 Аналіз предметної галузі та постановка задачі 9
1.1 Сегментація зображень 9
1.2 Графові алгоритми 10
1.3 Діаграми та блок схеми у розробці програмного забезпечення 12
1.4 Постановка завдання дослідження 15
2 Сегментація зображень нейронними мережами та графовими алгоритмами 17
2.1 Повнозв'язна згорткова мережа (FCN) 17
2.2 U-Net 18
2.3 Graph Cut 19
2.4 Алгоритм Фелзенсвальба 21
2.6 Порівняльний аналіз алгоритмів 24
3 Обґрунтування прийнятих технологій 27
3.1 Мова програмування Python 27
3.2 PyTorch 28
3.3 Програмне забезпечення PyTorch Geometric 30
3.4 Бібліотека OpenCV 32
3.5 Бібліотека Numpy 34
3.5 Hugging Face 36
3.6 NetworkX 39
4 Проведені експерименти та програмна реалізація 41
4.1 LambdaNetworks для інстансної сегментації зображень 41
4.2 LambdaNetworks для object detection 43
4.3 LambdaNetworks для сегментації ліній різного розміру 44
4.4 Пошук блоків на діаграммах за допомогою LambdaNetworks 45
4.5 Представлення сегментованих ліній та знайдених блоків у вигляді графа 48

5 Додатковий функціонал 58
5.1 ChatGPT 59
5.2 GitHub Copilot 60
5.3 OpenAI GPT 61
5.4 Висновки щодо генерації коду 62
Висновки 64
Перелік джерел посилання 66
Додаток А Відомість кваліфікаційної роботи 68

ВСТУП

Сегментація зображень відіграє важливу роль у бізнесі та індустрії. Відокремлення об’єктів на зображенні дозволяє підвищити точність розпізнавання та класифікації зображень, що може бути корисним у різних сферах, наприклад, у медицині для діагностики захворювань на основі медичних зображень, у виробництві для контролю якості продукції, у відеоспостереженні для виявлення порушень та в управлінні транспортними системами для розпізнавання автомобілів на дорогах. Крім того, розвиток методів сегментації зображень стимулює зростання нових інновацій та розвиток нових технологій, що створює нові можливості для бізнесу та підприємництва. Наприклад, у відеоігровій індустрії, методи сегментації зображень використовуються для створення реалістичних ефектів та графічного дизайну.

Отже, дослідження та розвиток методів сегментації зображень має велике значення для багатьох галузей людської діяльності та може стати ключовим чинником у покращенні якості та ефективності процесів, пов'язаних з обробкою зображень.

Використання графових алгоритмів для сегментації зображень також може бути важливим інструментом для оптимізації процесу розробки програмного забезпечення. Аналіз діаграм та схем, що використовують графові структури, може бути більш ефективним завдяки застосуванню графових алгоритмів для сегментації зображень, які дозволяють візуалізувати та аналізувати структуру діаграм та схем на основі графових моделей. Це може зробити процес розробки ПО більш точним та ефективним, зменшуючи можливість помилок та збільшуючи швидкість виконання завдань. Таким чином, застосування графових алгоритмів для сегментації зображень може мати велике значення для розробки ПО та інших галузей технічного прогресу

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Сегментація зображень

Задача сегментації зображень полягає у відокремленні об'єктів на зображенні та визначенні їх меж. Це є важливим етапом в обробці зображень і має широкі застосування в багатьох галузях, таких як медицина, промисловість та автоматичне керування. Сегментація зображень може бути розглянута як задача класифікації пікселів зображення на різні класи, що відповідають різним об'єктам на зображенні.

Сегментація є важливим процесом для когнітивної науки, яка вивчає процеси пізнання та розуміння людиною світу [1]. Вона відіграє важливу роль у сприйнятті зображень, оскільки вона дозволяє розрізняти окремі об'єкти на зображенні та зв'язувати їх з певним контекстом. Цей процес дозволяє людині більш ефективно розуміти та інтерпретувати оточуючий світ. Наприклад, при розгляді фотографії людина автоматично розрізняє обличчя людей на зображенні, визначає, де знаходяться люди, та як вони пов'язані між собою. Цей процес відбувається дуже швидко та без усвідомлення, оскільки він залежить від роботи мозку та спеціалізованих систем, які виконують сегментацію та аналіз зображень. З іншого боку, порушення сегментації може призвести до розладів у сприйнятті та розумінні світу навколо. Наприклад, у людей з порушеннями аутистичного спектра спостерігаються порушення в сприйнятті соціального контексту, які пов'язані зі зменшенням здатності розпізнавати та інтерпретувати емоції та інші знаки соціальної взаємодії. Це може бути пов'язано з порушенням роботи систем сегментації та аналізу зображень в мозку. Отже, вивчення процесів сегментації та розуміння їх роботи в мозку може допомогти краще зрозуміти процеси пізнання та розуміння світу людиною, а також сприяти розробці нових технологій та методів обробки зображень та аналізу даних.

Алгоритми сегментації зображень можуть бути подібні до того, як людина

відділяє об'єкти, якщо вони використовують природні механізми розпізнавання образів, що працюють у людському мозку. Такі алгоритми можуть моделювати процес розпізнавання образів у мозку, що включає в себе визначення границь об'єктів, розуміння їхньої форми та структури. Один з підходів до сегментації, який відтворює природні механізми розпізнавання образів, є метод "watershed". Він базується на аналізі яскравості пікселів зображення і використовує принцип "мінімальної енергії", щоб розділити зображення на різні регіони. Цей метод схожий на те, як люди відділяють об'єкти, спираючись на зміну контрастності між областями.

Інший підхід, що відтворює природні механізми розпізнавання образів, полягає у використанні глибоких нейронних мереж. Ці мережі можуть навчатися розпізнавати об'єкти на зображенні, визначаючи їхні границі та форми. Наприклад, у мережах, що використовуються для сегментації, можуть бути використані згорткові шари, які аналізують різні характеристики зображення на різних рівнях абстракції. Після цього зазвичай використовують алгоритм класифікації, який відносить кожен піксель зображення до відповідного класу.

Нейронні мережі є потужним інструментом у задачах машинного навчання, включаючи задачу сегментації зображень. У порівнянні з іншими підходами, такими як порогова обробка та кластеризація, нейронні мережі можуть автоматично вивчати складні залежності між зображенням та його сегментацією, що робить їх більш ефективними для розв'язання складних задач. Але в той самий час підходи засновані на нейронних мережах мають суттєвий недолік, це час та ресурси які потрібно витратити для успішного навчання мережі.

1.2 Графові алгоритми

Графові алгоритми – це підхід до обробки зображень, що

використовує графи для моделювання зв'язків між пікселями на зображенні. Застосування графових алгоритмів в задачах сегментації зображень стало популярним останнім часом і дозволяє отримати більш точні результати.

Одним з найбільш поширених графових алгоритмів є алгоритм сегментації на основі графових різниць (Graph Cut Segmentation) [2]. Цей алгоритм використовує мінімізацію функції енергії, яка включає в себе якість сегментації і стійкість до шуму. Задача полягає в тому, щоб розділити зображення на дві або більше областей, так щоб кожен піксель був класифікований в одну з цих областей, з мінімальною сумарною енергією. Цей алгоритм може бути застосований до зображень різного типу та структури. Ще одним графовим алгоритмом є алгоритм Normalized Cut Segmentation [3], який використовує спектральний аналіз графа зображення для сегментації. Він використовує інформацію про взаємодію між сусідніми пікселями та розбиває зображення на групи пікселів з подібними властивостями.

Графові алгоритми мають ряд переваг, які роблять їх потужним інструментом для вирішення різноманітних задач. По-перше, графи є потужним засобом для моделювання взаємозв'язків між об'єктами, де кожен об'єкт представляється як вершина, а взаємодії між об'єктами – як ребра. Це дозволяє аналізувати та розуміти структуру та зв'язки між елементами системи. По-друге, графові алгоритми дозволяють розбити складну задачу на більш прості підзадачі та вирішувати їх окремо. Це сприяє зменшенню складності задачі та підвищує шанси на успішне розв'язання. Також графові алгоритми можуть бути застосовані до широкого спектру задач в різних галузях, включаючи комп'ютерні науки, математику, фізику, біологію та соціальні науки. Вони є потужним інструментом для моделювання та аналізу складних систем, зв'язків та взаємодій між об'єктами. Графові алгоритми дозволяють знаходити найкоротші шляхи, найоптимальніші розподіли ресурсів та вирішувати задачі оптимізації. Вони використовуються для розв'язання проблем маршрутизації, планування

розкладів, оптимального розміщення ресурсів та багатьох інших оптимізаційних завдань. І наостанок, важливо зазначити, що графові алгоритми можуть ефективно опрацьовувати великі масиви даних, включаючи графи з мільйонами вершин та ребер. Вони здатні до паралельної обробки та оптимізовані для роботи з великими обсягами інформації, що робить їх придатними для сучасних завдань обробки даних.

Хоча графові алгоритми мають багато переваг, вони також мають свої обмеження та проблеми, коли застосовуються до задачі сегментації зображень. Одна з основних проблем – це підбір оптимального розміру графу. Малі графи можуть не мати достатньої інформації для сегментації, тоді як великі графи можуть приводити до збільшення складності обчислень та витрат пам'яті. Крім того, графові алгоритми вимагають відповідного вибору критерію подільності, щоб підібрати оптимальний поділ графу на сегменти. Невірний вибір цього критерію може призвести до недостатньої або зайвої сегментації, що може погіршити якість результуючого зображення. Отже незважаючи на переваги над класичними підходами, графові алгоритми також містять свої недоліки.

1.3 Діаграми та блок схеми у розробці програмного забезпечення

Діаграми у розробці програмного забезпечення є важливим інструментом для візуалізації та аналізу структури, функціональності та взаємодії компонентів програмного продукту. Вони надають можливість розуміти складні системи, моделювати їх процеси та забезпечувати зрозумілу комунікацію між розробниками, аналітиками та клієнтами.

Однією з основних переваг діаграм у розробці програмного забезпечення є їх здатність до абстракції та спрощення складних концепцій. Вони дозволяють представляти складні системи та процеси у вигляді графічних зображень, що спрощує їх розуміння та сприяє легшій комунікації між різними зацікавленими сторонами. Діаграми дозволяють

виявити та усунути потенційні проблеми ще на етапі проектування, що збільшує ефективність розробки та підвищує якість програмного забезпечення. Іншою перевагою діаграм у розробці програмного забезпечення є їх масштабованість. Вони можуть бути застосовані на різних рівнях архітектури програмного продукту, починаючи від високорівневих діаграм загальної структури та функціональності до детальних діаграм класів, послідовностей або компонентів. Це дозволяє розробникам фокусуватись на конкретних аспектах системи та використовувати відповідні діаграми для детального аналізу та реалізації. Крім того, діаграми у розробці програмного забезпечення сприяють зручності та ефективності розробки. Вони дозволяють розробникам легко орієнтуватися у складних системах, зрозуміти їх структуру та взаємозв'язки між компонентами. Це дозволяє ефективно розподіляти завдання між командою розробників, спрощує рефакторинг коду та розуміння системи новими членами команди. Діаграми у розробці програмного забезпечення також покращують комунікацію між розробниками, аналітиками, менеджерами та клієнтами. Вони дозволяють легко передавати інформацію про систему, її функціональність та вимоги. Діаграми стають зрозумілим інструментом для спілкування та обговорення, що допомагає уникнути непорозумінь та забезпечити згоду між усіма зацікавленими сторонами.

Одним з найпоширеніших видів діаграм є UML-діаграми. UML (Unified Modeling Language) – це мова моделювання, яка дозволяє описати різноманітні аспекти програмної системи. UML-діаграми допомагають візуалізувати структуру та поведінку системи, що дозволяє легше зрозуміти її функціонал. До основних видів UML-діаграм відносять:

– діаграма класів (Class diagram) – відображає структуру системи, в якій показані класи, їх властивості та зв'язки між ними;

– діаграма послідовностей (Sequence diagram) – відображає взаємодію об'єктів в системі в певному порядку, що дозволяє відстежувати послідовність виконання дій;

– діаграма активностей (Activity diagram) – описує послідовність дій та процесів в системі, дозволяє визначити алгоритм роботи програми.
Окрім UML-діаграм, використовуються і інші типи діаграм, наприклад, діаграми потоків даних (Data Flow Diagrams), діаграми прецедентів (Use Case Diagrams) та інші [4].

Блок-схеми – це ще один важливий інструмент у розробці програмного забезпечення, який використовується для відображення структури програми та логіки її виконання. Блок-схеми дозволяють візуалізувати послідовність виконання дій, умовні переходи та інше. Важливо згадати також ER-діаграми (Entity-Relationship Diagrams) – це графічні засоби моделювання баз даних, які відображають зв'язки між різними сутностями. ER-діаграми є потужним інструментом для моделювання баз даних, оскільки вони допомагають визначити, які дані потрібно зберігати та як вони пов'язані між собою. Це може сприяти ефективному проектуванню баз даних та допомагати забезпечувати їх правильність і цілісність.

Однією з основних переваг блок-схем є їх простота та доступність. Вони використовують стандартні символи та конструкції, такі як прямокутники для представлення дій, ромби для умов та стрілки для зв'язку між блоками. Це дозволяє легко створювати та читати блок-схеми незалежно від рівня технічної експертизи користувача. Іншою ж перевагою блок-схем є їх здатність до абстракції та спрощення складних процесів. Вони дозволяють представляти складні алгоритми та програмні функції у вигляді послідовності блоків, що допомагає зрозуміти логіку програми та виявити потенційні проблеми чи помилки. Блок-схеми дозволяють візуалізувати алгоритми перед їх реалізацією, що сприяє більш ефективній розробці та поліпшує якість програмного забезпечення. Крім того, блок-схеми допомагають узгоджувати роботу між розробниками та іншими зацікавленими сторонами. Вони служать зрозумілим засобом комунікації, що допомагає передавати інформацію про логіку програми, її

функціональність та взаємозв'язки між компонентами. Додатково, блок-схеми можуть слугувати основою для перетворення у реальний код. Вони допомагають перекласти логіку та алгоритми з більш високорівневого представлення на конкретну реалізацію у мові програмування. Блок-схеми дозволяють виявити потенційні помилки або неузгодженості в алгоритмах ще до їх конвертації у код, що сприяє зменшенню кількості помилок та полегшує налагодження програм.

Однак, блок-схеми також мають деякі недоліки. Одним з них є можливість недостатньої деталізації. У деяких випадках блок-схеми можуть бути надто загальними або не враховувати певні нюанси алгоритмів, що може призвести до неповного або некоректного розуміння програми. Тому важливо добре продумати деталі блок-схеми та врахувати всі аспекти алгоритму. Крім того, блок-схеми можуть стати недостатніми для моделювання складних систем або великих проектів. Вони можуть стати складними для розуміння та підтримки, особливо якщо програмний продукт має багато компонентів або великий обсяг функціональності. У таких випадках можуть використовуватись інші типи діаграм, такі як UML-діаграми або спеціалізовані моделювальні мови, для більш детального представлення системи. Можна сказати, що блок-схеми є ефективним інструментом у розробці програмного забезпечення. Вони дозволяють візуалізувати алгоритми та процеси, спрощують розуміння логіки програми та допомагають виявити потенційні проблеми. Загалом, блок-схеми є корисним інструментом, який сприяє якісній та структурованій розробці програмного забезпечення.

1.4 Постановка завдання дослідження

Аналіз блок-схем та діаграм дозволяє виявляти помилки та недоліки проектування програмного забезпечення на ранніх стадіях розробки. Це зменшує кількість помилок, які можуть бути виявлені на пізніших етапах

розробки, коли внесення змін є значно більш трудомістким і дорогим процесом.

Одним з методів, які можуть бути використані на основі аналізу блок-схем та діаграм, є автоматизована генерація коду. За допомогою цього методу можна створити початковий код програмного забезпечення на основі аналізу блок-схем та діаграм. Отже для вирішення цієї задачі необхідно дослідити можливості комбінування підходів до сегментації зображень за допомогою нейронних мереж та графових алгоритмів з метою досягнення більш високої точності та швидкодії. Для цього необхідно розглянути різні архітектури нейронних мереж, такі як Fully Convolutional Networks (FCN), U-Net, Mask R-CNN, а також різні графові алгоритми, такі як алгоритми на основі мінімального перерізу, алгоритми на основі співставлення графів, алгоритми на основі розбиття та злиття (split-merge) та інші.

Для досягнення мети дослідження потрібно виконати наступні завдання:

– провести огляд існуючих методів та алгоритмів сегментації зображень за допомогою нейронних мереж та графових алгоритмів;

– розробити комбіновану архітектуру, яка поєднає нейронні мережі та графові алгоритми для сегментації зображень;

– порівняти розроблену комбіновану архітектуру з існуючими методами за точністю та швидкодією на різних датасетах;

– зробити висновки щодо ефективності використання комбінованої архітектури для сегментації зображень та визначити перспективи подальшого розвитку даної технології.

2 СЕГМЕНТАЦІЯ ЗОБРАЖЕНЬ НЕЙРОННИМИ МЕРЕЖАМИ ТА ГРАФОВИМИ АЛГОРИТМАМИ

2.1 Повнозв’язна згорткова мережа (FCN)

Fully Convolutional Networks (FCN) [5] – це глибокі нейронні мережі, що використовуються для задач сегментації зображень. Ідея FCN полягає в тому, щоб замінити повністю зв'язані шари (fully connected layers) у звичайних глибоких нейронних мережах на шари згортки (convolutional layers). Основною перевагою FCN (рисунок 2.1) є те, що вони можуть приймати на вхід зображення будь-якого розміру та виходити з результатами сегментації такого ж розміру, як вхідне зображення. Завдяки цьому FCN можуть працювати з різними розмірами зображень, що є важливим для задач, де розмір зображення може змінюватися, наприклад, у відео або в зображеннях з дронів. У порівнянні з традиційними методами FCN здатні досягати високої точності в сегментації зображень за значно менший час. Також FCN можуть бути використані для задач розпізнавання образів, виявлення об'єктів, сегментації сцени, що робить їх дуже потужним інструментом у різноманітних задачах комп'ютерного зору.

Незважаючи на свої переваги, дана мережа також має деякі недоліки, які варто враховувати. Один з недоліків полягає у великій кількості обчислень, що потрібні для проходження через всю мережу. Це може призводити до високого споживання обчислювальних ресурсів та збільшення часу обробки. Іншим недоліком FCN є залежність від великої кількості навчальних даних. Для досягнення високої точності сегментації, FCN вимагає значної кількості розмічених зображень для тренування. Це може бути викликом в ситуаціях, коли наявність достатньої кількості навчальних даних є обмеженою. Крім того, FCN може мати обмежену здатність до точної сегментації дрібних деталей або об'єктів з низькою контрастністю. Відображення таких деталей може бути складним для FCN,

особливо якщо вони мають низьку інформативність або погано виділяються на фоні. Варто зазначити, що FCN може бути вразливою до зміщень та змін масштабу зображень. Відхилення від оригінального масштабу або невідповідність між вхідними зображеннями та навчальними даними можуть призвести до неточностей у сегментації та зміни форми об'єктів.

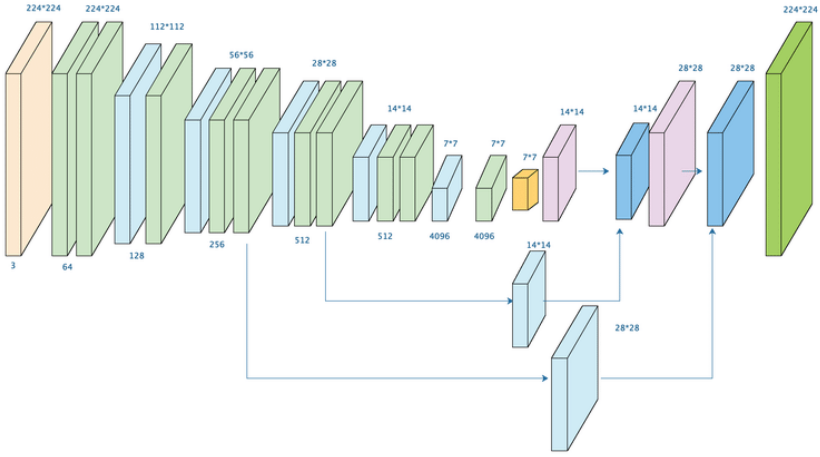

Рисунок 2.1 – Архітектура FCN.

2.2 U-Net

Найбільш популярною моделлю для сегментації зображень є U-Net. U-Net [6] є архітектурою нейронних мереж, спеціально розробленою для сегментації зображень, яка зазвичай використовується в біомедичному домені.

U-Net складається з енкодера та декодера (рисунок 2.2). Енкодер має типову архітектуру CNN, з кількома шарами з підвищенням рівня абстракції функцій. Кожен шар зменшує розмір зображення та збільшує кількість фільтрів. Декодер складається з блоків, які поєднують інформацію з енкодера та додаткову інформацію, що необхідна для відтворення вихідного розміру зображення. Основна ідея полягає в тому, щоб побудувати декодер, який має симетричну структуру до енкодера та використовує підвищення

рівня деталізації в зображенні. U-Net використовує пропускну зв'язку між відповідними шарами енкодера та декодера, що допомагає підвищити роздільну здатність сегментації. У звичайних CNN шарам потрібні багато шарів, щоб вони здатні вивчати складні закономірності в даних. Проте, це може призвести до втрати деталей при сегментації. U-Net вирішує цю проблему за рахунок збереження деталей на ранніх етапах виконання задачі та повторного їх використання на пізніших етапах.

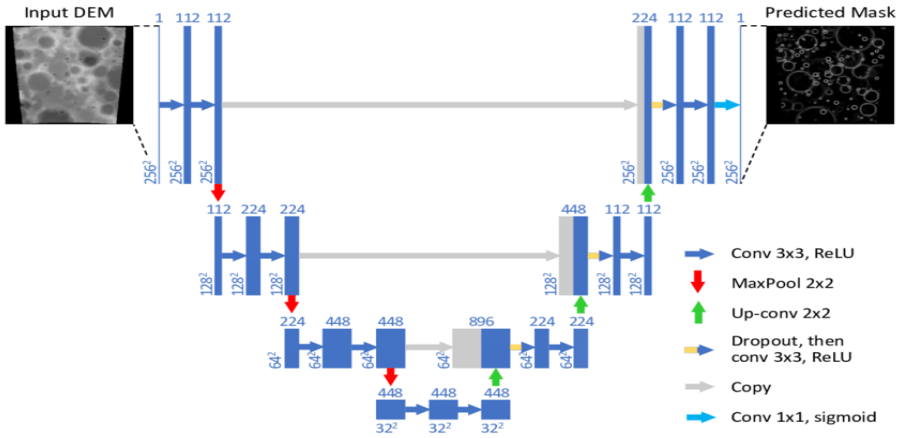

Рисунок 2.2 – Загальна архітектура моделі U-NET.

2.3 Graph Cut

Алгоритм на основі мінімального перерізу графу (англ.Graph Cut) – це метод сегментації зображень, який використовує графи для моделювання зображення та його сегментації. У цьому методі зображення представляється у вигляді графу, де кожен піксель має свій вузол, а ребра між вузлами відповідають схожості між пікселями. Задача полягає у тому, щоб розділити граф на дві частини (сегменти) так, щоб вартість розрізання графа була мінімальною (рисунок 2.3). Вартість розрізання графа визначається як сума вагів ребер, які розрізають граф. У задачі сегментації зображень, перша частина графу відповідає вузлам зображення, а друга частина – фону. Ребра між цими вузлами відповідають схожості між

пікселями. Метод на основі мінімального перерізу графу використовується для знаходження оптимального розрізання графа, щоб розділити зображення на об'єкти та фон. Цей метод знаходить широке застосування в сегментації зображень, зокрема в медичній діагностиці та обробці зображень, де важливо виділити об'єкти на зображенні з мінімальними помилками. Він також може бути використаний для сегментації зображень в інших галузях, де необхідно виділяти об'єкти на зображенні.

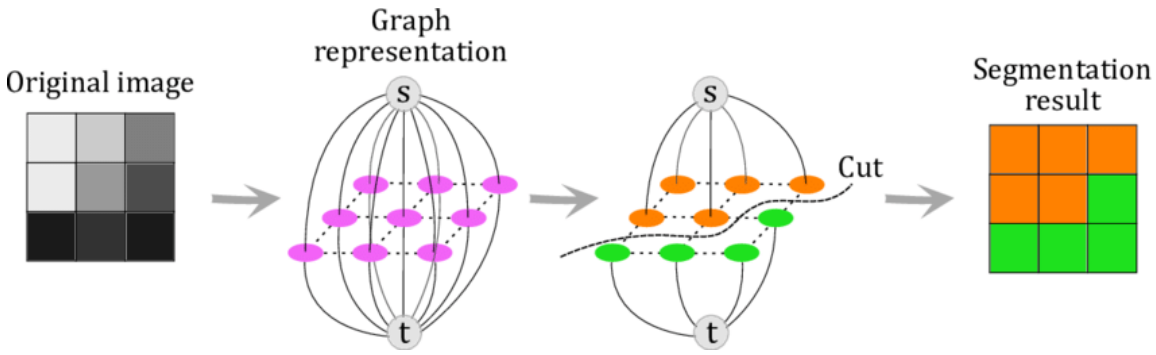

Рисунок 2.3 – Алгоритм перерізу графа.

Алгоритм Graph Cut володіє декількома перевагами. Перш за все, він дозволяє досягнути оптимального розбиття зображення на сегменти. Використовуючи графову модель, він знаходить найкращу розбивку зображення, яка мінімізує енергію або вартість сегментації. Це дає точний та надійний результат в порівнянні з іншими методами. Другою перевагою алгоритму Graph Cut є його здатність до обробки зображень з складними структурами та нечіткими межами об'єктів. Він враховує глобальну інформацію про зображення, таку як текстура, контекст та яскравість, для ефективного виділення об'єктів навіть у випадку перекривання та неоднорідностей. Третьою перевагою алгоритму Graph Cut є його гнучкість і можливість легко враховувати додаткові обмеження та контекстуальну інформацію. За допомогою введення вагових коефіцієнтів та приналежності пікселів до певних класів, можна виконати точну сегментацію з

урахуванням певних вимог та обмежень. Крім того, алгоритм Graph Cut є ефективним з обчислювальної точки зору. Його можна реалізувати з використанням оптимізованих структур даних та алгоритмів, що дозволяє виконувати сегментацію в режимі реального часу або обробляти великі зображення.

Незважаючи на свої переваги, алгоритм Graph Cut також має деякі недоліки, які варто враховувати. Перш за все, він може бути обчислювально вимогливим, особливо для великих зображень або складних графів. Обчислення енергії та знаходження оптимального розрізу можуть займати значний час, що обмежує його застосування у вимогливих за швидкістю системах. Також недоліком є алгоритму Graph Cut є його чутливість до початкового розбиття зображення. Початкові області, які вибираються для сегментації, можуть значно впливати на кінцевий результат. Якщо початкове розбиття неправильне або недостатньо точне, це може призвести до неправильної сегментації або втрати деталей. Важливо зазначити. що Graph Cut пов'язаний з обробкою зображень з великою кількістю перекриваючихся об'єктів або об'єктів з внутрішніми отворами. В таких випадках може бути важко правильно розподілити границі між об'єктами та врахувати їх внутрішню структуру. Отже, однією з його головних переваг є швидкість в порівнянні з іншими методами, що дозволяє використовувати його в реальному часі. Однак, він має обмеження в розв'язанні складних задач, де на зображенні присутні багато об'єктів.

2.4 Алгоритм Фелзенсвальба

Алгоритм Фелзенсвальба [7] – це алгоритм сегментації зображень, який був розроблений Фелзенсвальбом та Хатчісоном в 2004 році. Цей алгоритм працює зображеннями будь-якої роздільної здатності та не потребує попереднього навчання моделі. Принцип роботи алгоритму полягає в пошуку мінімальної ваги між сусідніми пікселями. Зображення

розбивається на декілька сегментів, починаючи з одного сегменту, де кожен піксель представляє один сегмент. На кожному кроці алгоритм розглядає сусідні пікселі та об'єднує два найбільш схожі пікселі у один сегмент. Вага між двома пікселями визначається як функція від різниці інтенсивності пікселів та відстані між ними. Процес злиття продовжується до тих пір, поки не буде досягнуто встановленого порогу ваги. Після цього сегменти, які утворилися, об'єднуються за допомогою злиття сегментів з найменшою вагою.

Однією з ключових переваг алгоритму Фелзенсвальба є його швидкодія. Він використовує ефективні структури даних та алгоритми, що дозволяють працювати з великими зображеннями у реальному часі. Це особливо важливо для застосувань, де швидкість обробки є критичним фактором, наприклад, в реальному часі системи моніторингу або відеоаналізу. Іншою перевагою алгоритму Фелзенсвальба є його здатність впоратися з варіативністю форм та розмірів об'єктів. Він може ефективно виділити як однорідні об'єкти зі змінною формою, так і об'єкти різних розмірів, включаючи малий деталі або великі структури. Крім того, алгоритм Фелзенсвальба відносно стійкий до шуму та змін в освітленні. Він може ефективно працювати навіть у випадку наявності шуму на зображенні або відхилень в яскравості. Це робить його корисним інструментом для застосування в умовах реального світу, де зображення можуть бути піддані різноманітним спотворенням.

Незважаючи на свої переваги, алгоритм Фелзенсвальба також має певні недоліки, які варто враховувати. Перш за все, він може бути чутливим до параметрів, таких як порігові значення, використовувані для обчислення меж розривів. Неправильний вибір цих параметрів може призвести до неправильної сегментації або недостатньої деталізації об'єктів.

Крім того, алгоритм Фелзенсвальба може мати обмежену ефективність при обробці дуже складних зображень з багатомірними структурами або з великою кількістю перекриваючихся об'єктів. В таких

випадках може виникнути проблема з правильним розподілом границь розривів та виділенням відокремлених сегментів. Також варто враховувати, що алгоритм Фелзенсвальба не завжди здатний точно виділити об'єкти з м'якими або нечіткими межами. В деяких випадках він може вважати такі об'єкти частинами більших сегментів або розглядати їх як шумові компоненти, що призводить до неправильної сегментації. Однією з основних проблем алгоритма Фелзенсвальба є його обчислювальна складність. В залежності від розміру зображення і використовуваних параметрів, алгоритм може вимагати значних обчислювальних ресурсів та тривати значний час. Це може бути недоцільним для застосувань, де важлива швидкість обробки в реальному часі.

Підсумовуючи, Алгоритм Фелзенсвальба є ефективним та швидким алгоритмом сегментації зображень, який може бути використаний для обробки великих обсягів даних. Однак, він не завжди генерує оптимальні результати, тому є потреба у подальшій оптимізації та налаштуванні параметрів.

2.5 Комбіновані алгоритми

Комбінований підхід, який використовує як нейронні мережі, так і графові алгоритми, стає все більш популярним у задачах сегментації зображень. Це пояснюється тим, що жоден із цих підходів не є універсальним і може бути кращим у визначених ситуаціях. Однією з переваг комбінованих алгоритмів є здатність до покращення точності прогнозування. Комбінація декількох алгоритмів може допомогти вирішити проблеми, з якими окремі алгоритми можуть не справитись. Кожен алгоритм може вносити свою унікальну перспективу та здатність до виявлення певних закономірностей або шаблонів в даних, що призводить до поліпшення загальної точності моделі. Крім того, комбіновані алгоритми можуть бути більш стійкими до перенавчання. Оскільки вони

використовують кілька базових алгоритмів, їх важче перенавчити на конкретних даних. Це дозволяє підвищити загальну універсальність та здатність моделі до узагальнення на нові дані. Комбіновані алгоритми також можуть бути ефективними при розв'язанні складних задач. Шляхом комбінування різних алгоритмів з різними характеристиками та підходами, можна створити адаптивну систему, яка здатна краще впоратися зі складними сценаріями або невизначеністю.

В комбінованому підході нейронні мережі використовуються для екстракції ознак зображення, а графові алгоритми застосовуються для отримання окремих сегментів зображення. Зазвичай, нейронні мережі використовуються для розпізнавання об'єктів на зображенні, а графові алгоритми – для обробки відповідних міток, які були надані нейронною мережею. Один з прикладів комбінованого підходу – Mask R-CNN [8]. Це нейронна мережа, яка базується на архітектурі R-CNN і містить додатковий модуль, який згенерує маску для кожного об'єкта на зображенні. Цей модуль використовує графові алгоритми для генерації точної маски для кожного об'єкту. Інший приклад комбінованого підходу – Deep Watershed Detector (DWD) [9]. Цей підхід використовує нейронні мережі для створення векторів ознак, що репрезентують кожен піксель зображення. Потім, вектори ознак використовуються для графової сегментації за допомогою алгоритму на основі мінімального перерізу графу.

2.6 Порівняльний аналіз алгоритмів

Порівняння нейронних та графових алгоритмів в задачі сегментації зображень можна провести з кількох точок зору.

Переваги нейронних мереж:

– завдяки своїй архітектурі, FCN/U-Net мають високу точність в задачах сегментації зображень;

– нейронні мережі дозволяють автоматично визначати різноманітні

функції, необхідні для сегментації, які складно визначити вручну;

– моделі можуть бути треновані на великій кількості даних, що підвищує їхню точність;

– нейронні мережі можуть працювати швидше, ніж графові алгоритми.

Переваги графових алгоритмів:

– графові алгоритми можуть бути ефективними для сегментації зображень, якщо об'єкти на зображенні мають чітко визначені контури або границі;

– графові алгоритми мають відносно просту інтерпретацію, що може бути корисно при аналізі результатів;

– графові алгоритми можуть бути використані для вирішення інших задач, таких як виявлення об'єктів та визначення їхнього положення на зображенні.

Недоліки нейронних мереж:

– нейронні мережі потребують великої кількості даних для тренування, що може бути проблемою у випадках, коли дані складно отримати;

– точність FCN/U-Net залежить від якості даних для тренування, тому при недостатній кількості даних, їхній результат може бути неадекватним;

– нейронні мережі є чорною скринькою, тобто неможливо зрозуміти, як саме мережа приймає рішення та робить прогнози. Це може бути проблемою у деяких доменних задачах, де потрібно пояснити результати моделі.

Недоліки графових алгоритмів:

– графові алгоритми не завжди ефективні для сегментації зображень зі складними формами або зображень зі значними перекриттями об'єктів;

– більшість графових алгоритмів працюють з дискретними даними, такими як сірі значення пікселів, що обмежує їхню точність при роботі з зображеннями високої якості.

Fully Convolutional Networks (FCN) та U-Net є найпоширенішими нейронними архітектурами, що застосовуються для сегментації зображень. Обидві архітектури здатні забезпечити високу точність сегментації зображень, проте U-Net має деякі переваги у випадку, коли розмір об'єктів сегментації значно менший, ніж розмір зображення в цілому.

Графові алгоритми, такі як алгоритм на основі мінімального перерізу та алгоритм на основі розбиття та злиття, засновані на обробці графів зображень, що дає можливість використовувати геометричну та топологічну інформацію для сегментації. Проте, ці алгоритми мають деякі обмеження, такі як обробка зображень лише з деякими типами розподілу яскравості, чутливість до параметрів та складність реалізації.

Комбінація нейронних мереж та графових алгоритмів є перспективним підходом, що дає можливість поєднати переваги обох методів та зменшити їх недоліки. Однак, такий підхід потребує додаткових зусиль у розробці та налаштуванні.

3 ОБҐРУНТУВАННЯ ПРИЙНЯТИХ ТЕХНОЛОГІЙ

3.1 Мова програмування Python

Для використання в проекті було обрано Python, бо це потужна та популярна високорівнева мова програмування, яка набула широкого визнання у галузях програмування, наукових досліджень та розвитку штучного інтелекту. Однією з головних переваг Python є його простий і зрозумілий синтаксис, що дозволяє розробникам писати зрозумілий і лаконічний код. Це робить Python доступним як для початківців, так і для досвідчених програмістів. Дана мова програмування має велику кількість потужних бібліотек, які розширюють її можливості. Наприклад, бібліотека NumPy надає потужні інструменти для роботи з числовими даними та обчисленнями, а бібліотека Pandas дозволяє ефективно маніпулювати та аналізувати даними в табличному форматі. Крім того, бібліотека Matplotlib дозволяє візуалізувати дані у вигляді графіків та діаграм, спрощуючи розуміння інформації.

Python також має широку спільноту розробників, що приносить численні переваги. Велика спільнота означає наявність багатьох ресурсів, документації та прикладів коду, що сприяє швидкому розв'язанню проблем та вирішенню складних завдань. Крім того, спільнота розробників активно підтримує та розвиває мову, випускаючи нові версії та вдосконалення.

Вибір мови програмування для дослідження методів аналізу графів для вирішення задач комп'ютерного зору є важливим рішенням. Однак, Python є однією з найбільш переконливих виборів з причин декількох факторів. Дуже великою перевагою є те, що Python має широкий вибір потужних бібліотек, спеціалізованих на аналізі графів та комп'ютерному зорі. Наприклад, NetworkX є потужним інструментом для роботи з графами, надаючи різноманітні функції для конструювання, візуалізації та аналізу графів. Окрім того, існують такі бібліотеки, як OpenCV, які спеціалізуються

на обробці зображень і відео, що знадобляться в задачах комп'ютерного зору. Крім того, Python має простий і зрозумілий синтаксис, що робить його доступним для початківців і досвідчених розробників. Це дозволяє швидко приступити до роботи з алгоритмами аналізу графів та комп'ютерного зору без значних зусиль. А також, Python є переносимою мовою програмування, що означає, що ваш код можна запускати на різних операційних системах без необхідності внесення значних змін.

3.2 PyTorch

PyTorch – це відкрите програмне забезпечення для машинного навчання і глибокого навчання, яке забезпечує високорівневі API для побудови та тренування нейронних мереж. Одна з його головних переваг полягає в його динамічному обчислювальному графі. Відмінність PyTorch від багатьох інших фреймворків полягає в тому, що він використовує концепцію "define-by-run". Це означає, що граф обчислень формується під час виконання коду, що забезпечує більшу гнучкість та деталізацію при визначенні моделей та обчислювальних операцій. Завдяки цьому, розробники можуть використовувати умовні оператори, цикли та інші структури мови Python без необхідності попереднього компілювання графа. PyTorch також надає багатий набір функцій для автоматичного диференціювання, що дозволяє зручно обчислювати градієнти для оптимізації моделей машинного навчання. Це особливо корисно при тренуванні глибоких нейронних мереж, де обчислення градієнтів є важливою складовою процесу навчання. Також даний фреймворк має багато інших корисних функцій, таких як підтримка розподіленого навчання, вбудована підтримка роботи з графічними процесорами (GPU), а також велику активну спільноту розробників, яка постійно робить внески у його вдосконалення та розвиток.

Серед інших переваг слід зазначити те, що інтерфейс PyTorch робить

його дружнім для розробників. Під час програмування з використанням PyTorch, розробники можуть ефективно використовувати концепції об'єктно-орієнтованого програмування, такі як успадкування та поліморфізм. Це дозволяє створювати модульні та перевикористовувані компоненти, спрощуючи розробку складних моделей. Ще однією перевагою PyTorch є його гнучкість та розширюваність. Завдяки підтримці Python, можна використовувати всі можливості мови програмування, включаючи широкий спектр бібліотек для наукових обчислень та обробки даних. Крім того, PyTorch дозволяє інтегрувати власні операції та модулі, що сприяє розширенню функціональності фреймворку для вирішення специфічних завдань. Також важливо зазначити, що одна з суттєвих переваг PyTorch полягає в його сумісності з бібліотекою NumPy. Це дозволяє легко і ефективно переміщувати дані між двома фреймворками, що дозволяє використовувати існуючий код на NumPy у проектах, побудованих на PyTorch, та навпаки.

Однак, PyTorch, як і будь-який інший інструмент, має свої недоліки, які варто враховувати при використанні фреймворку для розробки машинного навчання і глибокого навчання. Один з недоліків PyTorch полягає в його швидкості порівняно з іншими фреймворками. Незважаючи на те, що PyTorch зазнає постійних покращень у цьому напрямку, інші фреймворки, такі як TensorFlow, можуть мати кращу оптимізацію та продуктивність, особливо при роботі з великими обсягами даних або розподіленими системами. Іншим недоліком PyTorch є менша кількість готових моделей та архітектур порівняно з іншими популярними фреймворками. Оскільки PyTorch є молодшим гравцем на ринку, йому може бракувати встановлених моделей та ресурсів порівняно з іншими великими екосистемами. Однак ця проблема зменшується з часом, оскільки спільнота розробників PyTorch активно зростає та розширює набір доступних моделей. Крім того, недоліком PyTorch може бути складність використання на розподілених системах або в обмежених ресурсах. В деяких випадках,

робота з багатопоточними або розподіленими обчисленнями може вимагати додаткових зусиль та розуміння особливостей фреймворку. Нарешті, через свою гнучкість та динамічність, PyTorch може вимагати більше уваги до деталей та контролю з боку розробника. Динамічний граф обчислень може збільшити ризик помилок та зробити процес налагодження складнішим.

Незважаючи на ці недоліки, PyTorch залишається значним та популярним фреймворком для машинного навчання і глибокого навчання, який продовжує залучати велику спільноту розробників та продовжує розвиватися з кожним релізом.

3.3 Програмне забезпечення PyTorch Geometric

PyTorch Geometric (PyG) – це відкрите програмне забезпечення, яке надає ефективні інструменти для обробки та аналізу графових структур в машинному навчанні. Воно побудоване на базі фреймворку PyTorch і надає набір функцій та класів, спеціально розроблених для роботи з графами. PyTorch Geometric має кілька ключових компонентів, які роблять його потужним інструментом для аналізу графів. Воно надає широкий спектр графових операторів, таких як агрегація вузлів, редукція графів, згортки на графах та багато інших. Ці оператори дозволяють зручно виконувати різні операції з графами, необхідні для задач, пов'язаних з машинним навчанням.

Крім того, PyTorch Geometric надає зручний інтерфейс для завантаження та обробки даних графів. Він має вбудовану підтримку для різних форматів даних, таких як GraphML, GML, TSV та інші, що спрощує роботу з реальними графовими даними. Крім того, він надає можливість перетворювати графи на тензори PyTorch для використання в моделях глибокого навчання. Іншою важливою функціональністю PyTorch Geometric є підтримка графових нейронних мереж (Graph Neural Networks, GNNs). Воно надає набір готових архітектур та шарів GNN, які можна використовувати для побудови та тренування моделей, що працюють з

графами. Це дозволяє легко використовувати сучасні методи аналізу графів у своїх проектах. Також PyTorch Geometric підтримує використання графічних процесорів (GPU) для прискорення обчислень. Використання GPU дозволяє значно збільшити швидкодію обробки графів та навчання моделей на великих обсягах даних. Це особливо корисно при роботі з великими графами та складними архітектурами GNN. Важливим є наявність активної спільноти розробників, що сприяє постійному розвитку та підтримці фреймворка. Розробники активно вносять нові функції, виправляють помилки та надають документацію для полегшення роботи з PyTorch Geometric. Це забезпечує користувачам доступ до нових можливостей, а також можливість обміну знаннями та досвідом з іншими учасниками спільноти.

Крім того, PyTorch Geometric дозволяє поєднувати переваги PyTorch, такі як простота використання та гнучкість, з спеціалізованими функціями для графів. Це дозволяє дослідникам та розробникам зосередитися на самому алгоритмі аналізу графів, не витрачаючи багато часу на розробку та оптимізацію базових операцій. У підсумку, PyTorch Geometric є потужним інструментом для аналізу графів у контексті машинного навчання. Він надає широкий набір функціональностей, підтримку GPU, активну спільноту розробників та поєднує переваги фреймворку PyTorch зі спеціалізованими функціями для графів. Це робить його привабливим вибором для дослідження та розробки методів аналізу графів для розв'язання задач комп'ютерного зору.

При використанні PyTorch Geometric слід враховувати деякі його недоліки, які можуть впливати на процес розробки та роботу з графовими данними. Один з недоліків PyTorch Geometric полягає в обмеженому наборі підтримуваних графових операцій та алгоритмів порівняно з іншими бібліотеками. Хоча PyTorch Geometric надає багато функціональності для обробки графів, він може не мати деяких спеціалізованих алгоритмів, які можуть бути потрібні для конкретних дослідницьких або реальних задач.

Іншим недоліком є складність використання PyTorch Geometric для новачків у графовому аналізі. Інструмент має дещо крутішу криву навчання, оскільки для ефективного використання фреймворку потрібно мати розуміння основних концепцій та вміти працювати з PyTorch. Це може бути великою витратою часу для початківців або тих, хто має обмежений досвід з машинним навчанням. Крім того, PyTorch Geometric може мати проблеми з продуктивністю при роботі з великими графами або при використанні на обмежених ресурсах. В деяких випадках обробка великих обсягів даних або використання складних моделей може призводити до великої споживання пам'яті або тривалих часів обчислення. Це може вплинути на швидкодію та масштабованість додатку.

3.4 Бібліотека OpenCV

OpenCV (Open Source Computer Vision Library) є однією з найпопулярніших та широко використовуваних бібліотек комп'ютерного зору у світі. Вона розроблена з метою надання відкритих та ефективних інструментів для обробки зображень та відео, а також для розв'язання різноманітних задач у галузі комп'ютерного зору. Однією з основних переваг OpenCV є його кросплатформовість. Він підтримує різні операційні системи, включаючи Windows, macOS, Linux, а також мобільні платформи, такі як Android та iOS. Це дозволяє розробникам створювати програми комп'ютерного зору для різних пристроїв та платформ, що забезпечує більш широку аудиторію та можливості розповсюдження.

OpenCV надає багатий набір функцій та алгоритмів, що дозволяють виконувати різноманітні завдання комп'ютерного зору. Він включає в себе інструменти для обробки зображень, такі як фільтрація, виокремлення контурів, розмиття, корекція кольору та інші. Крім того, він підтримує роботу з відео, включаючи захоплення відео з камери, відтворення, аналіз руху, відстеження об'єктів та багато інших завдань.

OpenCV також пропонує підтримку глибокого навчання, включаючи інтеграцію з популярними фреймворками, такими як TensorFlow та PyTorch. Це дозволяє використовувати потужні моделі глибокого навчання для розпізнавання об'єктів, класифікації, сегментації та інших завдань комп'ютерного зору. Завдяки цьому, OpenCV стає універсальним інструментом для вирішення різноманітних завдань у галузі комп'ютерного зору.OpenCV також має широкий спектр підтримуваних форматів зображень та відео, включаючи популярні стандарти, такі як JPEG, PNG, TIFF, MPEG і багато інших. Це дозволяє безпроблемно працювати з різними типами медіа та забезпечує сумісність з іншими програмними засобами. Помітною перевагою OpenCV є його велика активна спільнота розробників та користувачів. Це означає, що є велика кількість ресурсів, таких як форуми, блоги та онлайн-курси, які допомагають користувачам знайти відповіді на свої питання, вирішити проблеми та навчитися використовувати бібліотеку ефективно. Крім того, наявність спільноти сприяє постійному розвитку та вдосконаленню OpenCV, що робить його більш сучасним і забезпечує відповідь на сучасні вимоги та тенденції у галузі комп'ютерного зору.

Однак, варто зазначити, що OpenCV має деякі обмеження. Одним з недоліків OpenCV є його складність у використанні для новачків у галузі комп'ютерного зору або програмування загалом. Бібліотека має велику кількість функцій, класів та методів, що можуть бути переповненістю для початківців. Розуміння правильного використання цих функцій та їх взаємодії може вимагати значного часу та досвіду. Іншим недоліком є відсутність вбудованої підтримки для глибокого навчання. В той час як OpenCV надає багатий набір функцій для обробки зображень та відео, він не пропонує вбудованих інструментів для розробки та використання глибоких нейронних мереж. Для таких завдань доведеться використовувати інші фреймворки або бібліотеки. Також варто відзначити, що OpenCV може мати обмежену швидкодію при обробці великих обсягів даних або при

використанні складних алгоритмів. Це особливо помітно, коли робота зображень або відео вимагає інтенсивних обчислень. У таких випадках може знадобитися оптимізація або використання більш продуктивних інструментів. А також, бібліотека OpenCV може мати деякі обмеження в підтримці нових технологій та стандартів. У зв'язку зі швидким розвитком галузі комп'ютерного зору, нові методи та алгоритми можуть з'являтися швидше, ніж вони стають доступними у OpenCV. Це може обмежити можливості використання найновіших технологій у проектах, особливо якщо потрібна спеціалізована функціональність. Але незважаючи на ці недоліки, кількість переваг значно покращує ситуацію, тож подальше використання даної бібліотеки є виправданим рішенням.

3.5 Бібліотека Numpy

Numpy (Numerical Python) – це потужна бібліотека для наукових обчислень у мові програмування Python. Вона надає високопродуктивні структури даних, такі як масиви та матриці, а також велику колекцію математичних функцій, що дозволяють виконувати широкий спектр обчислень.

Однією з ключових переваг Numpy є його підтримка для багатовимірних масивів. Він дозволяє створювати та маніпулювати масивами даних різного розміру та форми. Це особливо корисно для наукових та чисельних обчислень, де великі обсяги даних можуть бути легко представлені та опрацьовані. Ще однією важливою перевагою Numpy є швидкодія. Бібліотека написана на C, що дозволяє виконувати обчислення на низькорівневому рівні та використовувати оптимізовані алгоритми. Це дозволяє прискорити обробку даних та забезпечити ефективне використання ресурсів комп'ютера.

Numpy також надає багато математичних функцій і операцій, що спрощують обчислення. Він підтримує векторизацію, що дозволяє

виконувати операції на цілих масивах даних, що є більш ефективним та зручним способом програмування. Завдяки цьому, Numpy забезпечує широкий спектр функціональності для наукових обчислень, статистики, обробки сигналів, обробки зображень та багатьох інших областей.

Окрім того, Numpy є основою для багатьох інших бібліотек, таких як Pandas, Matplotlib та SciPy, що робить його незамінним інструментом для наукових обчислень в Python. Крім того, Numpy має вбудовану підтримку векторизованих обчислень, що дозволяє прискорити виконання обчислень шляхом заміни циклів на векторні операції. Також варто зазначити, що Numpy є відкритим програмним забезпеченням з вільною ліцензією BSD, що дозволяє використовувати його в комерційних проектах та розповсюджувати вільно безкоштовно. Узагальнюючи, Numpy є потужним інструментом для наукових обчислень в Python, та пропонує високу ефективність, широкі можливості роботи з масивами даних та багато математичних функцій для роботи з даними.

Незважаючи на свої переваги, Numpy також має деякі недоліки. Один з них полягає у тому, що використання Numpy може бути складним для новачків у програмуванні або наукових обчисленнях. Він вимагає певного розуміння концепцій масивів та векторизації, а також використання специфічного синтаксису. Це може становити виклик для тих, хто тільки починає працювати з Numpy. Інший недолік полягає в тому, що Numpy має фіксований розмір масивів при їх створенні. Це означає, що ви повинні заздалегідь визначити розмір масиву, і його зміна може бути обтяжливою або неможливою. Це може створювати проблеми, коли масив потребує динамічних змін розміру в процесі обчислень. Також варто відзначити, що Numpy не підтримує автоматичне розподілення пам'яті та управління пам'яттю. Це означає, що ви самі відповідаєте за вивільнення пам'яті, коли вона більше не потрібна. Неправильне управління пам'яттю може призвести до витоку пам'яті або неочікуваного зменшення продуктивності. Ще одним з найбільших недоліків є те, що NumPy не підтримує автоматичну

диференціацію, тобто автоматичний розрахунок градієнту функції. Це робить NumPy менш практичним для використання в глибокому навчанні та нейромережах, де градієнти мають ключове значення. Також недоліком є те, що NumPy не підтримує гнучке відстеження стану, що може бути проблемою для складних обчислювальних графів і різноманітних операцій з масивами. Також, якщо масиви даних занадто великі, може виникнути проблема з обмеженням пам'яті, оскільки NumPy масштабується не настільки добре, як деякі інші бібліотеки. Крім того, іноді NumPy може бути менш ефективним для обробки даних з більш високими рівнями складності, якщо порівнювати зі спеціалізованими бібліотеками, такими як pandas або Dask. Вони мають більш високий рівень абстракції, що дозволяє працювати з даними на більш високому рівні, що забезпечує зручність роботи.

Однак, в цілому, незважаючи на ці недоліки, NumPy залишається важливою і корисною бібліотекою для багатьох застосувань в галузі науки про дані та машинного навчання.

3.5 Hugging Face

Hugging Face [10] – це компанія та відкрита спільнота, яка спеціалізується на розробці та наданні інструментів і бібліотек для роботи з природними мовами (NLP). Вони відомі своїми інноваційними рішеннями та внеском у галузь глибокого навчання та NLP. Крім того, Hugging Face активно працює над розробкою інструментів для побудови та навчання моделей NLP. Вони надають інтерфейси та розширення для взаємодії з популярними фреймворками глибокого навчання, такими як TensorFlow та PyTorch, спрощуючи процес розробки та експериментування з моделями. Окрім того, Hugging Face створили платформу для спільного використання моделей та даних, відому як Hugging Face Hub. Ця платформа дозволяє дослідникам та розробникам обмінюватися моделями, даними та кодом, сприяючи колективному розвитку і покращенню моделей NLP. Загалом,

Hugging Face є визнаним лідером в галузі NLP та глибокого навчання, завдяки своїм інноваційним рішенням, доступній бібліотеці Transformers та платформі Hugging Face Hub.

Головною продуктовою лінією Hugging Face є бібліотека Transformers, яка надає розширений функціонал для розробки, навчання та застосування моделей глибокого навчання в області NLP. Однією з основних переваг бібліотеки Transformers є наявність передварчо навчених моделей, які можна завантажити та використовувати для вирішення різноманітних NLP завдань. Це дозволяє використовувати гарно навчені моделі, які здатні досягати вражаючих результатів на багатьох завданнях без необхідності великого обсягу даних та тривалого процесу навчання. Бібліотека надає легкий доступ до цих моделей і дозволяє зручно використовувати їх для класифікації тексту, машинного перекладу, витягування інформації та багатьох інших завдань. Крім попередньо навчених моделей, Transformers також забезпечує інтерфейси для розробки та навчання власних моделей NLP. Вона підтримує популярні фреймворки глибокого навчання, такі як TensorFlow та PyTorch, що дозволяє розробникам працювати з відомими інструментами та використовувати їх для побудови та навчання власних моделей NLP з використанням сучасних архітектур та методів. Також варто відзначити активну спільноту, яка оточує бібліотеку Transformers. Hugging Face надає платформу Hugging Face Hub, де дослідники та розробники можуть обмінюватися моделями, даними та кодом, сприяючи спільному розвитку та покращенню моделей NLP. Ця спільнота активно обговорює нові ідеї, ділиться знаннями та надає підтримку одне одному. Такий колективний підхід допомагає прискорити процес розвитку та забезпечує більш широке застосування моделей NLP на різних завданнях.

Також однією з вагомих переваг бібліотеки Transformers є його гнучкість та легкість використання. Вона надає високорівневі API для швидкого інтегрування моделей NLP в існуючі проекти. Крім того, багато

функціоналу, такого як обробка тексту, векторизація, візуалізація тощо, доступні в зручних інтерфейсах, що спрощує роботу з даними та моделями. Необхідно також відзначити широку підтримку та постійне оновлення бібліотеки. Hugging Face активно розвиває та підтримує Transformers, що включає виправлення помилок, вдосконалення функцій та випуск нових моделей та алгоритмів. Це гарантує користувачам наявність сучасних рішень та доступ до новітніх досягнень в галузі NLP.

Загалом, бібліотека Transformers від Hugging Face вирізняється своєю потужністю, гнучкістю та активною спільнотою. Вона надає зручний інструментарій для роботи з моделями NLP, дозволяючи дослідникам та розробникам швидко розвивати та впроваджувати інноваційні рішення в галузі природної мови.

Незважаючи на багато переваг, бібліотека Transformers також має свої недоліки. Перш за все, вона вимагає великого обсягу пам'яті для роботи з великими моделями NLP. Це може стати проблемою для ресурсоємних проектів або обмежених обчислювальних середовищ. Крім того, тренування та налаштування моделей з використанням бібліотеки Transformers можуть бути складними для новачків у галузі NLP. Бібліотека надає багато функцій та можливостей, що може вимагати глибокого розуміння алгоритмів та методів машинного навчання.

Крім того, моделі, навчені з використанням бібліотеки Transformers, можуть бути часом вимогливими до обчислювальних ресурсів при виконанні в реальному часі. Для деяких додатків, особливо на вбудованих пристроях або з обмеженими обчислювальними можливостями, це може бути проблемою. Також варто відзначити, що навчання нових моделей з використанням бібліотеки Transformers може бути тривалим та вимагати значних обчислювальних ресурсів. Для деяких дослідників та розробників це може стати обмежуючим фактором при експериментах та швидкому прототипуванні. Нарешті, важливо відзначити, що не всі моделі або методи NLP підтримуються бібліотекою Transformers. Хоча вона надає широкий

спектр моделей, деякі спеціалізовані архітектури або підходи можуть бути недоступними через обмеження бібліотеки.

Однак, зважаючи на багато переваг, ці недоліки бібліотеки Transformers можна вважати дрібними порівняно з її загальними можливостями та користувацьким досвідом, які вона надає. Більшість з них можуть бути усунені або зменшені шляхом належного налаштування обчислювальних ресурсів, оптимізації коду або вибору відповідних моделей залежно від конкретних вимог проекту. Крім того, спільнота розробників і дослідників, пов'язана з бібліотекою Transformers, активно веде роботу над усуненням недоліків та постійно вдосконалює функціонал. Через відкритий характер проекту, користувачі можуть активно вносити свої пропозиції та внести внесок у поліпшення бібліотеки.

Отже, незважаючи на певні недоліки, бібліотека Transformers є потужним інструментом для розвитку та застосування моделей NLP. Вона надає широкий спектр можливостей, має активну спільноту та постійно оновлюється. З правильним вибором моделей, належним налаштуванням та розумінням її особливостей, бібліотека Transformers може ефективно використовуватися для різних завдань в галузі обробки природної мови.

3.6 NetworkX

Бібліотека NetworkX [11] є потужним інструментом для аналізу та моделювання графів. Вона забезпечує зручний та ефективний спосіб представлення графів та виконання різних операцій на них.

У рамках дослідження з аналізу зображень блок-схем, ми використовуємо бібліотеку NetworkX для аналізу та моделювання графів, що представляють блоки та лінії на зображенні блок-схеми. Зокрема, ми використовуємо NetworkX для побудови графу, який представляє блок-схему у вигляді графу, де вершинами є блоки та лінії, а ребрами – їх зв'язки.

Однією з ключових переваг бібліотеки NetworkX є її гнучкість та можливості розширення. Вона надає багато різноманітних алгоритмів та функцій для аналізу та моделювання графів, таких як зведення графу, знаходження шляхів, агрегація ознак та інші. Крім того, бібліотека має добре документований та зрозумілий API, що робить її досить простою у використанні. Окрім використання NetworkX для побудови графу блок-схеми, ми також можемо використовувати її для аналізу та моделювання різних інших графових структур, пов'язаних з аналізом зображень, наприклад, графів розрізнення та графів залежностей між об'єктами на зображенні.

4 ПРОВЕДЕНІ ЕКСПЕРИМЕНТИ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ

В цьому розділі буде розглянута програмна реалізація та проведенні експерименти. Основною метою роботи є дослідження використання графових алгоритмів та структур з метою покращення процесу сигментації зображень. В свою чергу додатково буде розглянута можливість використання графів для вирішення задачі автоматичної генерації коду з діаграм, шляхом застосування графових [12] структур на кожному з окремих етапів розробки (рисунок 4.1).

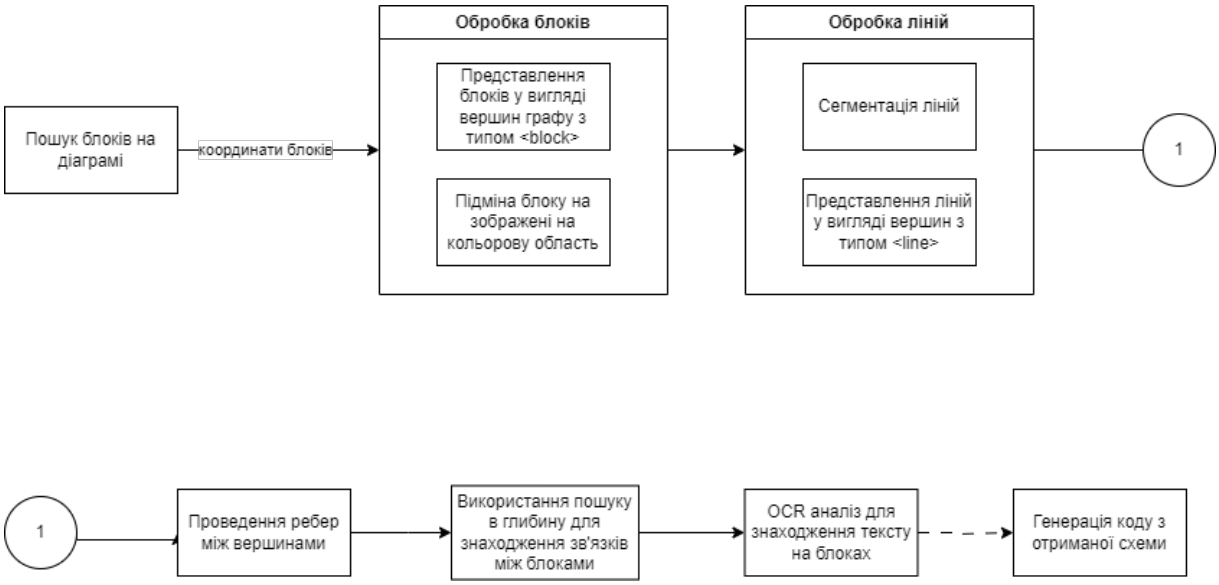

Рисунок 4.1 – Загальна схема алгоритму

4.1 LambdaNetworks для інстансної сегментації зображень

LambdaNetworks [13] – це нова та перспективна архітектура нейромережі, яка використовується для завдань інстансної сегментації зображень. Вона була запропонована в 2021 році та відрізняється від традиційних підходів до сегментації зображень.

Однією з ключових особливостей LambdaNetworks є використання графових операторів для забезпечення ефективного оброблення зображень

з великою кількістю пікселів (рисунок 4.2). Замість звичайних згорток, які працюють з локальними контекстами, LambdaNetworks використовує графові оператори, що дозволяє моделі працювати з глобальним контекстом зображення.

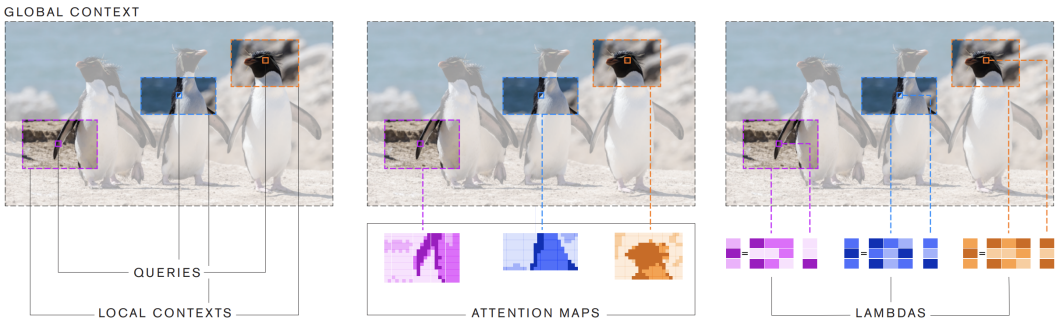

Рисунок 4.2 – Демонстрація алгоритму LambdaNetworks

Це дозволяє знизити кількість параметрів та обчислювальний затрати моделі, забезпечуючи при цьому високу точність сегментації, а також дозволяє моделі виявляти складні зв'язки та особливості на різних масштабах та положеннях, що сприяє точній інстансній сегментації. Іншою важливою особливістю LambdaNetworks є використання самоуваги (self-attention) для взаємодії між пікселями зображення. Самоувага дозволяє моделі віддавати більшу увагу значущим особливостям зображення та забезпечує взаємодію між далекими пікселями, що покращує здатність моделі розпізнавати об'єкти та проводити точну сегментацію. Крім того, LambdaNetworks може ефективно використовуватися для сегментації зображень з різною кількістю об'єктів. Вона може адаптуватися до змінюючого числа об'єктів та зберігати точність сегментації навіть при наявності різних розмірів та форм об'єктів на зображенні.

Незважаючи на свої переваги, LambdaNetworks також має кілька недоліків. Один з недоліків полягає в тому, що LambdaNetworks є новою архітектурою, і її використання може бути обмеженим порівняно з більш

встановленими та поширеними моделями. Відсутність широкого сприйняття та підтримки можуть обмежити доступність зовнішніх ресурсів, навчальних матеріалів та пакетів з високорівневим інтерфейсом. Однак, незважаючи на ці недоліки, LambdaNetworks є потужною моделлю, яка може бути використана для розв'язання поставленої задачі інстансної сегментації зображень. Її здатність працювати з глобальним контекстом та виявляти складні зв'язки на різних масштабах та положеннях робить її ефективним інструментом для точного виділення об'єктів на зображенні.

4.2 LambdaNetworks для object detection

Як було зазначено, основна перевага LambdaNetworks полягає в їх здатності до захоплення довгострокових залежностей та моделювання контексту на зображеннях. Це досягається шляхом використання графових операторів, які можуть обробляти інформацію залежно від відносин між вершинами графа. Таким чином, LambdaNetworks здатні ефективно виявляти об'єкти та виконувати класифікацію на зображеннях з високою точністю.

Крім того, LambdaNetworks мають декілька інших переваг для задачі виявлення об'єктів. Вони використовують графові оператори, що дозволяє їм захоплювати довгострокові залежності між пікселями, що поліпшує точність виявлення об'єктів зі складною структурою або контекстом. Графова структура дозволяє моделювати взаємозв'язки між об'єктами та їхніми оточуючими пікселями, що сприяє кращому розумінню семантики об'єктів на зображенні.

LambdaNetworks є гнучкими і можуть бути легко модифіковані та адаптовані до різних потреб задачі виявлення об'єктів. Вони можуть обробляти зображення різних розмірів і показують хорошу ефективність в різних сценаріях. Завдяки їхній здатності до аналізу довгострокових залежностей, LambdaNetworks здатні знаходити складні об'єкти на

зображеннях з високою точністю.

4.3 LambdaNetworks для сегментації ліній різного розміру

Для розв'язання задачі сегментації ліній на зображенні ми використовуємо мережі LambdaNetworks. Ці мережі базуються на підході, в якому зображення розглядається як граф, де кожен піксель представляє вершину, а сусідні пікселі пов'язані ребром. За допомогою цього графу мережа може ефективно аналізувати зображення та визначати різні характеристики, такі як наявність та положення ліній.

Основною особливістю мереж LambdaNetworks є використання функції Lambda для зведення зображення до графу та зведення виходу мережі до вихідного зображення. Функція Lambda базується на геометричних трансформаціях, що дозволяє зберегти інформацію про геометричну структуру зображення та враховувати взаємодію між пікселями.

Для сегментації ліній на зображенні ми використовуємо мережу LambdaMask, яка базується на мережі LambdaResNet. Мережа LambdaResNet складається з залишкових блоків, що дозволяє ефективно вирішувати задачі комп'ютерного зору. Після того, як зображення було зведено до графу за допомогою функції Lambda, мережа виконує операції згортки та підсумовування на графі, що дозволяє враховувати взаємодію між пікселями та забезпечує ефективне вирішення задачі.

Для здійснення сегментації ліній на зображенні мережа LambdaMask використовує дві гілки. Перша гілка виконує сегментацію ліній вздовж вертикальної осі зображення, а друга –вздовж горизонтальної. Кожна гілка містить декілька шарів згортки та підсумовування на графі, а також механізми уваги та відновлення зображення з графу. Після того, як мережа виконала сегментацію ліній по обох гілках, вихід мережі з'єднується та піддається фінальній обробці, що дозволяє отримати сегментацію ліній

різного розміру та товщини на зображенні (рисунок 4.3).

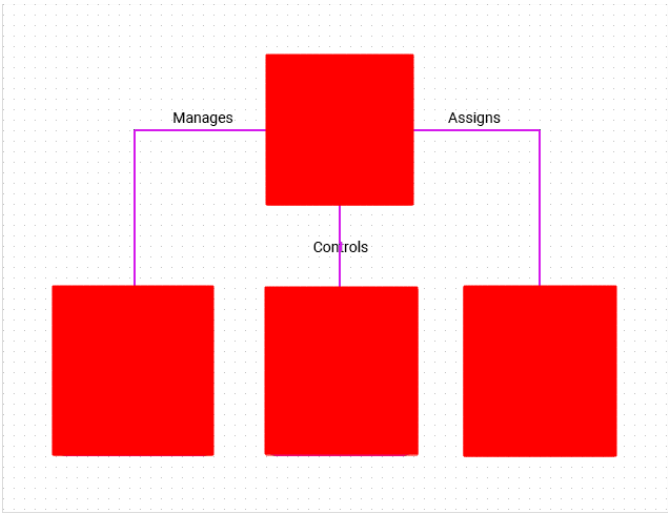

Рисунок 4.3 – Сегментація ліній

LambdaNetworks показує високу точність у сегментації ліній, навіть у випадках складних сценаріїв, таких як перекриття ліній, шум чи відсутність повного з'єднання між лініями. Це досягається завдяки глибокій архітектурі мережі та використанню глобальної інформації про зображення. Крім того, LambdaNetworks є гнучким інструментом, оскільки його можна використовувати для різних типів сегментації ліній на зображеннях. Незалежно від форми, кута або товщини ліній, LambdaNetworks може ефективно моделювати їх геометрію та визначати їх точну межу.

4.4 Пошук блоків на діаграммах за допомогою LambdaNetworks

Пошук блоків на діаграммах є однією з важливих задач у сфері комп'ютерного зору та обробки зображень. Використання методу LambdaNetworks для цієї задачі може мати деякі переваги і потенційні перспективи. LambdaNetworks є глибокою нейронною мережею, яка використовує графові згортки для здійснення аналізу зображень та виявлення важливих особливостей. Цей метод може бути застосований до

діаграм, де блоки представлені як вузли графа, а зв'язки між ними – як ребра. Пошук блоків на діаграммах може бути досягнутий шляхом використання LambdaNetworks для виявлення та класифікації цих блоків. Однією з переваг використання LambdaNetworks для пошуку блоків на діаграммах є їх здатність до врахування контекстуальної інформації та глобальних залежностей між блоками. Це дозволяє отримати більш точні та стабільні результати при виявленні та класифікації блоків на діаграммах. Крім того, застосування графових згорток дозволяє зменшити кількість параметрів моделі та поліпшити її швидкодію. Завдяки гнучкості та адаптивності LambdaNetworks можуть працювати з діаграмами різної складності та масштабу. Вони можуть ефективно виявляти блоки навіть у великих діаграмах з багатьма вузлами та зв'язками. Крім того, LambdaNetworks можуть бути навчені на великому обсязі даних, що дозволяє покращити їх універсальність та здатність до узагальнення. На рисунку 4.4 представлено результат роботи LambdaNetworks для пошуку блоків на блок схемі.

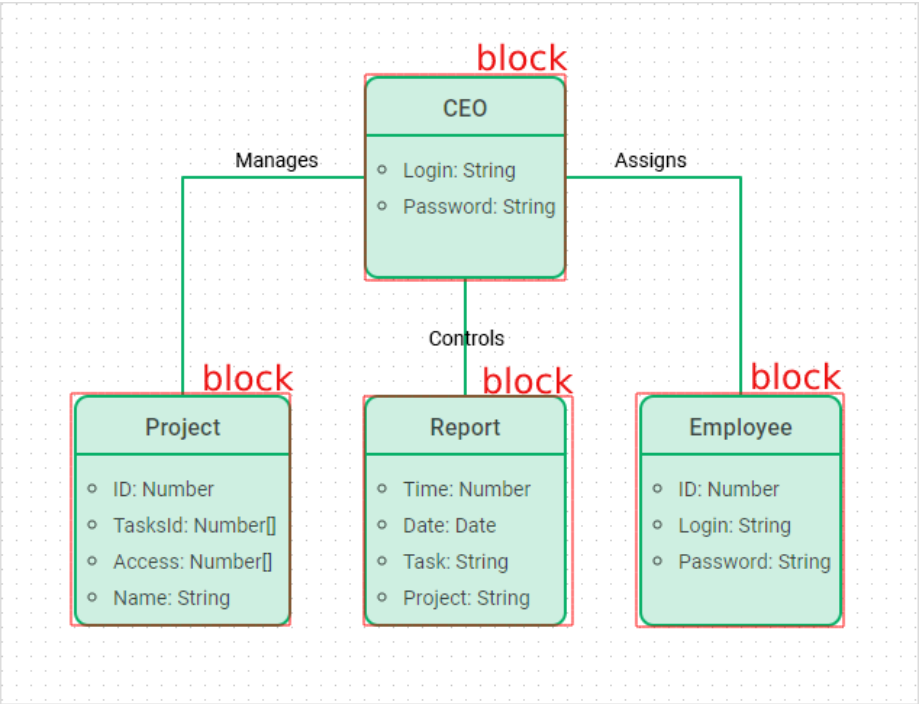

Рисунок 4.4 – Результат роботи LambdaNetworks для пошуку блоків на блок схемі

Слід зазначити, що використання LambdaNetworks для пошуку блоків на діаграммах також може мати свої обмеження. Наприклад, одним з недоліків LambdaNetworks для пошуку блоків на діаграммах є їх вимога до обсягу даних для навчання. Так як цей метод використовує глибокі нейронні мережі, які мають велику кількість параметрів, велика кількість даних може бути потрібна для досягнення задовільної точності. Це може бути обмеженням для задач, де обмежений обсяг даних доступний. Крім того, LambdaNetworks можуть мати складну архітектуру та вимагати високої обчислювальної потужності для навчання та інференсу. Використання графових згорток та інших операцій на графах може бути витратним з точки зору обчислень, особливо при обробці великих діаграм. Це може вплинути на продуктивність моделі та може вимагати потужнішої апаратної платформи для ефективного виконання.Також, LambdaNetworks можуть бути більш складними для реалізації та налаштування порівняно з іншими методами пошуку блоків на діаграммах. Вимагається розуміння графових згорток та вміння налаштувати гіперпараметри моделі для досягнення оптимальних результатів. Це може створювати додатковий труднощі для дослідників та практиків, які мають обмежений час або ресурси для вивчення та використання складних моделей.

Однак, незважаючи на ці недоліки, використання LambdaNetworks для пошуку блоків на діаграммах може бути перспективним напрямом досліджень. Залежно від конкретних вимог задачі та доступних ресурсів, LambdaNetworks можуть виявитися потужним інструментом для точного та ефективного виявлення та сегментації ліній різного розміру та товщини на зображеннях. Їх здатність до врахування контекстуальної інформації та глобальних залежностей між блоками може сприяти точному розпізнаванню та сегментації ліній навіть в складних ситуаціях. Крім того, їх гнучкість та адаптивність дозволяють ефективно працювати з різними розмірами та товщиною ліній, що робить їх придатними для різних типів зображень та додатків.

4.5 Представлення сегментованих ліній та знайдених блоків у вигляді графа

Представлення сегментованих ліній та знайдених блоків у вигляді графа є одним з ефективних способів візуалізації та аналізу результатів сегментації на зображеннях. Вхідними даними для нашого методу є сегментоване зображення блок-схеми. Спочатку ми знаходимо всі блоки та лінії на зображенні за допомогою методу сегментації, описаного в попередньому розділі. Кожен знайдений блок або лінія перетворюється в вершину графу з відповідним типом: блок або лінія. Вершини графу репрезентуються як точки в n-вимірному просторі, де n – кількість властивостей, які описують кожен вузол (рисунок 4.5).

Для кожної вершини визначається вектор ознак. Ознаки можуть включати координати пікселів, колір, розмір, форму та інші характеристики, які можуть бути важливими для подальшого аналізу. В даному випадку головною ознакою буде положення пікселя в просторі (на зображені), що в подальшому допоможе відтворити блок-схему або діаграму у вигляді графа. А також, додатково, будуть збережені зображення блоків у кожній відповідній вершині, для подальшого аналізу.

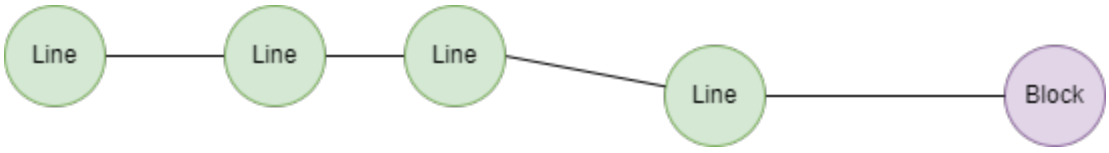

Рисунок 4.5 – Графове представлення елементу діаграми

Графове представлення надає структуровану інформацію про зв'язки між лініями та блоками, що дозволяє проводити подальші обчислення та аналіз [14]. У графовому представленні, кожен сегментований блок або лінія представляється як вузол графа. Зв'язки між вузлами відображаються у

вигляді ребер. Наприклад, якщо блок та лінія перетинаються або мають спільну точку, між відповідними вузлами буде наявне ребро. Таке графове представлення дозволяє відображати як прості, так і складні структури зв'язків між блоками та лініями. Однією з переваг графового представлення є його здатність до виявлення складних залежностей між сегментованими лініями та блоками. Завдяки графовим алгоритмам та аналізу структури графа, можна виявити важливі особливості, такі як групування ліній, ієрархічні зв'язки або симетрію. Це допомагає отримати більш повне розуміння структури зображення та виявлення важливих елементів. Крім того, графове представлення дає можливість застосовувати різні алгоритми та методи аналізу графів для виконання подальшої обробки та визначення характеристик блоків та зв'язків. Також графове представлення надає можливість передати багато інформації в компактному форматі. Воно дозволяє відображати зв'язки між блоками та лініями у вигляді ребер, що дозволяє швидко сприймати та аналізувати ці зв'язки. Важливо зазначити, що графове представлення зберігає контекстуальну інформацію про лінії та блоки. Розміщення вузлів графа та тип зв'язків між ними можуть передавати важливі характеристики, такі як просторові відношення або ієрархія. Графове представлення є універсальним інструментом, який можна використовувати в різних областях і додатках. Воно може застосовуватися у комп'ютерному зорі, медичному зображенні, графічному мистецтві та інших областях, що робить його універсальним засобом аналізу структури.

В процесі представлення сегментованих ліній та знайдених блоків у вигляді графа можна зіткнутися з певними труднощами, які варто враховувати. По-перше, обробка та візуалізація великих графів може вимагати значних обчислювальних ресурсів та часу. Це особливо актуально, якщо розмір зображення або кількість сегментованих об'єктів велика. По–друге, непередбачувана структура або зміна форми сегментованих об'єктів може ускладнити встановлення зв'язків між ними у вигляді графа. Наприклад, перекривання ліній можуть створювати проблеми при

визначенні їхніх зв'язків. По-третє, визначення точних зв'язків між сегментованими об'єктами може бути складним завданням. Це може вимагати використання геометричних властивостей, контекстуальних ознак або додаткових даних для ефективного встановлення зв'язків. Крім того, вибір оптимального способу представлення графа також є важливим аспектом. Він може впливати на зручність аналізу та ефективність подальших обчислень, і тому потребує уважного вибору. А також, аналіз та інтерпретація графових представлень можуть бути складними завданнями, які вимагають спеціалізованих методів та експертного знання. Результати аналізу графа повинні бути ретельно інтерпретовані [15] для отримання корисної інформації та дотримання вимог конкретного додатку.

Одним з важливих кроків при побудові графа, який відображає сегментовані лінії на зображенні, є вибір правильної товщини для ліній. У більшості випадків, товщину ліній необхідно враховувати, щоб правильно відобразити геометричну структуру об'єктів на зображенні. Однак, при побудові графа на основі сегментованих ліній, товщина ліній може бути сприйнята як константа, наприклад 1 піксель, для оптимізації процесу.

Оскільки на зображенні лінії можуть мати різні товщини, врахування товщини при побудові графа може призвести до збільшення кількості вершин і ребер у графі, що може погіршити його ефективність та збільшити час обробки. Таким чином, при побудові графа на основі сегментованих ліній, використання константної товщини для кожної лінії є одним з можливих підходів для оптимізації процесу.

4.6 Пошук зав’язків між блоками у графі за допомогою алгоритму пошуку в глибину

Пошук зв'язків між блоками у графі є важливим завданням, яке може бути вирішене за допомогою різних алгоритмів. Один з таких алгоритмів – це алгоритм пошуку в глибину. Алгоритм пошуку в глибину [16] (Depth–

First Search, DFS) є одним з основних алгоритмів для обходу графа. Він працює шляхом просування глибше в граф, відвідуючи всі можливі вершини до досягнення кінцевої точки або до повного просмотру графа (рисунок 4.6).

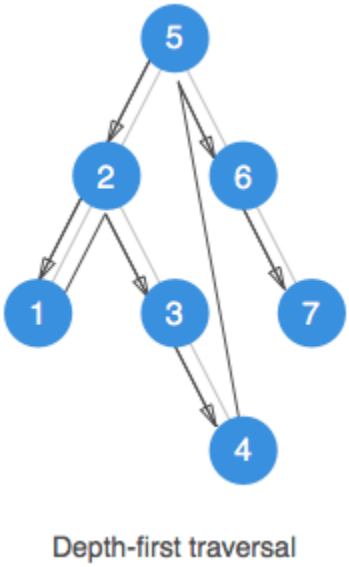

Рисунок 4.6 – Алгоритм пошуку в глибину

Для пошуку зв'язків між блоками у графі за допомогою алгоритму пошуку в глибину, спочатку обирається стартова вершина, яка відповідає початковому блоку. Потім алгоритм рекурсивно переходить до сусідніх вершин і відвідує їх, переходячи вглиб графа. Кожна вершина позначається як відвідана, коли до неї здійснюється перехід. Під час проходження алгоритму, можна збирати інформацію про зв'язки між блоками, наприклад, зберігати посилання на попередній блок або список блоків, з якими він зв'язаний.

Алгоритм пошуку в глибину має кілька переваг для пошуку зв'язків у графі. Він є простим у реалізації і доступним для широкого кола розробників. Алгоритм ефективний, оскільки розглядає всі можливі шляхи в глибину графа, виявляючи складні залежності та зв'язки між блоками. Крім того, він гнучкий і може бути налаштований для виявлення конкретних

типів зв'язків. Алгоритм гарантує відвідування всіх можливих вершин у графі, що дозволяє знайти всі можливі зв'язки між блоками. Крім того, він мало витратний щодо пам'яті, оскільки не вимагає зберігання історії відвіданих вершин. Враховуючи ці переваги, алгоритм пошуку в глибину є потужним і корисним інструментом для пошуку зв'язків у графі.

Алгоритм пошуку в глибину також має деякі обмеження. Одне з найважливіших обмежень полягає в тому, що він може застрягнути в циклі, якщо граф має циклічну структуру. Це може стати проблемою, особливо якщо граф має безліч вершин і ребер. Для запобігання цьому можна використовувати механізми, такі як обмеження глибини або позначення відвіданих вершин. Іншим обмеженням є те, що алгоритм пошуку в глибину може не знайти всі можливі шляхи або зв'язки у графі, особливо якщо вони знаходяться на значній відстані від початкової вершини. Це може бути проблемою для задач, де важливо виявити всі зв'язки між блоками. У таких випадках можуть бути використані інші алгоритми, які забезпечують більш широкий охоплення графу. Крім того, алгоритм пошуку в глибину може бути витратним з точки зору обчислювальних ресурсів і часу в обробці великих графів. Оскільки він розглядає всі можливі шляхи в глибину, його часова складність може залежати від розміру графа і кількості ребер. Великі графи можуть вимагати значних обчислювальних ресурсів для успішного виконання алгоритму.

Зазначені обмеження варто враховувати при використанні алгоритму пошуку в глибину для знаходження зв'язків між блоками у графі. Для вирішення цих обмежень можуть бути використані інші алгоритми або комбінація різних підходів для отримання бажаного результату

4.7 OCR тексту на блоках за допомогою TROCR

TROCR (Transformer-based Optical Character Recognition with Pre- trained Models) [17] – це технологія, яка використовується для

розпізнавання тексту на блоках або областях зображення. Вона використовує методи комп'ютерного зору та машинного навчання для автоматичного виділення текстових областей на зображенні та подальшого перетворення цього тексту в розпізнані символи. Однією з основних переваг TROCR є його здатність працювати з різноманітними типами тексту та шрифтами. Він може розпізнавати текст, написаний рукописом, друкувати, курсивом та іншими стилями. Це робить його корисним і універсальним інструментом для розпізнавання тексту у різних сценаріях, таких як сканування документів, обробка зображень або автоматичне витягування інформації з фотографій. Іншою перевагою TROCR є його висока точність розпізнавання тексту. Завдяки використанню сучасних алгоритмів машинного навчання, таких як моделі трансформери, TROCR може досягати вражаючої точності при розпізнаванні символів навіть на складних зображеннях або в умовах низької якості.

В нашому випадку аналізувалися зображення що зберігались у вершинах типу Блок. На лістингу 4.1 приведено код функції що виконує екстракцію тексту.

Лістинг 4.1 – Функція для екстракції тексту

```
    from transformers import TrOCRProcessor, VisionEncoderDecoderModel
    from PIL import Image
    import requests

    def ocr(image: str) -> str:

   processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-printed')

    model=VisionEncoderDecoderModel.from_pretrained('microsoft/trocr-base-printed')
```

Продовження лістингу 4.1

```
        pixel_values = processor(images=image,
    return_tensors="pt").pixel_values
    generated_ids = model.generate(pixel_values)
        generated_text  =  processor.batch_decode(generated_ids,
skip_special_tokens=True)[0]

     return generated_text
```

Крім того, TROCR є ефективним і швидким рішенням. Він може опрацьовувати великі обсяги даних швидко і ефективно, що робить його придатним для застосування в реальному часі або на великих наборах даних. Загалом, TROCR є потужним інструментом для розпізнавання тексту на блоках зображень. Він поєднує точність, універсальність та швидкість, що робить його корисним для широкого спектру застосувань, включаючи автоматичне витягування інформації, обробку документів, аналіз зображень та багато іншого.

Незважаючи на свої переваги, TROCR також має деякі обмеження, які варто враховувати. По-перше, точність розпізнавання тексту залежить від якості вхідного зображення. Якщо зображення має низьку роздільну здатність, шум або інші артефакти, це може призвести до помилок у розпізнаванні тексту. Отже, необхідно забезпечити якісне зображення перед процесом OCR. По-друге, TROCR може стикатися з обмеженнями щодо складності та розміру тексту. Великий обсяг тексту або складна структура можуть призвести до складнощів у правильному розпізнаванні. Деякі шрифти, особливо незвичні або маловживані, також можуть створювати проблеми для алгоритму OCR. Також важливо зазначити, що TROCR може мати обмежену підтримку для деяких мов або символів, особливо якщо вони не входять до стандартних наборів символів OCR. У таких випадках може бути потрібна додаткова настройка або розширення моделі для правильного розпізнавання. Загалом, хоча TROCR є потужним інструментом для OCR на

блоках зображень, він має свої обмеження. Розуміння цих обмежень і врахування їх у процесі роботи допоможе досягти кращих результатів у розпізнаванні тексту.

4.8 Аналіз проведених експериментів

Реалізувавши усі вищеописані експерименти у вигляді функцій на мові програмування Python можливо перейти до аналізу точності на більш складних та наближених до реального життя зображеннях. На рисунку 4.7 зображено діграму процесу обробки даних, у вигляді діаграми процесів.

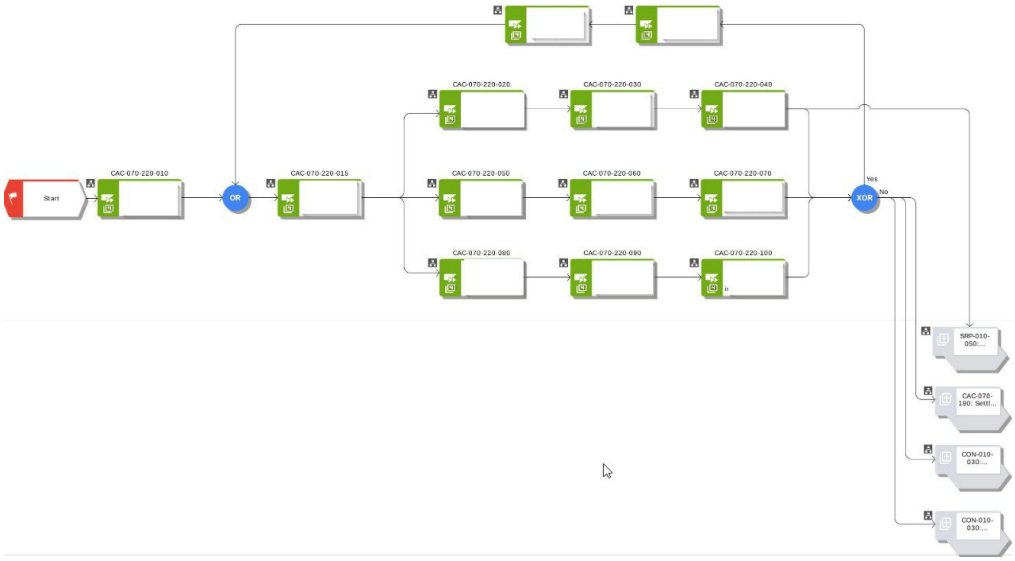

Рисунок 4.7 – Діаграма процесів обробки даних

Як видно, діаграма має багато елементів, розгалужену структуру ліній та декілька перетинів ліній. Застосовуючи Алгоритм описаний на початку розділу, вдалося отримати досить гарні результати, зважаючи на складність зображення. На рисунку 4.8 можна побачити візуалізацію знаходження блоків та сигментації ліній. Як видно з зображення LambdaNetworks неймовірно добре справляються з поставленою задачею. В додаток до цього було проведенно експеримент з використанням альтернативних алгоритмів,

в тому числі реалізованих з допомогою OpenCV. Жоден з них не мав такого самого результату, в основному через складність сигментації ліній.

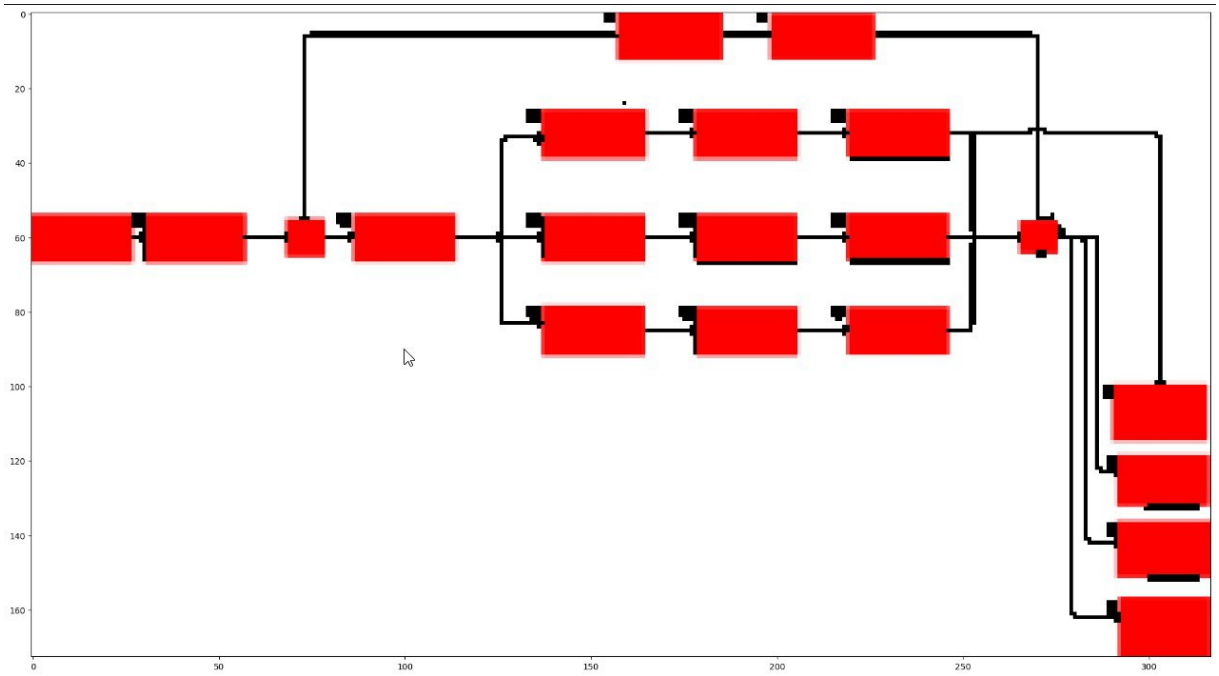

Рисунок 4.8 – Візуалізація роботи детекторів

Також можна побачити, що іноді LambdaNetworks утворюють артефакти сегментації (рисунок 4.9). В основному це спостерігається в місцях спеціальних позначок.

Рисунок 4.9 – Артефакти сегментації

Але ця проблема легко вирішується перетворенням ліній в представлення з товщиною в 1 піксель, як згадувалося раніше, що також допомагає оптимізувати побудову графу. На рисунку 4.10 можна побачити результат

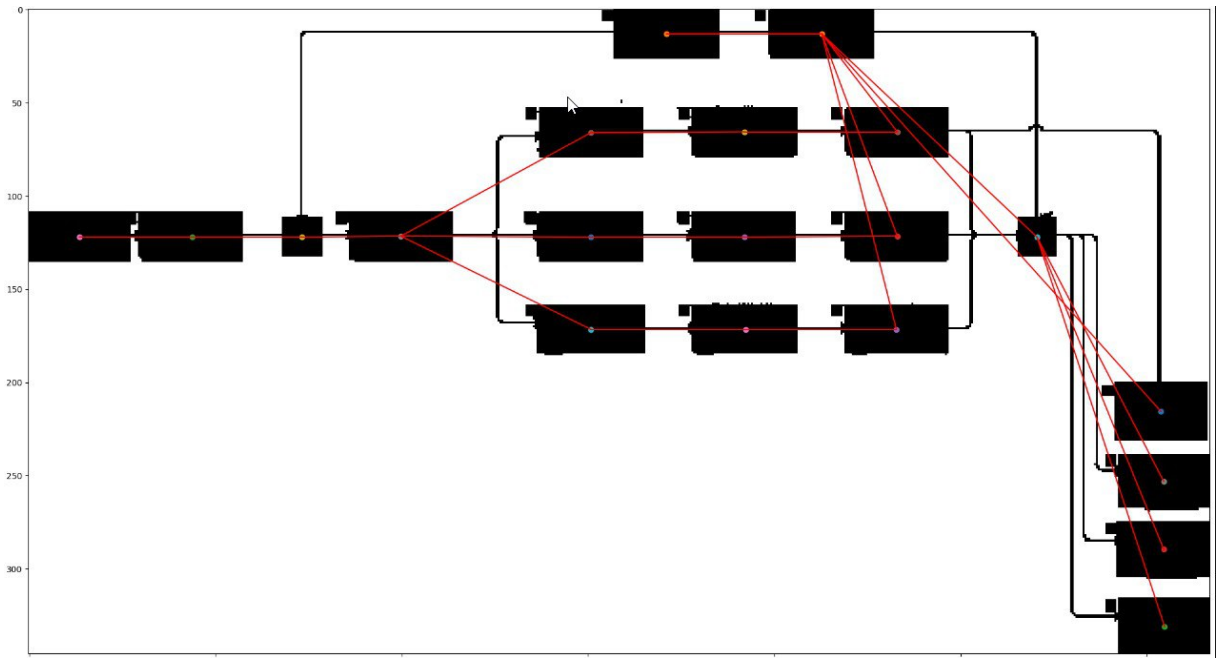

Рисунок 4.10 – Результат визначення зв’яків

проведення пошуку в ширину на графовому представленні діаграми для виявлення зв’язків між блоками. На цьому етапі алгоритм має більше недоліків у вигляді неточного визначення зв’язків, що може погіршити якість подальших кроків. Загалом, стає зрозуміло, що використання пошуку в глибину не є оптимальним рішенням для визначення зв’язків.

Можна зробити висновок, що даний алгоритм є досить перспективним в плані продукціоналізації та практичного використання. Більшість аналогічних сервісів мають безліч обмежень, починаючи від необхідності створення діаграм у спеціальному програмному забезпечені, закінчуючи ефективністю та точністю. Натомість, запропонований в атестаційній роботі алгоритм може бути вдосконалений на кожному кроці і гнучко адаптований під різні обставини.

5 ДОДАТКОВИЙ ФУНКЦІОНАЛ

Автоматична генерація програмного коду є важливим етапом в розробці програмного забезпечення, оскільки це дозволяє значно скоротити час, необхідний для написання коду та знизити кількість помилок, що можуть бути допущені людиною під час ручного введення коду. В останні роки було розроблено кілька інструментів для автоматичної генерації програмного коду. Одним з найбільш відомих інструментів є OpenAI Codex [18] (рисунок 5.1), який був представлений влітку 2021 року. Codex є великомасштабним нейромережевим модулем, побудованим на базі моделі GPT (Generative Pre-trained Transformer). Він навчений на великій кількості програмного коду та може генерувати функції, класи, методи та багато іншого з невеликими вказівками від користувача.

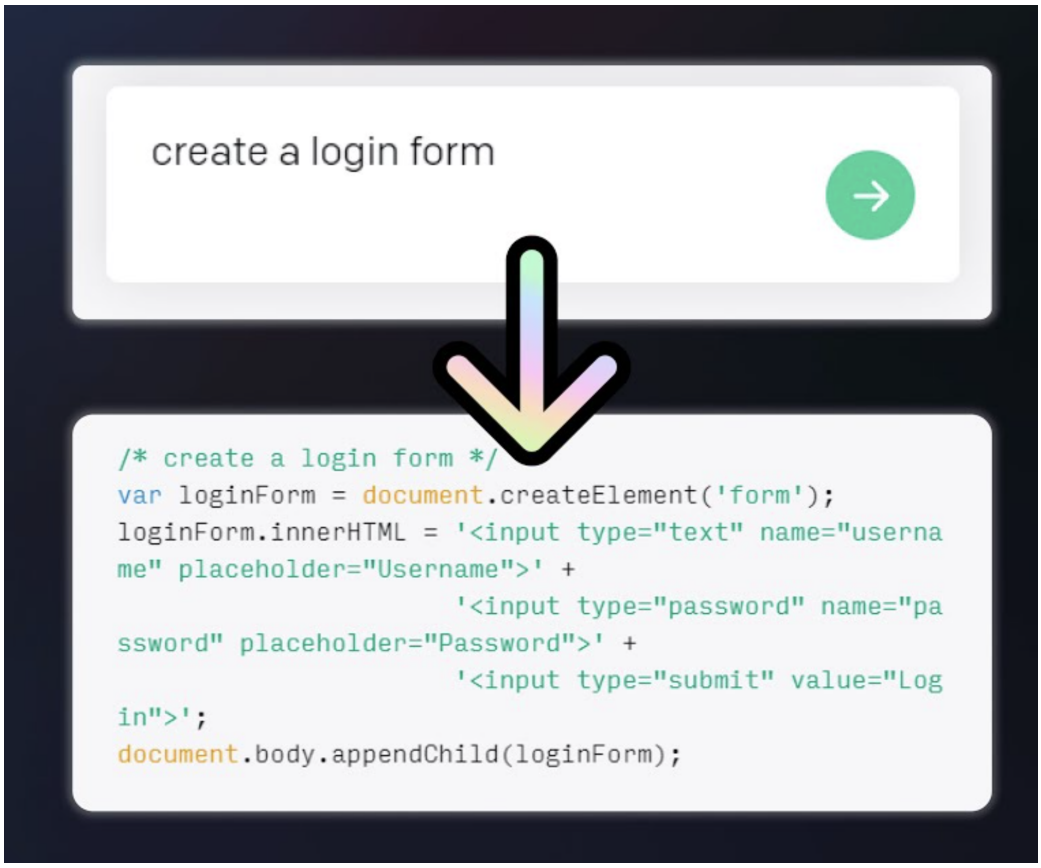

Рисунок 5.1 – Генерація коду за допомогою OpenAI Codex

5.1 ChatGPT

Одним з найбільш відомих і успішних моделей глибокого навчання для генерації тексту є ChatGPT [19], яка базується на архітектурі Transformer і була розроблена компанією OpenAI.ChatGPT є потужним інструментом для генерації тексту, який може використовуватися для створення відповідей на запитання, генерації тексту на основі заданого контексту, побудови діалогів та багатьох інших завдань. Також він може бути використаний для автоматичної генерації програмного коду, який зазвичай вимагає багато часу та зусиль від розробників. Використання ChatGPT для генерації програмного коду стало особливо популярним в останні роки. Наприклад, компанія OpenAI створила GPT-3, яка є найбільш потужною версією ChatGPT, здатною генерувати програмний код на основі прикладів та описувати його функціональність. Це відкрило шлях до більш швидкого та ефективного розроблення програмного коду. Застосування ChatGPT у дипломній роботі полягало в автоматичній генерації коду для обробки та аналізу діаграм. Для цього використовувалась платформа GitHub Copilot, яка базується на GPT-3. Ця платформа вміє генерувати код на основі запитів, що вводяться користувачем. Крім того, ChatGPT дозволяє автоматично генерувати код з невеликої кількості текстових описів задачі. Це особливо корисно для розробників, які не мають достатньої кількості досвіду для ручного написання складних програм. Для створення коду з текстового опису можна використовувати модифіковані версії GPT, що були навчені на великих корпусах програмного коду, таких як GitHub, StackOverflow та інші.

В рамках дипломної роботи було проведено дослідження використання ChatGPT для автоматичної генерації коду на прикладі задачі розпізнавання тексту на зображенні. Було використано модель GPT-3, що була навчена на корпусі програмного коду з використанням OpenAI API. Результати експерименту показали, що модель може генерувати коректний

код з високою точністю, особливо для простих задач.

Однак, використання ChatGPT для генерації коду має свої обмеження. Завдання, що потребують глибокого розуміння контексту та складного логічного мислення, можуть бути вирішені не задовільно. Також, згенерований код може бути не ефективним з точки зору швидкодії та ефективності використання ресурсів.

5.2 GitHub Copilot

GitHub Copilot – це інтелектуальний помічник для розробників, розроблений спільною ініціативою GitHub і OpenAI. Використовуючи потужні моделі машинного навчання, GitHub Copilot надає автоматичне завершення коду, пропозиції функцій та навіть цілі шаблони коду на основі контексту та структури проекту.

Основна перевага GitHub Copilot полягає в тому, що він може значно прискорити процес написання коду, забезпечуючи швидке і точне автодоповнення. Він може аналізувати існуючий код, коментарі, документацію та приклади з GitHub для надання рекомендацій згідно з конкретним контекстом. Це допомагає розробникам зосередитися на більш складних аспектах програмування, зменшуючи необхідність вручну вводити багато коду. Крім того, GitHub Copilot підтримує багато мов програмування та різноманітні фреймворки, що робить його універсальним інструментом для розробників з різними стеками технологій. Він також навчається від користувачів, тому з кожним використанням стає все кращим у розумінні контексту та наданні більш точних рекомендацій. GitHub Copilot також інтегрований безпосередньо з інтерфейсом редактора коду, що дозволяє розробникам зручно використовувати його без необхідності виходити зі свого редактора або IDE. Загалом, GitHub Copilot може значно покращити продуктивність розробників, забезпечуючи швидку і точну автоматичну генерацію коду, яка відповідає потребам проекту. Він

забезпечує зручний та ефективний спосіб розробки програмного забезпечення, зменшуючи час та зусилля для написання коду

Однак, GitHub Copilot має кілька обмежень, які варто враховувати при його використанні. Перш за все, його ефективність залежить від наявності відповідного контексту в існуючому коді. Якщо проект має обмежену кількість зразків коду або недостатню документацію, Copilot може надавати менш точні рекомендації. Крім того, хоча Copilot намагається врахувати стиль програмування проекту, він не завжди може повністю відповідати унікальним вимогам і стандартам команди. Також варто зазначити, що Copilot може генерувати схожі шаблони коду, що може призводити до повторення коду. Крім того, його рекомендації можуть бути обмеженими в менш відомих або специфічних мовах програмування або фреймворках. Необхідність підключення до Інтернету також є важливим обмеженням, оскільки GitHub Copilot вимагає хмарних сервісів для своєї роботи.

5.3 OpenAI GPT

OpenAI GPT (Generative Pre–trained Transformer) є моделлю глибокого навчання, розробленою компанією OpenAI. Вона базується на трансформерній архітектурі, яка дозволяє ефективно обробляти послідовні дані, такі як тексти. GPT використовує методи машинного навчання для автоматичного генерації тексту, перекладу мови, відповіді на питання та багатьох інших прикладних завдань, що вимагають розуміння та генерації природної мови. Однією з ключових особливостей GPT є його здатність до передбачення послідовностей слів чи токенів з великим контекстом. Вона навчається на великому обсязі текстових даних, що дозволяє моделі отримувати глибоке розуміння мови та контексту. GPT здатний генерувати продовження тексту, враховуючи контекст, що сприяє його використанню в різних застосуваннях, включаючи автоматичне завершення речень, розробку діалогових систем та покращення письмових навичок. Ще однією

з переваг GPT є його можливість до підтримки різних мов та гнучкість у використанні. Він може бути застосований для обробки текстів різної природи, від коротких фраз до довгих документів. Більш того, він може бути доопрацьований та адаптований до конкретних завдань та вимог проекту.

Незважаючи на ці переваги, GPT також має свої обмеження. По–перше, використання GPT вимагає потужних обчислювальних ресурсів, таких як GPU або TPU. Це може бути недоцільним для користувачів з обмеженими обчислювальними можливостями або фінансовими обмеженнями. Крім того, GPT навчається на великому обсязі текстових даних, тому якість його виконання залежить від навчального набору. Якщо дані недостатньо різноманітні або неправильно підібрані, це може призвести до проблем з узагальненням та якістю згенерованого тексту. Також, треба врахувати, що GPT не завжди адекватно розуміє довготривалі залежності між різними частинами тексту, що може призводити до неочікуваних результатів або недостатньо точних відповідей. Крім того, існує ризик збурення, оскільки GPT навчається на доступних Інтернет–даних. Це може призводити до включення неприпустимого або образливого контенту, що може створювати проблеми з використанням GPT в етичних аспектах. Важливо зазначити, що відповідальне використання GPT та обізнаність з його обмеженнями є необхідними для забезпечення точності та етичності його застосування.

5.4 Висновки щодо генерації коду

Аналізуючи використання GPT, GitHub Copilot та ChatGPT для генерації коду з графового представлення діаграми, можна сказати, що кожен з цих інструментів має свої переваги та недоліки.

GPT – це одна з перших моделей машинного навчання, яка використовувала архітектуру Transformer, яка є досить гнучкою для використання в генерації коду. За допомогою GPT, можна генерувати код за

попереднім контекстом, що дозволяє отримати досить точний та зрозумілий код для певної задачі. Проте, GPT не здатна генерувати великі програмні проекти та довгі функції, оскільки модель може втратити зв'язок між початковою та кінцевою частиною коду.

GitHub Copilot – це інтегрований інструмент, який розроблено на основі GPT. Вона має доступ до великої кількості відкритого коду та дозволяє використовувати його для генерації коду з контексту. За допомогою GitHub Copilot, можна генерувати код для різних мов програмування та бібліотек. Проте, вона може мати проблеми з генерацією точного та зрозумілого коду в окремих випадках.

ChatGPT – це мовний асистент, який розроблено на основі технології GPT. Вона дозволяє використовувати природну мову для генерації коду та пояснення різних аспектів програмування. За допомогою ChatGPT, можна генерувати код з високим рівнем абстракції, що дозволяє зменшити кількість рутинних завдань під час програмування. Проте, ChatGPT може мати проблеми з точністю та зрозумілістю згенерованого коду у складних завданнях. З іншого боку, ChatGPT використовується для генерації коду відповідно до заданої мети та контексту. Це може бути корисним в розрізі генерації абстракного коду з діаграм.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи були проведені експерименти по застосуванню новітніх архітектур мереж згорткових нейронних мереж для розв'язання задач сегментації діаграм та пошуку блоків діаграм. Зокрема, була досліджена архітектура LambdaNetworks, що базується на концепції трансформера і включає в себе механізм обробки зв'язків між різними елементами діаграми. Першим етапом було застосування LambdaNetworks для сегментації ліній на діаграмах. Після навчання моделі на наборі даних, який містив різні типи діаграм та їх ліній, було встановлено, що вона добре справляється з задачею сегментації ліній та забезпечує високу точність визначення кордонів ліній. Другим етапом було застосування тієї ж архітектури для пошуку блоків діаграм. Для досягнення цієї мети було згенеровано набір даних, який містив різні типи діаграм та їх блоків, і була навчена модель на цьому наборі даних. Результати дослідження показали, що модель, навчена на наборі даних блоків діаграм, має достатню точність для знаходження блоків в нових діаграмах. Отримані дані з сегментації ліній та знаходження блоків діаграм були використані для побудови графового представлення діаграми, що відображає взаємозв'язки між різними елементами. Це графове представлення може бути використане для подальшого аналізу та зрозуміння структури діаграми. Однак, на цьому етапі роботи є потреба в додатковій оптимізації та випробування альтернативних підходів.

Також, в процесі роботи було використано претреновані моделі та інструменти для автоматизації генерації коду, такі як GitHub Copilot та ChatGPT. Використання таких інструментів дозволило скоротити час на написання. Для подальшого вдосконалення системи необхідно провести додаткові дослідження та розробити нові методи та інструменти для побудови графів та аналізу діаграм. Є необхідність у врахуванні деяких обмежень. Наприклад, робота з діаграмами різних масштабів та складностей

може потребувати додаткових алгоритмічних підходів або адаптації моделей до конкретних умов. Крім того, збільшення розміру даних може вимагати більш потужних обчислювальних ресурсів і тривалого часу для навчання моделей. Додатковою складністю є також вирішення проблеми обробки шуму або недосконалостей у даних діаграм. Також важливо збалансувати точність та швидкість обробки даних. Застосування глибоких нейронних мереж може забезпечити високу точність, але вимагатиме більшого обчислювального часу, тоді як менш складні моделі можуть працювати швидше, але матимуть обмежену точність. Додатковою проблемою може стати розмітка даних для навчання моделей. Збір і підготовка достатньо великого та репрезентативного набору даних, особливо для складних діаграм, може бути часо- та ресурсозатратним завданням. Необхідно також враховувати етичні аспекти використання цих технологій. З метою забезпечення конфіденційності та захисту особистих даних, слід використовувати відповідні методи шифрування та захисту інформації.

У цілому, результати дослідження є задовільними та вказують на високий потенціал застосування технологій машинного навчання та обробки зображень для аналізу діаграм та автоматизації процесів розробки програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mishsra A. *et al.* From Cognitive Psychology to Image Segmentation: A Change of Perspective. URL: https://www.researchgate.net/publication/325806783_From_Cognitive_Psychology_to_Image_Segmentation_A_Change_of_Perspective (date of access: 23.03.2023)

2. Anjin Park *et al*. Graph Cuts–Based Automatic Color Image Segmentation IEEE Access (2008)

3. Jianbo Shi Normalized Cuts and Image Segmentation IEEE Access (2000)

4. UML @ Classroom / M. Seidl et al. Cham : Springer International Publishing, 2015.

5. Long J. Fully Convolutional Networks for Semantic Segmentation. URL: https://arxiv.org/abs/1411.4038 (date of access: 23.03.2023)

6. Ronneberger O. U–Net: Convolutional Networks for Biomedical Image Segmentation. URL: https://arxiv.org/abs/1505.04597 (date of access)

7. Image Segmentation with Felzenszwalb's algorithm. URL: https://www.analyticsvidhya.com/blog/2021/05/image–segmentation–with–felzenszwalbs–algorithm/ (date of access: 23.03.2023).

8. He K. *et al.* Mask R–CNN. URL: https://arxiv.org/abs/1703.06870 (дата звернення: 23.03.2023)

9. Bai M. Deep Watershed Transform for Instance Segmentation. URL: https://arxiv.org/abs/1611.08303 (date of access: 23.03.2023).

10. Hugging Face – Blog. Hugging Face – The AI community building the future. URL: https://huggingface.co/blog (date of access: 08.05.2023).

11. Introduction – NetworkX 3.1 documentation. NetworkX. URL: https://networkx.org/documentation/stable/reference/introduction.html (date of access: 08.05.2023).

12. Gould R. Graph theory. Menlo Park, Calif : Benjamin/Cummings Pub. Co., 1988. 332 p.

13. LambdaNetworks: Modeling Long-Range Interactions Without Attention. arXiv.org. URL: https://arxiv.org/abs/2102.08602 (date of access: 14.04.2023).

14. Bondy J. A., Murty U. S. R. Graph Theory. London : Springer London, 2008. URL: https://doi.org/10.1007/978-1-84628-970-5 (date of access: 01.02.2023).

15. Erciyes K. Algebraic Graph Algorithms. Cham : Springer International Publishing, 2021. URL: https://doi.org/10.1007/978-3-030-87886-3 (date of access: 04.11.2022).

16. Even S., Even G. Graph Algorithms. Cambridge University Press, 2011.

17. TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models. arXiv.org. URL: https://arxiv.org/abs/2109.10282 (date of access: 10.02.2023).

18. OpenAI Codex. OpenAI. URL: https://openai.com/blog/openai-codex (date of access: 01.05.2023).

19. Introducing ChatGPT. OpenAI. URL: https://openai.com/blog/chatgpt (date of access: 10.03.2023).

ДОДАТОК А

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	_68_ с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата		
Розробив		Кошурников Г.О.			Шифр групи	Код напр./спец.
Перевірив		Єрохін А.Л.			СШІм-21-21	122
Н.контр.		Малєєва І.А.			ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.				

Дослідження методів аналізу графів для вирішення задач комп’ютерного зору