МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет

імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри,
д-р техн. наук, проф.

_____ І. М. Журавська

« __ » _____ 2023 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Побудова нейронної мережі для детектування малопомітних рухомих об'єктів

Спеціальність 123 Комп'ютерна інженерія

123 – КБР.ПЗ.00 – 405.21910502

Студент _____ Д. Є. Басов
підпис
«__» _____ 202__ р.

Керівник канд. фіз.-мат. наук, доцент _____ С. В. Пузирьов
підпис
«__» _____ 202__ р.

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет імені Петра Могили
Факультет комп'ютерних наук
Кафедра комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Зав. кафедри _____ І. М. Журавська

«____»_____20__ р.

ЗАВДАННЯ
на виконання кваліфікаційної бакалаврської роботи

Видано студенту групи 405 факультету комп'ютерних наук

Басову Даніілу Євгенійовичу
(прізвище, ім'я, по-батькові студента)

1. Тема кваліфікаційної роботи
Побудова нейронної мережі для детектування малопомітних рухомих об'єктів

Затверджена наказом по ЧНУ від «___»_____20___ р. №_____

2. Строк представлення кваліфікаційної роботи «____»_____20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні
Очікуваний результат: нейронна мережа для детектування малопомітних рухомих об'єктів, реалізована програмними засобами мови програмування Python
Початкові дані: кадри відеофрагментів з датасету MoCa-Mask

4. Перелік питань, що підлягають розробці
– вивчити найсучасні наукові дослідження на тему детектування малопомітних рухомих об'єктів;
– проаналізувати існуючі рішення для побудови нейронної мережі для детектування малопомітних рухомих об'єктів;
– визначити математичний апарат, що буде використаний при розробці нейронної мережі;
– розробити архітектуру нейронної мережі;

– визначити апаратне забезпечення для реалізації АПЗ, що використовує нейронну мережу;
– реалізувати нейронну мережу для детектування малопомітних рухомих об'єктів методами мови програмування Python та бібліотеки PyTorch;
– виконати експериментальне дослідження побудованої нейронної мережі на тестовому датасеті;
– порівняти ефективності роботи побудованої нейронної мережі з аналогічними моделями нейронних мереж.

5. Перелік графічних матеріалів
Слайди презентації

6. Завдання до спеціальної частини
– дослідити нормативно-правову базу в галузі охорони праці під час роботи з комп'ютерними пристроями;
– визначити вимоги щодо безпеки та захисту здоров'я працівників під час роботи з комп'ютерними приладами.

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
А. О. Алексєєва, к.т.н., доцент	кафедра екології Медичного інституту ЧНУ імені Петра Могили	Спеціальна частина з охорони праці

Керівник роботи канд. фіз.-мат. наук, доцент Пузирьов С.В.
(посада, прізвище, ім'я, по батькові)

_____
(підпис)

Завдання прийнято до виконання

Басов Д. Є.
(прізвище, ім'я, по батькові студента)

Басов
(підпис)

Дата видачі завдання «___»_____ 20___ р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної бакалаврської роботи

Тема: Побудова нейронної мережі для детектування малопомітних рухомих об’єктів.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КБР	20.12.2022	26.12.2022	Виконано
2	Огляд літератури за темою роботи	14.01.2023	01.02.2023	Виконано
3	Складання календарного плану КБР	04.02.2023	06.02.2023	Виконано
4	Аналіз предметної області	07.02.2023	02.03.2023	Виконано
5	Розробка проєктних рішень	03.03.2023	18.03.2023	Виконано
6	Моделювання апаратної частини	18.03.2023	06.04.2023	Виконано
7	Розробка програмного забезпечення та його тестування	07.04.2023	08.05.2023	Виконано
8	Написання розділу з охорони праці	09.05.2023	19.05.2023	Виконано
8	Відгук керівника КБР	20.05.2023	21.05.2023	Виконано
9	Оформлення КБР та презентації	22.05.2023	30.05.2023	Виконано
10	Перший передзахист КБР	30.05.2023	30.05.2023	Виконано
11	Другий передзахист КБР	13.06.2023	13.06.2023	Виконано
12	Рецензування	13.06.2023	20.06.2023	Виконано
14	Завершення оформлення КБР та презентації	13.06.2023	20.06.2023	Виконано
13	Захист кваліфікаційної бакалаврської роботи	27.06.2023	28.06.2023	Виконано

Розробив здобувач ВО Басов Д. Є. Басов

(прізвище, ім’я, по батькові) *(підпис)*

«____» _____ 2023 р.

Керівник роботи канд. фіз.-мат. наук, доцент Пузирьов С.В. _____

(посада, прізвище, ім’я, по батькові) *(підпис)*

«____» _____ 2023 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи
«Побудова нейронної мережі для детектування малопомітних рухомих об’єктів»
Студент 405 гр. Басов Даніїл Євгенійович
Керівник: канд. фіз.-мат. наук, доцент Пузирьов С.В.

Актуальність теми кваліфікаційної бакалаврської роботи. В останнє десятиріччя науково-технічна сфера переживає підйом у напрямку розробки штучного інтелекту. З’являються нові, більш досконалі архітектури нейронних мереж, які дозволяють реалізовувати виконання все більш інтелектуальних задач. «Розумне» програмне забезпечення на основі штучних нейронних мереж знаходить своє застосування у нових сферах.

Однією з таких сфер є детектування малопомітних рухомих об’єктів. І хоча детектування об’єктів взагалі є одним з вже класичних та найпоширеніших застосувань нейронних мереж, але серйозне та систематичне дослідження детектування саме малопомітних об’єктів почалося відносно недавно. Звідси, актуальним є узагальнення зроблених напрацювань у цьому напрямку, аналіз найкращих рішень, подальше дослідження і удосконалення технологій побудови нейронних мереж для детектування малопомітних рухомих об’єктів.

Об’єкт дослідження: технології комп’ютерного зору.

Предмет дослідження: нейронна мережа для детектування малопомітних рухомих об’єктів.

Мета бакалаврської кваліфікаційної роботи: реалізація нейронної мережі для детектування малопомітних рухомих об’єктів шляхом поєднання таких архітектур нейронних мереж, як згорткова нейронна мережа та трансформер.

Бакалаврська кваліфікаційна робота була апробована на науковій конференції «Ольвійський форум – 2023: стратегії країн Причорноморського регіону в геополітичному просторі».

Кваліфікаційна робота містить: перелік скорочень, вступ, три розділи, висновок, перелік джерел посилання та два додатки.

Вступ містить основні обґрунтування актуальності розробки обраної теми, об’єкт, предмет дослідження, мету та завдання які необхідно виконати для досягнення поставленої мети.

У першому розділі проаналізовано об’єкт і предмет дослідження, та виконано аналітичний огляд наукової літератури на тему детектування малопомітних об’єктів.

У другому розділі описано архітектуру нейронної мережі для детектування малопомітних рухомих об’єктів, математичний апарат, що використовуються при її розробці, та обране апаратне забезпечення.

У третьому розділі проаналізовано існуючі технології для побудови нейронних мереж для детектування об’єктів, описано результат програмної реалізації нейронної мережі для детектування малопомітних рухомих об’єктів, виконано оцінку ефективності її роботи у порівнянні з аналогами.

У висновку описано результати виконання кваліфікаційної роботи.

Додатки містять довідку про перевірку на унікальність пояснювальної записки кваліфікаційної бакалаврської роботи та код програмного забезпечення.

В спеціальній частині з охорони праці розглянуто забезпечення вимог охорони праці під час роботи з комп’ютерними пристроями.

Бакалаврська кваліфікаційна робота містить 81 сторінку, 31 рисунок, 29 джерел посилання, 2 додатки.

Ключові слова: *комп’ютерний зір, детектування малопомітних рухомих об’єктів, детектування об’єктів, нейронні мережі*.

ABSTRACT

of the Bachelor’s Thesis
"Building neural network for detecting camouflaged moving objects"
Student: Basov Daniil Yevheniyovych
Supervisor: Candidate of physico-mathematical sciences, Docent Puzyrov S. V.

Relevance of the topic of the bachelor's thesis. In the last decade, the scientific and technological field has been experiencing an upturn in the development of artificial intelligence. New, more sophisticated neural network architectures are emerging that allow for the implementation of increasingly intelligent tasks. "Smart" software based on artificial neural networks is being used in new areas.

One of these areas is the detection of moving camouflaged objects. Although object detection in general is one of the most classic and widespread applications of neural networks, a serious and systematic study of detecting camouflaged objects has begun relatively recently. Hence, it is important to summarize the developments in this area, analyze the best solutions, further research and improve the technologies for building neural networks for detecting camouflaged moving objects.

Object of research: computer vision technologies.

Subject of research: neural network for detecting camouflaged moving objects.

Objective of the bachelor's thesis: implementation of a neural network for detecting camouflaged moving objects by combining such neural network architectures as convolutional neural network and transformer.

The bachelor's thesis was approbated at the "Olbia Forum 2023: Strategies of the Black Sea Region Countries in the Geopolitical Space" scientific conference.

The bachelor's thesis contains a list of abbreviations, an introduction, three chapters, a conclusion, a list of references and two appendices.

The introduction contains the main justifications for the relevance of the chosen topic, the object, subject of research, purpose and tasks to be performed to achieve the goal.

The first section analyzes the object and subject of the study and provides an analytical review of the scientific literature on the topic of detecting camouflaged objects.

The second section describes the architecture of the neural network for detecting camouflaged moving objects, the mathematical framework used in its development, and the selected hardware.

The third section analyzes existing technologies for building neural networks for object detection, describes the result of the software implementation of a neural network for detecting camouflaged moving objects, and evaluates the effectiveness of its work in comparison with analogues.

The conclusion describes the results of the bachelor's thesis.

The appendices contain a certificate of uniqueness verification of the explanatory note of the bachelor's thesis and the software code.

The special part on occupational safety and health considers the requirements of occupational safety and health when working with computer devices.

The bachelor's thesis contains 81 pages, 31 figures, 29 references, 2 appendices.

Key words: *computer vision, camouflaged moving object detection, object detection, neural networks*.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

АПЗ – апаратно-програмне забезпечення

CNN – convolutional neural network
ResNet – residual neural network
COD – camouflaged/concealed object detection
GRA – group-reversal attention
NCD – neighbor connection decoder
TEM – texture enhanced module
CAD – Camouflaged Animal dataset
CAMO – Camouflaged Object dataset
MoCA – Moving Camouflaged Animals video dataset
CPU – central processing unit
CUDA – Compute Unified Device Architecture
IDE – integrated development environment

ВСТУП

Актуальність теми кваліфікаційної роботи. В останнє десятиріччя науково-технічна сфера переживає підйом у напрямку розробки штучного інтелекту. З’являються нові, більш досконалі архітектури нейронних мереж, які дозволяють реалізовувати виконання все більш інтелектуальних задач. «Розумне» програмне забезпечення на основі штучних нейронних мереж знаходить своє застосування у нових сферах.

Однією з таких сфер є детектування малопомітних рухомих об’єктів. І хоча детектування об’єктів взагалі є одним з вже класичних та найпоширеніших застосувань нейронних мереж, але серйозне та систематичне дослідження детектування саме малопомітних об’єктів почалося відносно недавно. Звідси, актуальним є узагальнення зроблених напрацювань у цьому напрямку, аналіз найкращих рішень та подальше дослідження і удосконалення технологій побудови нейронних мереж для детектування малопомітних рухомих об’єктів.

Об’єктом дослідження є технології комп’ютерного зору.

Предметом дослідження є нейронна мережа для детектування малопомітних рухомих об’єктів

Метою цієї кваліфікаційної роботи є реалізація нейронної мережі для детектування малопомітних рухомих об’єктів шляхом поєднання таких архітектур нейронних мереж, як згорткова нейронна мережа та трансформер.

Відповідно до поставленої мети визначено такі завдання:

– виконання аналізу найсучасніших наукових досліджень на тему детектування малопомітних рухомих об’єктів;

– аналіз існуючих рішень для побудови нейронної мережі для детектування малопомітних рухомих об’єктів;

– визначення вимог до нейронної мережі, що розробляється;

– визначення математичного апарату, що буде використаний при розробці нейронної мережі;

– розробка архітектури нейронної мережі;

– вибір апаратного забезпечення для реалізації АПЗ, що використовує нейронну мережу;

– реалізація нейронної мережі для детектування малопомітних рухомих об’єктів методами мови програмування Python та бібліотеки PyTorch;

– експериментальне дослідження побудованої нейронної мережі на тестовому датасеті;

– порівняння ефективності роботи побудованої нейронної мережі з аналогічними моделями нейронних мереж;

– дослідження нормативно-правової бази в галузі охорони праці під час роботи з комп’ютерними пристроями;

– визначення вимог щодо безпеки та захисту здоров’я працівників під час роботи з комп’ютерними приладами.

Дослідження і розробка нейронної мережі для детектування малопомітних рухомих об’єктів має велике практичне значення. Побудована нейронна мережа може бути застосована у багатьох сферах: для пошуково-рятувальних задач, для систем розумного відеоспостереження, для знаходження та захисту рідкісних видів тварин, для візуальних задач дослідження океану та космосу, для систем медичного візуального аналізу та ін.

Дана кваліфікаційна робота була апробована на науковій конференції «Ольвійський форум – 2023: стратегії країн Причорноморського регіону в геополітичному просторі».

1 АНАЛІТИЧНИЙ ОГЛЯД ДЖЕРЕЛ

1.1 Розкриття об’єкту та предмету дослідження

Проблема побудови нейронної мережі для детектування малопомітних рухомих об’єктів розглядається в рамках галузі комп'ютерних наук – теорії та технологій комп’ютерного зору (англ. computer vision), та належить до кола задач детектування об’єктів (англ. object detection).

Комп'ютерний зір – це галузь досліджень і технологій, яка фокусується на тому, щоб дозволити комп'ютерам отримати високорівневе розуміння візуальних даних, таких як зображення і відео. Сфера комп’ютерного зору включає розробку алгоритмів, моделей і систем, які дозволяють комп'ютерам інтерпретувати, аналізувати і витягувати значущу інформацію з візуальних даних, подібно до того, як люди сприймають і розуміють візуальний світ.

Метою комп'ютерного зору є відтворення та автоматизація можливостей людського зору, що дозволяє машинам сприймати, інтерпретувати та приймати рішення на основі візуальних даних. Ця сфера включає в себе різні завдання, в тому числі розпізнавання зображень і відео, виявлення і відстеження об'єктів, сегментацію зображень, розуміння сцени, генерацію зображень і візуальне міркування.

Детектування об'єктів – це завдання комп'ютерного зору, яке передбачає ідентифікацію та локалізацію об'єктів, що представляють інтерес, на зображенні або відео. Метою детектування об'єктів є не тільки класифікація об'єктів за попередньо визначеними категоріями, а й точне визначення їхнього просторового розташування у візуальному вхідному сигналі.

Алгоритми детектування об'єктів зазвичай складаються з двох основних компонентів: механізму пропозиції регіонів і компонента класифікації об'єктів. Механізм пропозиції регіонів генерує набір регіонів-кандидатів, які, ймовірно, містять об'єкти. Компонент класифікації об'єктів відповідає за присвоєння міток класів запропонованим регіонам і уточнення координат обмежувальних рамок.

Детектування малопомітних рухомих об’єктів поєднує задачі детектування замаскованих об’єктів (англ. camouflaged object detection, COD) [1] або детектування прихованих об’єктів (англ. concealed object detection, COD) [5], сегментацією руху на відео (англ. video motion segmentation, VMS), сегментації об’єкту на відео (англ. video object segmentation, VOS) та детектування рухомих об’єктів (англ. moving object detection, MOD).

Особливість та складність даної задачі полягає у автоматизації процесу ідентифікації та локалізації таких об’єктів, які візуально «зливаються» з фоном навколишнього середовища за рахунок кольору, текстури чи низького контрасту.

Якщо в якості формату матеріалу з замаскованим об’єктом, який потрібно детектувати, використовується відео, то цю задачу іноді називають детектуванням замаскованих об’єктів на відео (англ. video camouflaged object detection, VCOD) [9].

Технології детектування об’єктів базуються або на опорно-векторних машинах (англ. support vector machine, SVM), або на штучних нейронних мережах.

1.2 Штучні нейронні мережі

Штучна нейронна мережа – це обчислювальна система, яка складається з поєднаних між собою одиниць – нейронів, та здатна навчатися, покращуючи свій результат.

Нейрони в штучних нейронних мережах організовані у декілька шарів: шару входу (англ. input layer), прихованих шарів (англ. hidden layers) та шару виходу (англ. output layer). Кількість нейронів в шарах входу та виходу, а також кількість прихованих шарів та нейронів у них залежить від конкретної задачі, яку вирішує нейронна мережа.

Зв’язок нейрона одного шару з нейроном наступного шару має певне числове значення та називається вагою (англ. weight). Вага вказує наскільки сильно зміна значення даного нейрона впливає на зміну значення нейрона наступного шару.

Також кожен нейрон, крім нейронів шару входу, має певний регульований числовий параметр, що називається зміщення (англ. bias). Зміщення дозволяє

нейронній мережі виявляти та моделювати нелінійні зв'язки між входами та виходами. Використання зміщення допомагає змістити вихід функції активації, що додає нейронній мережі гнучкості та дозволяє їй навчатися і відображати складні закономірності в даних.

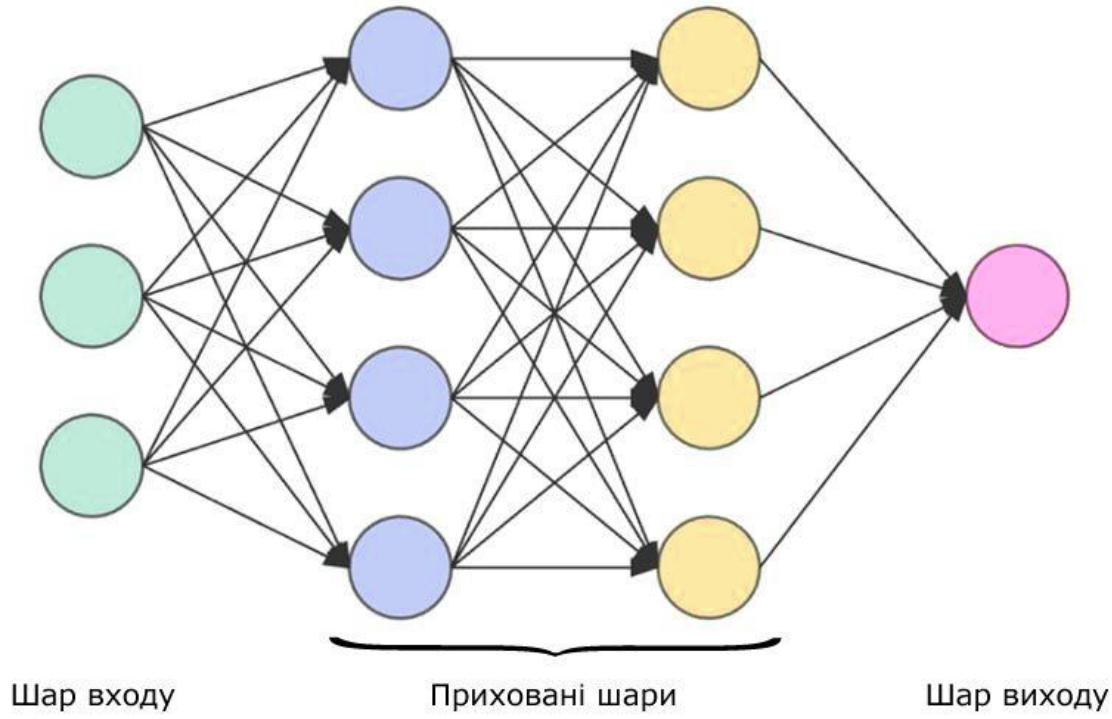


Рисунок 1.1 – Загальний приклад нейронної мережі

Всі нейрони, крім нейронів шару входу, мають функцію активації нейрона (англ. activation function), яка відповідає за введення нелінійності в обчислення. Вона бере зважену суму входів плюс зміщення, застосовує нелінійне перетворення і формує кінцевий вихід нейрона. Функція активації дозволяє нейрону моделювати складні взаємозв'язки між входами і виходами, дозволяючи мережі навчатися і представляти нелінійні патерни в даних.

Нейронні мережі навчаються, ітеративно змінюючи свої внутрішні параметри, відомі як ваги та зміщення, на основі спостережуваних вхідних даних і бажаного результату. Цей процес називається навчанням, або тренуванням нейронної мережі.

Спочатку випадковим чином ініціалізуються ваги та зміщення нейронної мережі. Далі виконується крок прямого поширення (англ. forward propagation). Це

передбачає проходження вхідних даних через шари мережі, застосування зважених обчислень і функцій активації для генерації вихідних даних.

Згенеровані вихідні дані порівнюються з бажаними або цільовими значеннями за допомогою функції витрат (англ. cost function), яка кількісно визначає різницю між прогнозованим і бажаним результатом. Мета навчання нейронної мережі полягає в тому, щоб мінімізувати ці втрати.

Наступним кроком застосовується зворотне поширення (англ. backpropagation), яке є ключовим механізмом для оновлення ваг і зсувів нейронної мережі. Це відбувається за допомогою обчислення градієнта функції витрат, який кількісно вказує величину коригування параметрів мережі. Розмір кроку корегування визначається темпом навчання (англ. learning rate).

Далі, вище наведені кроки повторюються для пакету (англ. batch) або підмножини навчальних даних Цей процес називається ітерацією або епохою. Кілька ітерацій виконуються для покращення продуктивності мережі. Нейронна мережа продовжує ітерації через навчальні дані, оновлюючи свої параметри і зменшуючи втрати. З часом мережа вчиться робити кращі прогнози та мінімізувати різницю між прогнозованими та бажаними результатами

Нейронні мережі здатні обробляти великі обсяги даних і складні моделі. Зі збільшенням розміру та складності проблеми нейронні мережі можна масштабувати, додаючи більше шарів, нейронів або параметрів. Це дозволяє їм ефективно вловлювати і представляти складність, притаманну проблемі.

Нейронні мережі можна навчати поступово, що дозволяє їм вчитися на нових даних, зберігаючи при цьому знання, отримані при виконанні попередніх завдань. Ця здатність має важливе значення для додатків, які включають дані, що змінюються, або вимагають безперервного навчання.

Загалом, нейронні мережі особливо ефективні для вирішення складних завдань завдяки своїй здатності навчатися і виявляти складні закономірності та внутрішні взаємозв'язки в даних. Вони можуть навчатися на навчальному наборі даних і робити точні прогнози на нових, небачених прикладах. Це має вирішальне

значення для вирішення складних проблем, оскільки гарантує, що модель зможе вловити основні закономірності та взаємозв'язки в даних, а не запам'ятовувати конкретні приклади.

1.2.1 Архітектури нейронних мереж

Існує багато різних архітектур нейронних мереж: перцептрон, рекурентна нейронна мережа (англ. recurrent neural networks, RNN), довга короткочасна пам'ять (англ. long short-term memory, LSTM), згорткова нейронна мережа (англ. convolutional neural network, CNN), залишкова нейронна мережа (англ. residual neural network, ResNet), трансформер та ін.

Для задач детектування малопомітних об’єктів найбільш часто застосовуються згорткові нейронні мережі, залишкові нейронні мережі, трансформери та їх гібриди.

Згорткова нейронна мережа – це спеціалізована архітектура нейронної мережі, що призначена для обробки та аналізу структурованих сіткоподібних даних, зокрема зображень і відео. Розроблена у 80-х рр. XX ст., згорткова нейронна мережа зробила революцію в сфері комп'ютерного зору, продемонструвавши дуже високу продуктивність у таких завданнях, як класифікація зображень, детектування об'єктів і сегментація зображень.

Ключовою особливістю згорткових нейронних мереж є використання згорткових шарів, які використовують фільтри або ядра (англ. kernel) для виконання локальних операцій над вхідними даними. Згорткові шари застосовують до вхідних даних набір фільтрів, що навчаються, переміщуючись по просторових вимірам даних та обчислюючи скалярний добуток між вагами фільтрів і вхідними значеннями. Ця операція дозволяє мережі автоматично навчатися і виявляти локальні закономірності та особливості у вхідних даних.

Згорткові нейронні мережі зазвичай складаються з декількох згорткових шарів, за якими йдуть нелінійні функції активації (наприклад, ReLU) та об'єднувальні шари, які зменшують просторові розміри для зменшення

обчислювальної складності та виокремлення домінантних ознак. Вихідні дані цих шарів потім згладжуються і з'єднуються з повністю з'єднаними шарами, які виконують високорівневе вилучення ознак і створюють остаточний результат класифікації або регресії.

Однією з ключових переваг згорткових нейронних мереж є їхня здатність фіксувати просторові ієрархії ознак. Початкові шари мережі вивчають низькорівневі ознаки, такі як краї, кути і текстури, тоді як глибші шари вивчають високорівневі ознаки, які представляють більш складні патерни і семантичну інформацію. Таке ієрархічне представлення дозволяє згортковій нейронній мережі досягати дуже високої продуктивності в задачах, що вимагають просторового розуміння і виявлення інваріантних ознак.

Залишкова нейронна мережа – це архітектура нейронної мережі, яка була презентована у 2015 р. Вона створювалась для вирішення проблеми навчання дуже глибоких нейронних мереж, а саме – для пом’якшення проблеми деградації, коли точність мережі починає знижуватись або навіть деградувати в міру того, як мережа стає глибшою.

Залишкова нейронна мережа складається із залишкових блоків. Кожен залишковий блок складається з декількох згорткових шарів, за якими слідують пропускні з'єднання. Пропускні з'єднання дозволяють безпосередньо передавати вхідні дані до глибших шарів мережі, створюючи короткі шляхи, які оминають згорткові шари.

Використовуючи дану архітектуру, залишкова нейронна мережа дозволяє навчати дуже глибокі нейронні мережі з сотнями шарів. Така глибина дозволяє мережі вловлювати і представляти більш складні ознаки, що призводить до покращення продуктивності в різних завданнях, таких як класифікація зображень, виявлення об'єктів і семантична сегментація.

У 2017 р. була винайдена нова архітектура нейронних мереж – трансформер. Початково її застосовували в межах сфери обробки натуральної мови (англ. Natural language processing, NLP) для таких задач як: машинний переклад, генерація текстів

та питально-відповідальні системи (англ. question answering, QA). Проте, після того як ця архітектура показала свою ефективність, її почали використовувати в інших галузях, включаючи й сферу комп’ютерного зору.

В основі архітектури трансформера лежать шари самоуваги (англ. self-attention layers). Самоувага дозволяє нейронній мережі зважувати важливість різних позицій в межах вхідної послідовності при здійсненні прогнозів в певній позиції. Трансформер обчислює ваги уваги, порівнюючи кожну позицію з кожною іншою позицією у вхідній послідовності. Таким чином нейронна мережа здатна фіксувати залежності і відношення між в послідовності, незалежно від їх відносної відстані.

Архітектура трансформера також включає нейронні мережі прямого поширення (англ. feedforward neural network), залишкові з'єднання та нормалізацію шару. Ці компоненти підвищують здатність нейронної мережі помічати складні патерни та підвищувати ефективність навчання.

1.3 Огляд існуючих рішень

1.3.1 Детектування малопомітних об'єктів

Детектування об’єктів є класичною задачею сфери комп’ютерного зору, для вирішення якої застосовуються штучні нейронні мережі. Проте, незважаючи на це, помітні наукові дослідження в галузі детектування малопомітних об’єктів почалися лише досить недавно.

У роботі [1] було виконано ґрунтовний огляд проблеми сегментації замаскованого об’єкта, визначено основні особливості замаскованого об’єкта як такого та процесу його детектування. Особливу увагу було звернено на наступне. Малопомітні (замасковані, приховані) об’єкти легко ігноруються людським оком. Проте, як тільки людина дізнається, що такий об’єкт існує на зображенні чи відео, вона починає ретельно роздивлятися все зображення чи відео, щоб його ідентифікувати. Ця аналогія надихнула архітектуру нейронної мережі ANet. Вона

складається з двох потоків: класифікаційного, який передає обізнаність о наявності замаскованого об’єкта, та сегментаційного. Перший заснований на згортковій нейронній мережі, а другий – на повно-згорткової мережі (англ. fully convolutional network, FCN).

У роботі [2] було виконано комплексне дослідження цієї нової теми – детектування замаскованих об’єктів. У даній роботі було дано загальне формулювання задачі детектування замаскованих об’єктів як привласнення кожному пікселю i впевненості (англ. confidence) $p_i \in [0,1]$, де p_i позначає ймовірнісне значення пікселя i – значення 0 дається пікселям, які не належать до замаскованих об’єктів, у той час коли значення 1 вказує на повну приналежність пікселя до замаскованого об’єкта. Також у [2] було запропоновано оптимальний фреймворк нейронної мережі для детектування замаскованих об’єктів SINet, чия архітектура натхнена першими двома стадіями полювання. Він складається з двох модулів: модулю пошуку (англ. search module, SM) та модулю ідентифікації (англ. identification module, IM). Модуль пошуку є гібридом залишкової нейронної мережі (англ. residual neural network, ResNet) та елементів згорткової нейронної мережі, що називаються рецепторними полями (англ. the receptive field, RF). Його задача – знайти характерні ознаки (англ. features), які можуть свідчити про наявність об’єкта. Модуль ідентифікації будує карту маскування (англ. camouflage map) об'єкта, використовуючи модуль уваги пошуку (англ. search attention, SA), рецепторні поля та компоненти часткового декодеру (англ. partial decoder component, PDC). Для тренування нейронної мережі в якості функції витрат використовується перехресна ентропія. В [2] було виконано порівняння моделі нейронної мережі SINet з аналогами на різних датасетах, яке зокрема показало її більшу продуктивність, у порівнянні з ANet.

Схожий з [2] підхід продемонструвала робота [3], де було презентовано модель нейронної мережі PFNet. Її головними будівельними блоками є модуль позиціювання (англ. positioning module, PM) та модуль фокусування (англ. focus module, FM). Архітектура нейронної мережі також є поєднанням архітектур ResNet

та CNN. Для доведення ефективності введених модулів було проведено багато експериментів. Їх можна об'єднати у дві групи: перша – порівняння результатів роботи PFNet з її аналогами на різноманітних датасетах, друга – абляційні експерименти. Суть останніх полягає у видаленні певних елементів (в даному випадку – модулів) нейронної мережі, та подальше тестування продуктивності цієї модифікованої нейронної мережі проти звичайної – це дозволяє визначити значення цих видалених елементів у роботі всієї системи. Згідно з проведеними експериментами PFNet є більш продуктивною за свої аналоги, у тому числі за SINet.

Примітною є робота [4], де в якості архітектури нейронної мережі для детектування було запропоновано використовувати графову згорткову мережу (англ. Graph Convolutional Networks, GCN). Використання методів, заснованих на графах, дозволяє включити деталі країв детектуємого об’єкта у потік сегментації. Для цього використовується три основних компонента: мультизадачне виявлення ознак (англ. Multi-Task Feature Extraction, MTFE), модуль регіонально-індукованого обґрунтування графів (англ. Region-Induced Graph Reasoning, RIGR), та модуль обмеженого краями обґрунтування графів (англ. Edge-Constricted Graph Reasoning, ECGR).

Подальший розвиток напрацювань, зроблених у [2], містить робота [5], яка присвячена детектуванню прихованих об’єктів. В неї було покращено продуктивність моделі нейронної мережі SINet за допомогою використання нових компонентів: декодеру сусідського з’єднання (англ. neighbor connection decoder, NCD), модуля покращення текстури (англ. Texture Enhanced Module, TEM) та уваги до групової зміни (англ. group-reversal attention, GRA). [5] має обширний експериментальний розділ з демонструванням роботи оновленої SINet порівняно з аналогами на різних датасетах, у тому числі на датасетах з суб- та суперкласами замаскованих об’єктів (видів та груп тварин з природнім маскуванням). Також у роботі представлені спеціальні абляційні експерименти для доведення ефективності введення запропонованих компонентів до нейронної мережі.

У Таблиці 1.1 здійснено порівняння описаних вище моделей архітектур нейронних мереж на тестовому датасеті CAMO-Test за такими показниками, як: структурний вимір S_α (чим вище, тим краще), адаптивний E-вимір $E_\emptyset$ (чим вище, тим краще), взважений F-вимір F_β^w (чим вище, тим краще), середня абсолютна похибка M (чим менше, тим краще). Найкращі результати виділено жирним шрифтом. Посилання вказують на те, звідки були взяті дані результатів тестування

Таблиця 1.1 – Порівняння моделей архітектур нейронних мереж для детектування малопомітних об’єктів на тестовому датасеті CAMO-Test.

Назва моделі	$S_\alpha \uparrow$	$E_\emptyset \uparrow$	$F_\beta^w \uparrow$	$M \downarrow$
ANet [5]	0.682	0.685	0.484	0.126
SINet [5]	0.751	0.771	0.606	0.100
PFNet [3]	0.789	0.852	0.695	0.085
S-MGL [4]	0.772	0.850	0.664	0.089
R-MGL [4]	0.775	0.847	0.673	0.088
SINetv2 [5]	**0.820**	**0.882**	**0.743**	**0.070**

1.3.2 Детектування рухомих об'єктів

За знаходження рухомих об’єктів відповідають задачі, що називають сегментацією руху на відео (англ. video motion segmentation, VMS), сегментації об’єкту на відео (англ. video object segmentation, VOS) та детектування рухомих об’єктів (англ. moving object detection, MOD).

Традиційними підходами до вирішення цих задач є: вилучення меж руху в полі потоку (англ. flow field) і покращення початкової оцінки за допомогою зовнішніх ознак, або поєднання візуальних та рухових сигналів за допомогою гібдридної архітектури .

Більш сучасний напрям – це явне використання оптичного потоку (англ. optical flow) як вхідого значення для тренування згорткової нейронної мережі, та генерування міток руху на попіксельному рівні.

1.3.3 Детектування малопомітних об'єктів на відео

На відміну від звичайних, візуальні особливості малопомітних об'єктів дають менше інформації, ніж сигнали руху об’єкта, тому вони вважаються менш ефективними для детектування.

У [6; 7] сегментація малопомітного об’єкта на відео виконується за допомогою розрахунку оптичного потоку та поля кута (англ. angle field). У моделі архітектури нейронної мережі, що була запропонована у [6], виконується робота безпосередньо з оптичним потоком пари послідовних кадрів та використовується інформація з поля кута для отримання імовірнісної моделі руху об'єкта. У [7] представлений інший підхід. Через складність сегментації рухомого об’єкта безпосередньо за допомогою оптичного потоку, запропонована модель нейронної мережі тренується виконувати сегментацію об’єкта та середовища, базуючись на використанні поля кута, замість оптичного потоку.

У [8] було представлено модель архітектури нейронної мережі для реєстрації відео та сегментації руху. Вона складається з двох різних модулі: реєстрації та сегментації руху. Модуль реєстрації є диференційованим, чия задача – виділити межи об’єкта. Модуль сегментації руху має пам’ять, його задача – виявити рухомі області. Наведена модель використовує явний метод вирівнювання оптичним потоком, який будує просторову відповідність між сусідніми кадрами. Недоліком цього метода є те, що вихід нейронної мережі може бути недостатньо точною в динамічних сценах з швидким рухом об’єкта.

Фреймворк, який об’єднує оцінку руху та сегментацію об’єкта, був представлений у [9]. Його архітектура складається з двох модулів: модуля короткострокової динаміки та довгострокового модуля уточнення. Перший модуль фіксує рух об’єкта у послідовних кадрах імпліцитно, на відміну від експліцитного підходу, що використовує оптичний потік. Довгостроковий модуль уточнення приймає результати першого модуля та зменшує накопичені неточності. З точки

зору архітектури нейронних мереж, дана модель є гібридом трансформеру та згорткової нейронної мережі.

1.3.4 Датасети

Датасетами називають набір даних для тренування чи перевірки роботи нейронної мережі. Тренувальні та тестові датасети мають анотовані об’єкти, на яких і вчяться нейронні мережі. Анотація зазвичай має вигляд прямокутної рамки, що позначає межі об’єкту. Іноді використовуються маски – повний контур об’єкта. У випадку, якщо задачею є класифікація об’єктів, то анотація буде також містити назву об’єкта.

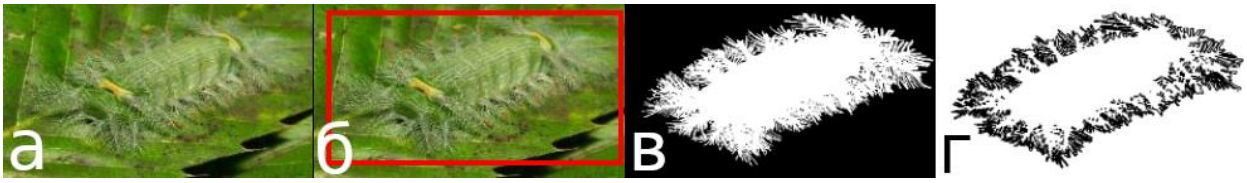


Рисунок 1.2 – Приклад анотацій зображення з датасета COD10K: б) прямокутної рамки; в) маски; г) країв об’єкта.

Саме відсутність стандартизованих та об’ємних датасетів було причиною дуже малої кількості досліджень на тему детектування малопомітних об’єктів. До певного часу існувало лише декілька датасетів, що відповідали цій задачі. Але їх істотним недоліком був малий обсяг даних. Датасет, запропонований у [6], містив не зображення, а відео, яких було лише 9. Датасет CHAMELEON, що був представлений у неопублікованій роботі [10], мав лише 76 зображень тварин з природним камуфляжем.

Ситуація кардинально змінилась після публікації роботи [1], в якій було вперше презентовано датасет, зроблений спеціально для задач детектування замаскованих об'єктів, що має назву Camouflaged Object dataset (CAMO). Даний датасет налічує 1250 високоякісних зображень замаскованих об’єктів (тварин та людей). З них 1000 призначені для тренувальних цілей, а 250 – для тестування.

Також у [1] був представлений інший датасет CAMO-COCO, що складається з 1250 додаткових зображень з датасету MS-COCO, які містять незамасковані об’єкти.

Наступною значною віхою стала публікація [2], в якій було запропоновано датасет COD10K, який складається з 10000 ретельно підібраних зображень: 5066 с замаскованими об'єктами, 3000 фонових та 1934 без замаскованих об'єктів. Також зображення поділяються на 10 супер-класів (літаючі, морські, наземні, амфібії, інші, небо, рослинність, приміщення, океан та пісок) та 78 під-класів (69 замаскованих та 9 незамаскованих). Пропонується використовувати для тренування 6000 випадково обраних зображень, а для тестування – залишкові 4000. На даний момент датасет COD10K є найбільшим датасетом зображень з замаскованими об’єктами.

Першим датасетом з відео, що містить замасковані об’єкти, був датасет Camouflaged Animal Dataset (CAD), запропонований у вже згаданій публікації [6]. Це невеликий датасет, що складається з 9 відеофрагментів, зібраних з платформи YouTube. Відеофрагменти мають вручну виставлені маски об’єктів на кожному 5-ому кадрі.

Рисунок 1.3 – Приклад різних зображень з датасета COD10K

У роботі [8] був представлений датасет відео Moving Camouflaged Animals (MoCA), що має 37 тисяч кадрів з 141 відеофрагментів, більшість яких мають роздільну здатність 1280x720 та частоту кадрів 24 кадра/с. Головним недоліком датасету MoCA є те, що анотації до об’єктів представлені у вигляді прямокутних рамок, що ускладнює оцінку ефективності сегментації замаскованого об’єкта на відео.

Вище вказаний недолік був виправлений із публікацією роботи [9], де датасет MoCA був реорганізований у датасет MoCA-Mask, що містить 87 відеофрагментів з 22939 кадрами в цілому з попіксельними масками. Надано анотації, рамки об’єктів та щільні сегментаційні маски для кожного 5-ого кадру для кожного відео в датасеті. Також у [9] було надано перший грунтовний бенчмарк для існуючих методів детектування замаскованих об’єктів на відео. Весь датасет поділений на дві частини: для тренування – 71 відеофрагментів з 19313 кадрами, для тестування – 16 відеофрагментів з 3626 кадрами.

Рисунок 1.4 – Приклад кадру відеофрагмента з датасету MoCA-Mask (а), його бінарної маски (б) та ілюстрації їх накладання (в)

1.4 Вибір та обґрунтування підходів

Спираючись на досвід досліджень у сфері детектування малопомітних рухомих об’єктів, було прийнято рішення також застосовувати двохмодульний підхід при розробці архітектури нашої нейронної мережі. Перший модуль нейронної мережі бере два послідовні кадри та знаходить орієнтовну маску об’єкта. Другий модуль її уточнює та знаходить довгострокові відповідності, отримуючи таким чином остаточний результат.

Архітектура нашої нейронної мережі є гібридом згорткової нейронної мережі та трансформера. Застосування гібридної архітектури дає нам поєднання сильних сторін обох архітектур. Елементи згорткової мережі вилучають характерні ознаки об’єкта з окремих кадрів, в той час як трансформатор фіксує довгострокові залежності та враховує важливі просторові положення. Поєднуючи ці компоненти, мережа може ефективно фіксувати як локальну, так і глобальну інформацію для детектування малопомітних рухомих об'єктів.

Для тренування та перевірки результатів роботи нашої нейронної мережі для детектування малопомітних рухомих об’єктів був обраний датасет відео MoCA-Mask. Саме він є найбільш досконалим датасетом малопомітних об’єктів на відео, який має і достатню кількість матеріалу, і високу інформативність, забезпечену використанням анотацій об’єктів у вигляді масок.

Висновок до розділу 1

Побудова нейронної мережі для детектування малопомітних рухомих об’єктів розглядається в рамках галузі комп’ютерного зору та є задачею детектування об’єктів. Для її вирішення найчастіше використовуються штучні нейронні мережі – обчислювальні системи, що складаються з поєднаних між собою штучних нейронів, та здатні навчатися, покращуючи свій результат. Серед архітектур нейронних мереж для задачі детектування об’єктів зазвичай використовують архітектуру згорткової нейронної мережі, а в останні роки – трансформера, чи їх гібридного поєднання.

Детектування малопомітних об’єктів є відносно молодим напрямком досліджень. Задача детектування малопомітних об’єктів полягає у ідентифікації і локалізації таких об’єктів, зовнішній вигляд яких зливається з фоном зображення чи відео. Для вирішення цієї задачі були запропоновані такі моделі архітектур нейронних мереж: ANet [1], SINet [2], PFNet [3], S-MGL [4] та R-MGL [4], SINetv2 [5].

Детектування рухомих малопомітних об’єктів є ще більш молодим напрямком досліджень у сфері комп’ютерного бачення. Він поєднує напрацювання, зроблені для вирішення задач детектування малопомітних об’єктів на зображеннях [1–5] з методами детектування рухомих об’єктів [6–9].

Для навчання нейронних мереж використовуються спеціальні набори анотованих даних – датасети. Саме відсутність стандартизованих та об’ємних датасетів було причиною дуже малої кількості досліджень на тему детектування малопомітних об’єктів. Зараз існує достатня кількість датасетів для цієї задачі. Датасети CHAMELEON [10], CAMO [1] та CAMO-COCO [1], COD10K [2] використовуються для детектування малопомітних об’єктів на зображеннях, а CAD [6], MoCA [8] та MoCA-Mask [9] – на відео.

В результаті огляду наукової літератури та аналізу останніх напрацювань в цій галузі було зроблено рішення використовувати в архітектурі два окремих модуля: один з яких шукає об’єкт, другий – остаточно його ідентифікує. Нейронна мережа буде гібридом згорткової нейронної мережі та трансформера. Для навчання мережі буде використовуватися датасет MoCA-Mask [9].

2 ПРОЄКТУВАННЯ НЕЙРОННОЇ МЕРЕЖІ ТА МАТЕМАТИЧНИЙ АПАРАТ

2.1 Архітектура нейронної мережі для детектування малопомітних рухомих об’єктів

2.1.1 Вхід та вихід нейронної мережі

Нейронна мережа для детектування малопомітних рухомих об’єктів в якості входу приймає відеофрагмент, на якому знаходиться малопомітний рухомий об’єкт. Результатом роботи мережі (виходом) є набір прогнозованих бінарних масок даного детектованого об’єкта для кожного кадру у відеопослідовності на попіксельному рівні.

Позначимо відеофрагмент з Т кадрами в часі *t*, як:

$$\{I^t\}_{t=1}^{T}, I^t \in \mathbb{R}^{3XHxW}, \quad (2.1)$$

де H – висота кадру;

W – ширина кадру.

Позначимо прогнозовану маску для кадра відеофрагмента I^t в часі *t*, як:

$$P^t \in \{0,1\}^{HxW}, \quad (2.2)$$

де H – висота кадру;

W – ширина кадру.

2.1.2 Модулі нейронної мережі

Нейронна мережа для детектування малопомітних рухомих об'єктів складається з двох головних модулів: модуля детектування та модуля узгодженості. В архітектурі цих модулів використовуються поєднання елементів архітектури трансформеру та згорткових нейронних мереж.

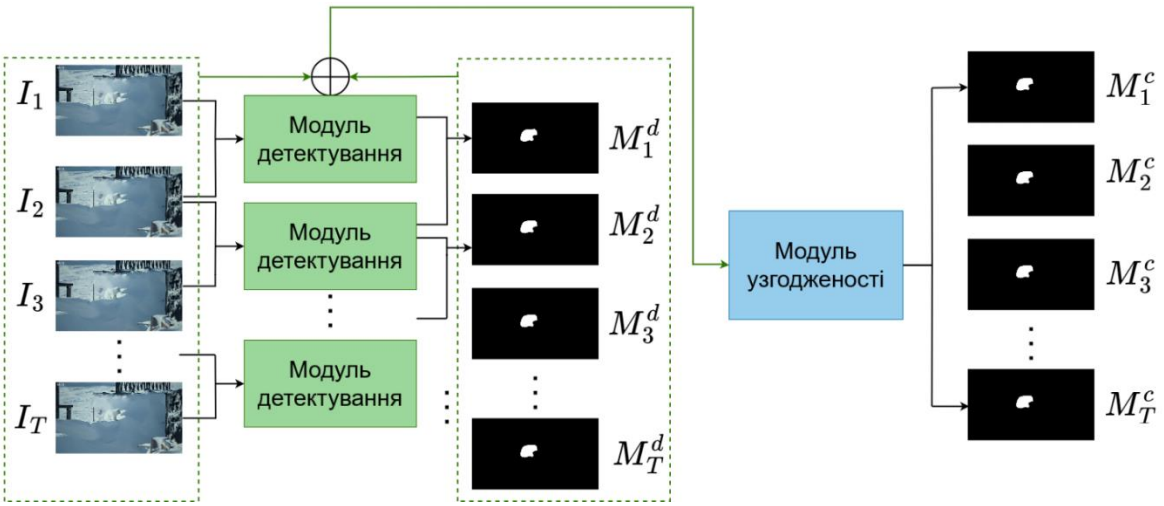


Рисунок 2.1 – Архітектура мережі для детектування малопомітних рухомих об'єктів

2.1.3 Модуль детектування

Модуль детектування приймає в якості свого входу два послідовні кадри з відеофрагмента та на їх основі будує бінарну маску об’єкта для обраного кадру.

Модуль детектування складається з трьох підмодулів: кодувальника, кореляційної піраміди та згорткового декодеру.

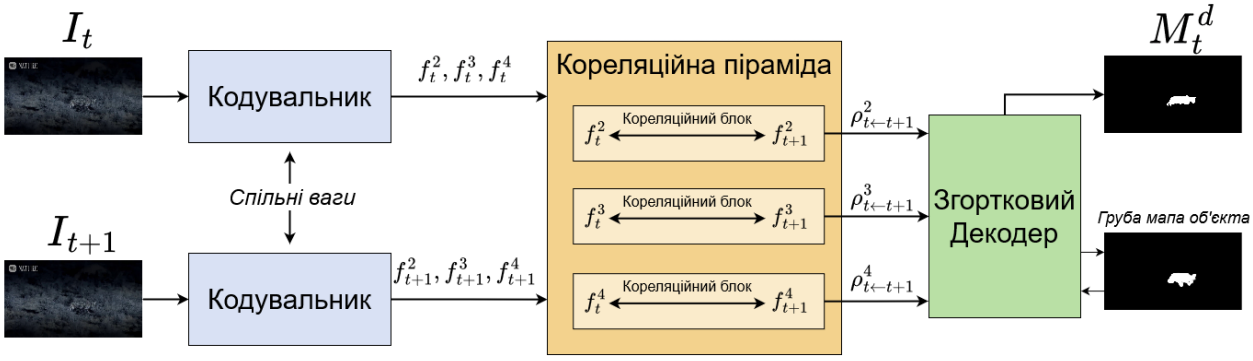


Рисунок 2.2 – Архітектура модуля детектування

2.1.4 Кодувальник

Кодувальник складається з двох ідентичних компонентів, що базовані на різновиді архітектури трансформера, призначеної спеціально для задач

комп’ютерного зору – pyramid vision transformer [11], та які пов’язані між собою за допомогою використання спільних ваг. Його функція полягає у знаходженні характерних ознак малопомітного рухомого об’єкту. Для цього він бере в якості входу два послідовних кадри $I_{t,}$ та I_{t+1} та будує для них карти ознак f_t та f_{t+1},. Цей процес кодування виконується чотири рази, при цьому при кожному разі застосовується різне масштабування.

В результаті виконання кожної ітерації i процесу кодування на виході отримується карти ознак f_t^i та f_{t+1}^i, розмір якої визначається так:

$$C_i \times \frac{H}{2^{i+1}} \times \frac{W}{2^{i+1}}, i \in \{1,2,3,4\}, \quad (2.3)$$

де H – висота кадру;

W – ширина кадру;

C – кількість каналів;

i – номер ітерації процесу кодування.

Саме ті карти ознак, що були побудовані під час останніх трьох ітерацій процесу кодування, будуть в подальшому використовуватися в роботі мережі. Тому для посилення ознак в цих трьох картах ознак застосовується компонент, що називається модулем посилення текстур (TEM) [5].

2.1.5 Кореляційна піраміда

Кореляційна піраміда складається з трьох компонентів, що називаються кореляційними блоками. Кожен з них приймає на вхід побудовані в результаті роботи кодувальника карти ознак f_t^i та f_{t+1}^i , $i \in \{2,3,4\}$ двох сусідніх кадрів, та розраховує їх кореляційний об’єм $C(I_t, I_{t+1})$.

Кореляційний об’єм – це чотирьохвимірний тензор, який містить візуальну кореляцію для пари попіксельних ознак з карт ознак двох сусідніх кадрів, розраховану за допомогою радіальної базисної функції.

Для пари карт ознак сусідніх кадрів $\{f_t^i,\ f_{t+1}^i\} \in \mathbb{R}^{C \times H' \times W'}$ кореляційний об’єм $C(I_t,\ I_{t+1}) \in \mathbb{R}^{H' \times W' \times H' \times W'}$ визначається наступним чином:

$$C(I_t, I_{t+1})_{xyuv} = \exp\left(\sum_c {f_t^i}_{xyc} \times {f_{t+1}^i}_{uvc}\right), \quad (2.4)$$

де c – індекс виміру каналів карт ознак.

Останні два виміри (uv) кореляційного об'єму $C(I_t, I_{t+1})_{xyuv}$ являються кореляційною матрицею. Саме вона описує відповідність між картами ознак обраного та сусіднього кадру у всіх просторових позиціях.

Для того, щоб здійснити агрегацію ознак на основі цієї відповідності, потрібно нормалізувати отриманий кореляційний об’єм $C(I_t, I_{t+1})_{xyuv}$ відносно суми останніх двох вимірів uv:

$$\tilde{C}(I_{t,}, I_{t+1})_{xyuv} = \frac{C(I_t, I_{t+1})_{xyuv}}{\sum_u \sum_v C(I_t, I_{t+1})_{xyuv}}. \quad (2.5)$$

Кореляційний об’єм враховує відповідність карт ознак лише у просторових вимірах. Тому для урахування інформації з канального рівня потрібно застосовувати згортку до карти ознак сусіднього кадру. В результаті такого застосування ми отримуємо уточнену карту ознак $\phi(I_{t+1}) \in \mathbb{R}^{C\times H'\times W'}$.

Карта агрегованих ознак з карт ознак сусідніх кадрів $\rho(I_{t\leftarrow t+1}) \in \mathbb{R}^{C\times H'\times W'}$ розраховується наступним чином:

$$\rho(I_{t\leftarrow t+1}) = \tilde{C}\left(I_{t,}, I_{t+1}\right)\phi(I_{t+1}). \quad (2.6)$$

Отримана карта агрегованих ознак $\rho(I_{t\leftarrow t+1})$ є результатом роботи кореляційного блоку.

Таким чином, виходом усієї кореляційної піраміди є три карти агрегованих ознак $\{\rho^i(I_{t\leftarrow t+1})\} \in \mathbb{R}^{C\times H/2^{i+1}\times W/2^{i+1}}$ $i \in \{2,3,4\}$.

2.1.6 Згортковий декодер

Згортковий декодер складається з компонентів уваги до групової зміни (GRA) [5] та декодеру сусідського з’єднання (NCD) [5]. Компонент уваги до групової зміни використовується для більш точного виявлення меж малопомітного об’єкта на кадрі відеофрагмента. Декодер сусідського з’єднання будує карту

орієнтовного місцезнаходження об'єкта, яка також використовується в інших компонентах.

Вихідним результатом згорткового декодера зокрема і модуля детектування взагалі є прогнозована бінарна маска об'єкта $\mathrm{M}_t^d, t \in [1:T]$, де верхній індекс d позначає, що це результат роботи лише першого модуля – модуля детектування.

2.1.7 Модуль узгодженості

Модуль узгодженості приймає на вхід бінарні маски M_t^d, отримані в результаті роботи модуля детектування, разом з відповідними їм кадрами відео, та виконує їх покращення, узгоджуючи карти ознак об'єкта з послідовних кадрів між собою.

Він використовує модуль детектування як основу, а конкретно його підмодулі: кодувальник та згортковий декодер. Особливим підмодулем модуля узгодженості є підмодуль Seq2Seq («послідовність-до-послідовності»), що використовує архітектуру трансформера, має в собі блоки нормалізованої самоуваги, та базований на моделі архітектури PNS-Net [12].

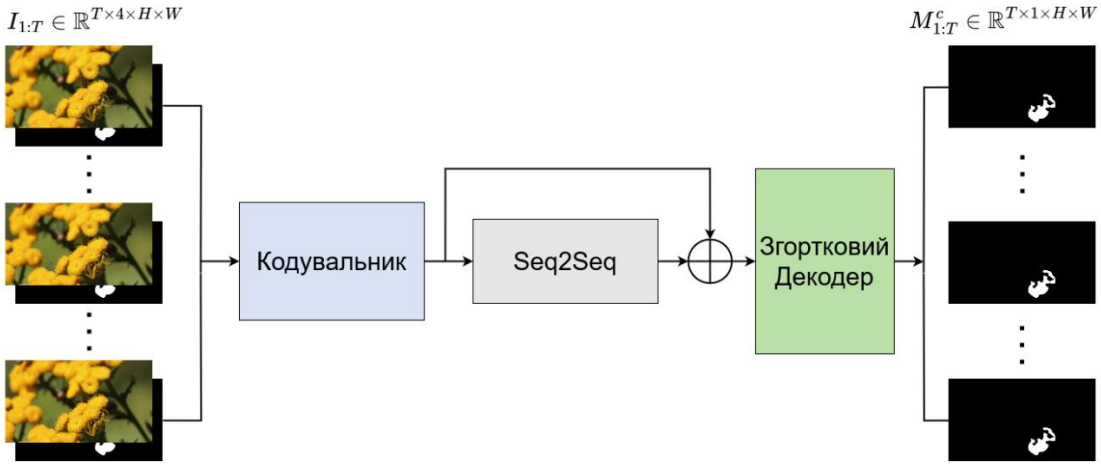


Рисунок 2.3 – Архітектура модуля узгодженості

Спочатку для кожного вхідного кадру I_t, $t \in [1:T]$ виконується конкатенація з відповідною йому прогнозованою модулем детектування бінарною маскою M_t^d, $t \in [1:T]$ у канальному вимірі. Потім кожний отриманий

кадр об'єднується в послідовність, яка утворює чотирьохвимірний тензор $X_t \in \mathbb{R}^{T\times 4\times H\times W},\ t \in [1:T]$.

Модуль узгодженості приймає цей утворений тензор X_t як вхід. Далі кодувальник вилучає ознаки з кадрів тензора та будує відповідні карти ознак. Підмодуль Seq2Seq будує просторово-часову матрицю для узгодження карт ознак усієї послідовності кадрів відеофрагменту між собою, виконує агрегацію ознак і в кінці синтезує агреговані ознаки з побудованою просторово-часовою матрицею. Виходом підмодуля Seq2Seq і є такі карти агрегованих ознак. Згортковий декодер приймає ці карти агрегованих ознак, виконує їх покращення та на виході видає остаточну послідовність прогнозованих бінарних масок $M^c_{1:T} \in \mathbb{R}^{T\times 1\times H\times W}$.

2.1.8 Тренування нейронної мережі

Тренування мережі відбувається у два етапи. Спочатку тренується модуль детектування. Після цього до моделі приєднується модуль узгодженості та виконується другий етап тренування, вже для всієї нейронної мережі.

Тренування модуля детектування виконується за допомогою мінімізації функції витрат з урахуванням положення пікселів (англ. pixel position aware) [13]:

$$\mathcal{L}_{ppa} = \mathcal{L}_{wbce} + \mathcal{L}_{wiou}, \tag{2.7}$$

де $\mathcal{L}_{wbce}$ – зважена функція витрат бінарної перехресної ентропії;

$\mathcal{L}_{wiou}$ – зважена функція витрат відношення перетину до об'єднання (коефіцієнт Жакара, англ. intersection-over-union).

Звичайна функція витрат бінарної перехресної ентропії є однією з найпоширеніших функцій витрат для задач детектування об'єктів [14; 15]. Проте її недоліками є: обчислення витрат кожного пікселя у відриві від глобального контексту зображення, слабка ефективність у роботі з зображеннями з великою кількістю фонових пікселів, однаковий підхід до всіх пікселів. Ці недоліки були узяті до уваги та усунуті у її зваженому варіанті, де кожному пікселю присвоюється певна вага [13]. Таким чином, більш проблемним пікселі (такі як пікселі країв

об’єктів, або пікселі вузьких чи подовжених областей) приділяється більше уваги, ніж до незначних (наприклад, фонових).

Функція витрат відношення перетину до об'єднання має також широке використання у задачах комп’ютерного зору [16]. На відміну від функції витрат бінарної перехресної ентропії вона фокусується не на кожному індивідуальному пікселі, а на всій глобальній структурі зображення. Але незважаючи на це, вона також не враховує різниці між різними пікселями. Це також було виправлено у зваженому варіанті цієї функції витрат [13].

Тренування модуля узгодженості виконується за гібридною функцією витрат [18]:

$$\mathcal{L}_{hybrid} = \mathcal{L}_{ce}^{w} + \mathcal{L}_{iou}^{w} + \mathcal{L}_{e}, \tag{2.8}$$

де $\mathcal{L}_{wbce}$ – зважена функція витрат перехресної ентропії;

$\mathcal{L}_{wiou}$ – зважена функція витрат відношення перетину до об'єднання;

$\mathcal{L}_{e}$ – функція витрат посиленого вирівнювання (англ. Enhanced-alignment).

Вище описані функції витрат фіксують ознаки на різних рівнях: зважена функція витрат бінарної перехресної ентропії – на попіксельному, зважена функція витрат відношення перетину до об'єднання – на усього рівні зображення. Для того, щоб фіксувати ознаки безпосередньо об’єкта була запропонована функція витрат посиленого вирівнювання [17].

Таким чином, використання гібридної функції витрат дає змогу нейронній мережі вчитись помічати ознаки детектуємого об’єкта на всіх рівнях: попіксельному, рівні зображення та рівні самого об’єкта.

2.1.9 Оцінка ефективності нейронної мережі

Оцінка ефективності роботи натренованої нейронної мережі для детектування малопомітних рухомих об’єктів відбувається за допомогою кількісних показників, зазначених у бенчмарку MoCA-Mask [9]. Чотири з цих показників, а саме *S-показник* S_α, *зважений F-показник* F_β^w, *показник посиленого*

вирівнювання E_ϕ та *середня абсолютна похибка* M використовуються також в бенчмарку датасета COD10K [5] та широко застосовуються для оцінки ефективності моделей нейронних мереж в сфері детектування прихованих об’єктів.

S-показник S_α оцінює два види структурної схожості: орієнтованої на регіон та орієнтованої на об’єкт [19].

Зважений F-показник F_β^w об’єднує зважені властивості бінарної маски під одною оцінкою [20].

Показник посиленого вирівнювання E_ϕ оцінює відповідність на попіксельному рівні разом зі статистикою на рівні зображення [17]

Середня абсолютна похибка є одним з найпоширеніших показників для вимірювання ефективності нейронних мереж для детектування об’єктів [21]. Середня абсолютна похибка M між отриманою маскою S та бінарною цільовою маскою GT визначається так:

$$M = \frac{1}{H \times W} \sum_{x=1}^{H} \sum_{y=1}^{W} |S(x,y) - GT(x,y)|, \tag{2.9}$$

де H – висота маски;

W – ширина маски.

Інші два показника: *середній коефіцієнт Дайса* $mDic$ та *середнє відношення перетину до об'єднання* $mIoU$ [9]. Перший з них вимірює схожість двох наборів даних, другий – величину перетинання двох масок

2.2 Математичний апарат

2.2.1 Згортка як математична операція

Згортка – це математична операція, яка об'єднує дві функції для створення третьої функції, яка показує як форма однієї змінюється іншою. Вона широко використовується в обробці сигналів і аналізі зображень для вилучення значущої

інформації (наприклад, ознак об'єктів) та виконання різних перетворень сигналів і зображень.

Математично згортка визначається як інтеграл від добутку двох функцій, після того як одна з них віддзеркалена відносно осі y та зміщена.

Нехай f – це перша функція, а g – друга функція. Тоді згортка цих двох функцій позначається як $(f \times g)(t)$ та виражається наступним чином:

$$(f \times g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau. \quad (2.10)$$

Функцію g називають фільтром. Вона віддзеркалюється, а потім ковзає вздовж горизонтальної осі x. Для кожної позиції обчислюється площа перетину між g і оберненою f. Ця площа перетину і є значенням згортки в цій конкретній позиції.

Згортка має декілька важливих властивостей, включаючи комутативність, лінійність та асоціативність.

Комутативність означає, що порядок згортки не впливає на результат, тому $(f \times g)$ еквівалентна $(g \times f)$:

$$f \times g = g \times f. \quad (2.11)$$

Лінійність означає, що згортка лінійної комбінації функцій – це те й саме, що лінійна комбінація їх окремих згорток:

$$f \times (g + h) = (f \times g) + (f \times h). \quad (2.12)$$

Асоціативність означає, що згортка декількох функцій може бути виконана в будь-якому порядку без впливу на кінцевий результат:

$$(f \times g) \times h = (f \times g) \times h. \quad (2.13)$$

Для двох дискретних функцій згортка позначається як:

$$(f \times g)(n) = \sum_{m=-\infty}^{\infty} f[m]g[n-m]. \quad (2.14)$$

2.2.2 Взаємна кореляція

Незважаючи на те, що згорткові нейронні мережі так називаються, насправді в них використовується операція не згортки, а взаємної кореляції.

Взаємна кореляція двох комплексних функцій $f(t)$ та $g(t)$ дійсної змінної t позначається $(f \star g)(t)$ та визначається так:

$$(f \star g)(t) = f(-t) \times g(t). \tag{2.15}$$

Виходячи з визначення операції згортки, слідує, що:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(-\tau) g(t - \tau) d\tau. \tag{2.16}$$

Нехай $\tau' \equiv -\tau, d\tau' = -d\tau$, звідси:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau') g(t + \tau')(-d\tau') \tag{2.17}$$

$$= \int_{-\infty}^{\infty} f(\tau) g(t + \tau) d\tau \tag{2.18}$$

Як видно, порівнюючи формули (2.10) та (2.18), згортка та взаємна кореляція відрізняються лише знаком у аргументі функції g. Іншими словами, взаємна кореляція не включає в себе віддзеркалення фільтра g.

Головним наслідком цього є відсутність асоціативності у взаємній кореляції, на відміну від згортки.

$$(f \star g) \star h \neq (f \star g) \star h. \tag{2.19}$$

У всьому іншому, згортка та взаємна кореляція є ідентичними.

2.2.3 Двовимірна згортка

Двовимірні сценарії є найбільш поширеними у обробці зображень методами нейронних мереж. Тому для них застосовується двовимірна згортка (англ. 2D convolution).

Під двома вимірами найчастіше мається на увазі висота та ширина зображення. В згорткових нейронних мережах вхідні дані подаються у вигляді

чотирьохвимірного тензора, вимірами якого є: пакетні одиниці (англ. batch samples), канали ознак, висота та ширина зображення. Базові операції згортки виконуються лише у вимірах висоти та ширини зображення, не перетинаючи канали ознак чи пакетні одиниці. Саме це є причиною, чому цей тип згортки називається двовимірним, хоча при цьому використовуються чотирьохвимірні тензори.

Нехай матриця $I(x, y)$ – вхідне зображення розміром (N_x, N_y), матриця $K(u, v)$ – ядро (фільтр) з N_u рядками та N_v стовпчиками. Звідси, двовимірна згортка між I та K визначається наступним чином:

$$I * K = \sum_{u=1}^{N_u} \sum_{v=0}^{N_v} I(x + v, y + u) K(u, v). \tag{2.20}$$

Як було вказано вище, в контексті нейронних мереж під «згорткою» мається на увазі взаємна кореляція, адже перед добутком не відбувається віддзеркалення ядра. Причина того, що взаємна кореляція в контексті нейронних мереж називається згорткою, наступна: в нейронних мережах елементи в ядрі є параметрами, значення яких отримуються в процесі навчання нейронної мережі. Тому віддзеркалення ядра лише змінює положення елементів, а не їх значення, не впливаючи на результат операції.

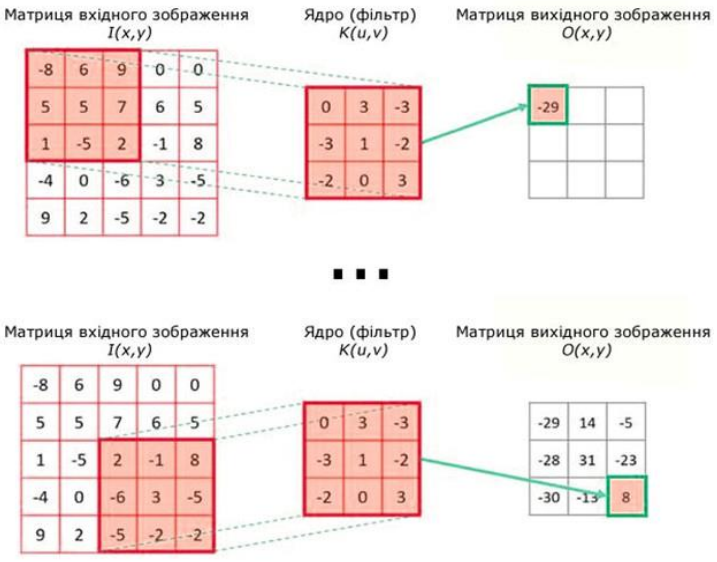


Рисунок 2.4 – Процес виконання двовимірної згортки

Важливими параметрами двовимірної згортки є крок (англ. stride) та доповнення (англ. padding). Крок визначає кількість пікселів, на яку ядро переміщується по вхідному зображенню. Доповнення розширює вхідне зображення за допомогою додавання додаткових рядків та стовпчиків за межами оригінального зображення. Найчастіше використовується такий тип доповнення, який називається доповненням нулями (zero-padding). У цьому випадку всі пікселі у додаткових рядках и стовпчиках заповнюються нульовими значеннями.

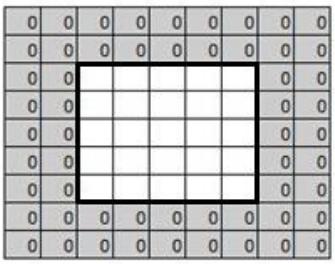


Рисунок 2.5 – Доповнення нулями

2.3 Навчання нейронної мережі як мінімізація функції витрат

Навчання нейронної мережі полягає у знаходженні оптимальних значень параметрів мережі (перш за все: ваг та зміщень). Математично це вирішується за допомогою мінімізації *функції витрат* шляхом застосування ітеративного методу *стохастичного градієнтного спуску*.

2.3.1 Функція витрат/втрат

В науковій та технічній літературі з штучних нейронних мереж зустрічаються як термін «функція витрат» (англ. cost function), так і «функція втрат» (англ. loss function). Ці два терміни дуже часто використовують як синонімічні. [22] Однак деякі автори проводять чітку різницю між ними [23]. На їхню думку, функція витрат вимірює похибку моделі на групі об'єктів, тоді як функція втрат має справу лише з одним екземпляром даних. Іншими словами, функція втрат вимірює похибку одного спостереження, тоді як функція витрат вимірює похибку для всіх спостережень. Функція витрат може бути сумою функцій втрат.

Функцією витрат називається функція, яка кількісно оцінює похибку між прогнозованим вихідними значеннями нейронної мережі та фактичними вихідними (або цільовими) значеннями і представляє її у вигляді одного дійсного числа.

Функція витрат в контексті нейронних мереж слугує мірою того, наскільки добре ця мережа виконує задачу, адже надає кількісну оцінку помилки або відхилення між прогнозованим результатом та істинним значенням.

2.3.2 Стохастичний градієнтний спуск

Стохастичний градієнтний спуск – це ітеративний метод знаходження локального мінімуму функції за допомогою руху вздовж градієнта. Градієнт – вектор, що своїм напрямом вказує напрям найшвидшого росту чи спаду певної скалярної величини.

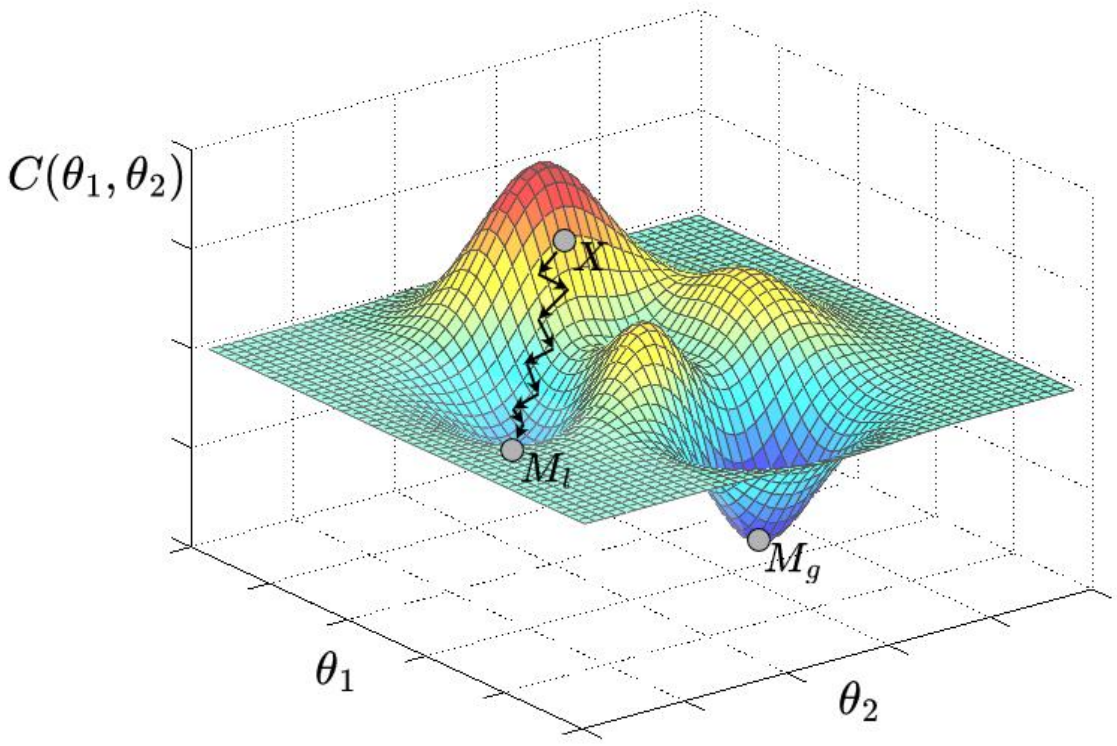


Рисунок 2.6 – Тривимірний графік функції витрат $C(\theta_1, \theta_2)$.
M_g – глобальний мінімум функції, M_l – локальний мінімум функції, X – початкові значення параметрів $\theta_1,\ \theta_2$

Стохастичний градієнтний спуск мінімізує деяку функцію витрат $C(\theta)$ ітеративно змінюючи вектор параметрів θ на деяку невелику величину, що визначена від’ємним градієнтом одного тренувального набору даних b.

Алгоритм виконання стохастичного градієнтного спуску наступний:

1) дані з набору даних b перемішуються випадковим чином;

2) обирається значення темпу навчання η (гіперпараметр, який регулює величину кроку вздовж градієнта) ;

3) обирається певний вектор параметрів як θ початкова точка;

4) для $i = 1,2, \dots, n$, обчислити:

$$\theta := \theta - \eta \nabla C_i(\theta); \quad (2.21)$$

5) повторювати крок 4, доки не буде досягнутий локальний мінімум функції.

2.4 Вибір апаратного забезпечення АПЗ

Нейронна мережа для детектування малопомітних рухомих об'єктів може бути розгорнута на різноманітних пристроях. Для цього може бути використовуватися як звичайний персональний комп’ютер, так і невеликий одноплатний комп’ютер. Головною вимогою є наявність графічного процесора та достатньо потужних процесора (чи мікропроцесора) та пам’яті для забезпечення ефективної роботи нейронної мережі

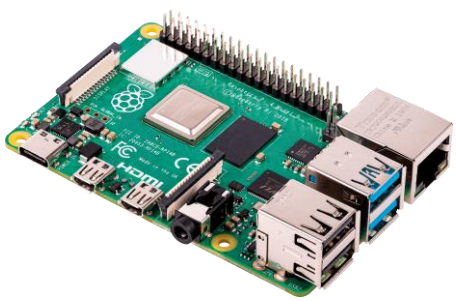
Одноплатний комп'ютер – це повноцінна комп'ютерна система, всі компоненти якої знаходяться на одній друкованій платі. Основними компонентами одноплатного комп’ютера є процесор, пам'ять та інтерфейси вводу/виводу. Багато сучасних одноплатних комп’ютерів також мають вбудовані графічні процесори. На відміну від традиційних комп'ютерів, які мають окремі компоненти, та які розподілені по декількох платах або модулях, одноплатні комп’ютери є компактними і автономними обчислювальними пристроями, які призначені для забезпечення базової функціональності комп'ютера в невеликому форм-факторі.

Перевагами одноплатних комп’ютерів є компактність, автономність, менша за персональний комп’ютер вартість та значно нижче енергоспоживання. Саме тому вони використовуються у таких сферах як вбудовані системи та прототипування. Оглядаючись на ці переваги, було прийнято рішення використовувати одноплатний комп’ютер в якості основи апаратної частини апаратно-програмного комплексу з нейронною мережею для детектування малопомітних рухомих об’єктів.

Ринок одноплатних комп'ютерів є досить великим та при цьому стрімко розвивається. До вже відомих брендів, таких як Raspberry Pi, Asus Tinkerpad, BeagleBone, NVIDIA Jetson, кожен рік приєднується багато нових. Відповідно до свого призначення одноплатні комп’ютери поділені на категорії: бюджетні для навчання, більш потужні для прототипування та дорогі, проте повноцінні і надійні – для використання на промислових та ін. виробництвах.

У зазначеній категорії для прототипування найбільш популярними та доступними одноплатними комп’ютерами, які можна використати для роботи з нейронною мережею, є дві моделі: Raspberry Pi 4 Model B 4GB та NVIDIA Jetson Nano Developer Kit.

Таблиця 2.1 – Порівняння основних характеристик одноплатних комп’ютерів Raspberry Pi 4 Model B 4GB та NVIDIA Jetson Nano Developer Kit

Модель одноплатного комп’ютера	Raspberry Pi 4 Model B 4GB [24]	NVIDIA Jetson Nano Developer Kit [25]
Зовнішній вигляд		
Оперативна Пам’ять	4 GB LPDDR4	4 GB 64-bit LPDDR4 25.6 GB/s

Центральний Процесор	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz	Quad-core ARM A57 @ 1.43 GHz
Графічний процесор	Broadcom VideoCore VI (32-bit)	NVIDIA Maxwell w/ 128 CUDA cores @ 921 Mhz
Роздільна здатність та стандарт відеокодування	1080p 30 fps (H.264)	4K 30 fps, 4x 1080p 30 fps, 9x 720p 30 fps (H.264/H.265)
Роздільна здатність та стандарт відеодекодування	1080p 60 fps (H.264), 4K 60 fps (H.265);	4K 60 fps, 2x 4K 30 fps, 8x 1080p 30 fps, 18x 720p 30 fps (H.264/H.265)
Інтерфейси вводу/виводу	2 × USB 3.0 2 × USB 2.0	4 × USB 3.0, 1 × USB 2.0 Micro-B
Мережеві інтерфейси	Gigabit Ethernet, Wifi 802.11ac	Gigabit Ethernet, M.2 Key E
Сховище	Micro-SD	Micro-SD
Вартість (станом на 20.05.2023)	8486 грн [26]	9614 грн [27]

NVIDIA Jetson Nano Developer Kit, будучи трохи дорожчим за Raspberry Pi 4 Model B 4GB, проте є більш потужною моделлю. Головними перевагами, що є ключовими для реалізації на одноплатному комп’ютері нейронної мережі для детектування малопомітних рухомих об’єкті в – це наявність у NVIDIA Jetson Nano Developer Kit графічного процесора та краща якість кодування та декодування відео. Саме через ці переваги NVIDIA Jetson Nano Developer Kit було обрано в якості бази для апаратної частини апаратно-програмного комплексу.

Важливою особливістю NVIDIA Jetson Nano Developer Kit є його початкова орієнтованість на реалізацію проєктів у сфері машинне навчання. Разом з Jetson Nano постачається набір програмних інструментів для розробників JetPack SDK, що містить повноцінне Linux-середовище з увімкненим прискоренням графіки. JetPack SDK має повну підтримку NVIDIA CUDA Toolkit і таких необхідних бібліотек як cuDNN і TensorRT. До того ж JetPack SDK надає можливість встановлення таких популярних фреймворків машинного навчання, як TensorFlow,

PyTorch, Caffe, Keras і MXNet, а також фреймворків для розробки нейронних мереж в галузях комп'ютерного зору і робототехніки, таких як OpenCV і ROS.

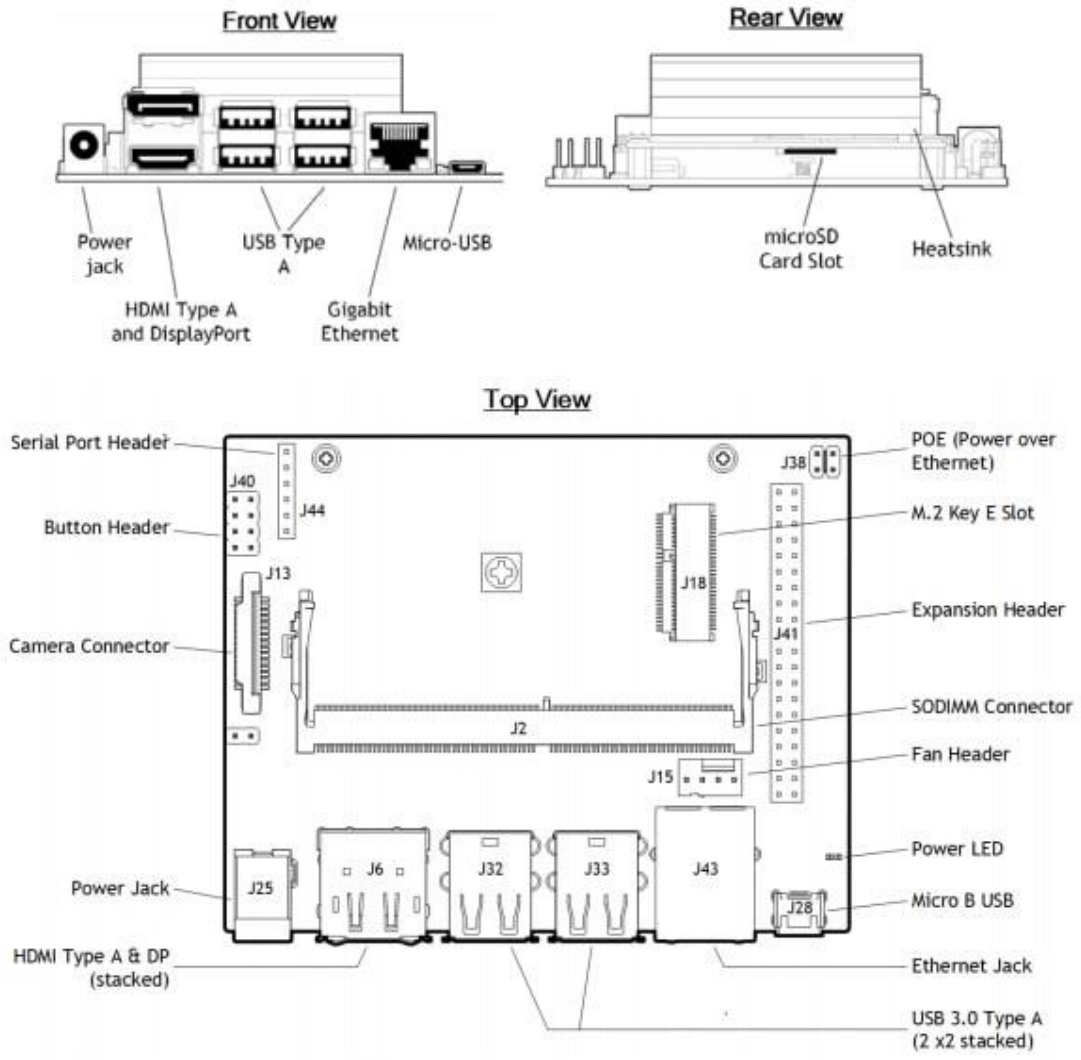


Рисунок 2.7 – NVIDIA Jetson Nano Developer Kit

Вбудована повна сумісність з популярними фреймворками та платформою NVIDIA для штучного інтелекту робить розгортання додатків, що використовують нейронні мережі, на Jetson Nano простим та зручним.

2.4.1 Вибір камери

Для забезпечення власної автономності апаратно-програмний комплекс з нейронною мережою для детектування малопомітних рухомих об’єктів повинен

мати під’єднану до одноплатного комп’ютера камеру. Це дозволить працювати нейронній мережі не тільки на тестових датасетах, але й на відеоматеріалі з камери у реальному часі.

NVIDIA співпрацює з великою кількістю виробників камер для одноплатних ком’ютерів для забезпечення сумісності їх продуктів з Jetson Nano Developer Kit. Деякі виробники також мають власні бібліотеки сумісності для непідтримуваних NVIDIA лінійок камер – наприклад, компанія Arducam має для цих цілей бібліотеку Jetvariety.

Хоча використання нативних камер є більш бажаним, вони можуть бути відсутніми на місцевому ринці. Тому доцільним є звернути увагу на більш доступні сторонніх виробників.

Одним з таких виробників є компанія Waveshare Electronics. Вона є провідним виробником одноплатних комп’ютерів та їх комплектуючих – дисплеїв, відеокамер, рушіїв, сенсорів та ін. Продукти виробника Waveshare Electronics є поширеними на українському ринку одноплатних комп’ютерів та їх комплектуючих. Їх доступність та менша вартість, при збереженні належної якості, є.

В якості відеокамери для Jetson Nano було обрано модель IMX477-160 12.3MP виробника Waveshare Electronics. Її технічні характеристики наведено у Таблиці 2.2.

Рисунок 2.8 – Зовнішній вигляд відеокамери IMX477-160 12.3MP

Таблиця 2.2 – Технічні характеристики відеокамери IMX477-160 12.3MP [28]

Матриця	MX477
Роздільна здатність	12.3 мегапікселів, 4056 × 3040
Довжина діагоналі CMOS	7.9 мм
Розмір пікселя	1.55 мкм (H) × 1.55 мкм (V)
Діафрагма	2.2
Фокусна відстань	5.52
Поле зору (FOV):	160°(D) 118°(H) 87°(V)
Спотворення:	<16%
Робоча напруга:	3.3 В
Інтерфейс підключення камери	SCI/MIPI
Формат вихідного відео	RAW12/1 /8, COMP8
Вартість (станом на 20.05.2023) [29]	3 296 грн

Рисунок 2.9 – Габаритні розміри відеокамери IMX477-160 12.3MP (у мм)

Відеокамера IMX477-160 12.3MP розроблена під одноплатні комп’ютери Raspberry Pi Compute Module і Working with Jetson Nano та призначена для додатків з використанням штучного інтелекту. Саме через ці особливості та потужні технічні можливості серед усіх моделей відеокамер було обрано модель IMX477-160 12.3MP.

Для підключення відеокамери IMX477-160 12.3MP до Jetson Nano Developer Kit потрібно спочатку встановити драйвер NVIDIA Jetson IMX477 HQ RPI V3 camera driver. Далі потрібно вставити кабель камери металевою стороною до радіатора в SCI-коннектор на платі Jetson Nano.

Рисунок 2.10 – Підключення відеокамери IMX477-160 12.3MP до одноплатного комп’ютера Jetson Nano Developer Kit

Висновок до розділу 2

Нейронна мережа для детектування малопомітних рухомих об'єктів має двохмодульну архітектуру. Вона складається з модуля детектування та модуля узгодженості, які поєднують елементи архітектури трансформеру та згорткових нейронних мереж. Модуль детектування приймає в якості входу два послідовні кадри з відеофрагмента та на їх основі будує бінарну маску об’єкта для обраного кадру. Модуль узгодженості приймає на вхід побудовані модулем детектування бінарні маски об'єкта разом з відповідними їм кадрами відео, та виконує їх покращення, узгоджуючи карти ознак об’єкта з послідовних кадрів між собою.

Тренування мережі відбувається у два етапи. Спочатку тренується модуль детектування. Після цього до моделі приєднується модуль узгодженості та виконується другий етап тренування, вже для всієї нейронної мережі. Для тренування модуля детектування використовується функція витрат з урахуванням

положення пікселів. Тренування модуля узгодженості виконується за гібридною функцією витрат. Оцінка ефективності роботи натренованої нейронної мережі для детектування малопомітних рухомих об’єктів відбувається за допомогою кількісних показників, зазначених у бенчмарку MoCA-Mask.

Елементи згорткової мережі, що використовується в нашій архітектурі, застосовують операцію двовимірної згортки для роботи з кадрами відеофрагменту. Двовимірна згортка являє собою операцію над двома двовимірними матрицями: матриці-репрезентації пікселів зображення та ядра (фільтра), що «ковзає» по неї.

Навчання нейронної мережі полягає у знаходженні оптимальних значень параметрів мережі (ваг та зміщень). Для цього потрібно застосувати функцію витрат та мінімізувати її. Конкретним алгоритмом мінімізації функції витрат є стохастичний градієнтний спуск.

В якості апаратного забезпечення для нейронної мережі було обрано одноплатний комп’ютер NVIDIA Jetson Nano Developer Kit та відеокамеру IMX477-160 12.3MP.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ДЕТЕКТУВАННЯ МАЛОПОМІТНИХ РУХОМИХ ОБ’ЄКТІВ

Нейронна мережа для детектування малопомітних рухомих об’єктів реалізована програмними засобами мови програмування Python, як найпопулярнішої мови програмування для машинного навчання, фреймворку PyTorch, який використовується у задачах комп’ютерного бачення, та інтегрованого середовища розробки PyCharm. Для виконання розрахунків ефективності роботи нейронної мережі було застосовано програмне забезпечення MATLAB.

3.1 Мова програмування Python

Python – це високорівнева інтерпретована мова програмування з динамічною типізацією, головними особливостями якої є простота, читабельність та універсальність. Вона містить обширну стандартну бібліотеку та широку екосистему сторонніх бібліотек. Все це робить Python популярною та широко застосованою мовою програмування у таких сферах як: наука про дані (англ. Data Science), наукові розрахунки та машинне навчання.

Мова програмування Python була розроблена Гвідо ван Россумом та представлена у 1991 році. На даний момент активно розвивається версія Python 3.

Python має простий і читабельний синтаксис, який дуже схожий на псевдокод, а звідси – на звичайну природну мову. Це полегшує написання і розуміння коду, що робить Python ідеальним вибором для вчених, що займаються науковими розрахунками.

Рисунок 3.1 – Логотип мови програмування Python

Python має динамічну типізацію та автоматичне керування пам’яттю (також відоме як автоматичне збирання сміття). Динамічна типізація дозволяє однієї й тієї ж самої змінній приймати значення різних типів. Автоматичне збирання сміття підвищує безпеку програми, запобігаючи витоку пам’яті, та полегшує процес програмування.

Простота і легкість використання Python забезпечує швидке створення прототипів та експерименти в машинному навчанні. Це дозволяє дослідникам і розробникам швидко реалізовувати ідеї, тестувати різні алгоритми та ітерації на своїх моделях. Така гнучкість має вирішальне значення в такій галузі, як машинне навчання.

Вище наведені переваги мови програмування Python сприяли її швидкому поширенню та створення великої спільноти розробників та дослідників у галузі машинного навчання, що займається активною розробкою різноманітних бібліотек та фреймворків, спеціально розроблених під задачі машинного навчання та аналізу даних. До них відносяться такі популярні бібліотеки, як NumPy, Pandas, PyTorch, Tensorflow, які надають потужні інструменти для маніпулювання даними, чисельних обчислень, візуалізації та алгоритмів машинного навчання.

3.2 Фреймворк Pytorch

PyTorch – це фреймворк для машинного навчання з відкритим вихідним кодом, який розробляється лабораторією штучного інтелекту Meta AI. PyTorch є фреймворком, що базується на бібліотеці Torch та має інтерфейси на мовах програмування Python та C++. Найбільш часто він використовується для побудови штучних нейронних мереж для вирішення задач комп’ютерного бачення та обробки природньої мови.

Його головними особливостями є:

а) обчислення тензорів з сильним прискоренням за допомогою використання графічних процесорів;

б) використання автоматичного диференціювання на основі плівки (англ. tape-based autograd) при глибокому навчанні.

Рисунок 3.2 – Логотип фреймворку PyTorch

PyTorch використовує Python як головну мову програмування та поділяє його принципи дизайну. Першочерговою метою PyTorch є зручність користування і лише потім – прийнятна продуктивність. За основний принцип PyTorch бере простоту коду, тобто його експліцитність. Простота та наочність будівних блоків робить код легшим для розуміння та налагодження.

PyTorch має також інтерфейс на мові програмування C++. Проте Python-інтерфейс є більш досконалим та перспективним. Саме застосування мови програмування Python дозволяє використовувати найважливіші її бібліотеки для роботи з науковими розрахунками та машинного навчання – NumPy, SciPy, scikit-learn та ін.

Базовим та основним блоком PyTorch є тензор. Тензори – це матрично-подібні структури даних. За властивостями та функціональністю вони дуже схожі на масиви з бібліотеки Numpy. Головною особливістю, що робить тензори з PyTorch відмінними від масивів Numpy – це їх можливість працювати як на центральному процесорі, так і на графічному процесорі.

Використання прискорення на графічному процесорі робить PyTorch привабливим для задач машинного навчання. Безшовна інтеграція з графічними процесорами дозволяє проводити високопродуктивні обчислення і прискорювати навчання глибоких нейронних мереж. Це досягається за допомогою використання технології CUDA – платформи для паралельних розрахунків та програмного інтерфейсу, який дозволяє програмному забезпеченню використовувати певні типи графічних процесорів для виконання загальних розрахунків. Таке використання

потужності графічних процесорів для виконання паралельних обчислень призводить до значного прискорення працездатності моделей порівняно з традиційними обчисленнями на базі CPU.

Більшу продуктивність під час глибокого навчання моделей нейронних мереж та їх роботи PyTorch забезпечує за допомогою використання автоматичного диференціювання на основі плівки. Його особливість полягає у тому, що під час фази прямого поширення плівка автоматичного диференціювання запам’ятовує всі операції, що були виконані, а потім, на фазі оберненого поширення, вона просто відтворює запам’ятовані операції.

PyTorch має багату колекцію власних бібліотек для вирішення задач з різних галузей: TorchAudio – для обробки звукових та інших сигналів, TorchText – для обробки природної мови, TorchVision – для комп’ютерного бачення. Окрім цього фреймворк PyTorch підтримує велику екосистему сторонніх бібліотек, програмних засобів та платформ для машинного навчання.

Загалом, PyTorch є популярним фреймворком для машинного навчання, відомим своєю гнучкістю та простотою, автоматичним диференціюванням на основі плівки, прискоренням на графічних процесорах та широкою підтримкою різносторонніх бібліотек. Він дозволяє дослідникам та розробникам ефективно та швидко створювати, навчати і розгортати різні типи нейронних мереж для широкого спектру застосувань.

3.3 Інтегроване середовище розробки PyCharm

PyCharm – це інтегроване середовище розробки (IDE), спеціально розроблене для програмування на Python. Воно розроблене компанією JetBrains і доступне у двох версіях: безплатної з відкритим вихідним кодом PyCharm Community Edition та пропрієтарної PyCharm Professional Edition з додатковим функціоналом. PyCharm надає повний набір інструментів та функцій для підвищення продуктивності розробки на Python. PyCharm пропонує зручний інтерфейс,

розширені можливості редагування коду, підтримку налагодження та інтеграцію з популярними системами контролю версій.

PyCharm має потужний редактор коду з функціями підсвічування синтаксису, завершення коду та навігації по коду. Він пропонує інтелектуальні пропозиції щодо коду, автоматичні відступи та параметри форматування коду для підвищення ефективності кодування. Редактор також підтримує операції рефакторингу, дозволяючи розробникам легко перейменовувати змінні, витягувати код у функції тощо.

Рисунок 3.3 – Логотип інтегрованого середовища розробки PyCharm

Наявність вбудованого налагоджувача у PyCharm дозволяє розробникам переглядати код, встановлювати точки зупинки, перевіряти змінні та аналізувати потік програми. Цей функціонал налагодження допомагає виявляти та виправляти помилки, підвищуючи якість та надійність Python-додатків.

PyCharm пропонує інтегровані інструменти для написання та запуску тестів. Він підтримує популярні фреймворки тестування, такі як pytest, unittest та doctest, що полегшує написання та виконання тестів для коду на Python. Крім того, PyCharm включає інструменти профілювання для виявлення вузьких місць у продуктивності та оптимізації виконання коду.

У інтегрованому середовищі розробки PyCharm є можливість створення та керування віртуальними середовищами, які забезпечують ізольоване середовище виконання Python для різних проєктів. PyCharm також підтримує інтеграцію з популярними менеджерами пакетів, такими як pip та conda, спрощуючи встановлення та керування пакунками та залежностями Python.

PyCharm легко інтегрується з системами контролю версій, наприклад з Git. Він надає такі функції, як фіксація, управління гілками, вирішення конфліктів та візуальні інструменти для порівняння, що дозволяє розробникам ефективно співпрацювати над проєктами та відстежувати зміни коду.

PyCharm має вбудовану підтримку наукових бібліотек, таких як NumPy, SciPy та Matplotlib. Це полегшує роботу з великими масивами даних та відкриває такі можливості, як інтерактивне дослідження даних, візуалізація масивів та ін.

3.4 MATLAB

MATLAB – це пропрієтарне програмне середовище, призначене для чисельних обчислень і наукових досліджень, а також мова програмування, що використовується у даному середовищі. Програмне забезпечення MATLAB було розроблено компанією MathWorks. MATLAB надає різноманітний набір інструментів і функцій для аналізу даних, розробки алгоритмів, візуалізації та чисельних обчислень.

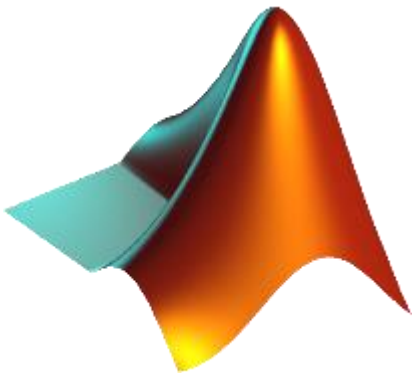


Рисунок 3.4 – Логотип програмного забезпечення MATLAB

Програмне забезпечення MATLAB надає широкий спектр вбудованих математичних функцій і бібліотек для виконання різноманітних чисельних обчислень. Це включає в себе диференційні рівняння, функції для інтерполяції, оптимізації, обробки сигналів, статистики та багато іншого, що робить його потужним інструментом для наукових та інженерних розрахунків.

MATLAB підтримує усі необхідні функції та інструменти для виконання матричних операцій та обчислень лінійної алгебри. Це робить виконання користувачем операцій над масивами і матрицями ефективним, легким та зручним. Таким чином, програмне забезпечення MATLAB є добре придатним для задач, пов'язаних з великомасштабними чисельними обчисленнями.

Середовище MATLAB пропонує багаті можливості візуалізації для створення графіків, діаграм і діаграм для дослідження та презентації даних. MATLAB надає гнучку систему побудови графіків з опціями для налаштування та коментування візуалізацій, що робить його корисним для аналізу даних та передачі результатів.

Також програмне забезпечення MATLAB має обширні та зручні інструменти для розробки та реалізації різноманітних алгоритмів. Однойменна мова програмування, що використовується в MATLAB, також підтримує використання концепцій об'єктно-орієнтованого програмування. MATLAB має всі необхідні засоби інтегрованого середовища розробки, що включає в себе налагоджувач і профайлер. Мова програмування з MATLAB має синтаксис, який легко читати і писати, що дозволяє користувачам швидко створювати прототипи і тестувати свої алгоритми.

MATLAB підтримує широку інтеграцію з іншими мовами програмування, базами даних та програмними інструментами. Він також надає можливості для розгортання коду MATLAB як автономних додатків, веб-додатків або як частини більших програмних систем.

Загалом, MATLAB є потужним програмним забезпеченням для виконання науково-технічних розрахунків, яке широко використовується в академічних і дослідницьких установах, інженерних і наукових сферах. Однією з таких сфер є побудова нейронних мереж для детектування малопомітних рухомих об’єктів. Тріальна версія середовища MATLAB використовується в даній роботі для розрахунку кількісних показників ефективності роботи натренованої нейронної мережі.

3.5 Реалізація нейронної мережі для детектування малопомітних рухомих об’єктів засобами мови програмування Python

3.5.1 Структура проєкту з нейронною мережею

Проєкт з нейронною мережею для детектування малопомітних рухомих об’єктів має наступну структуру:

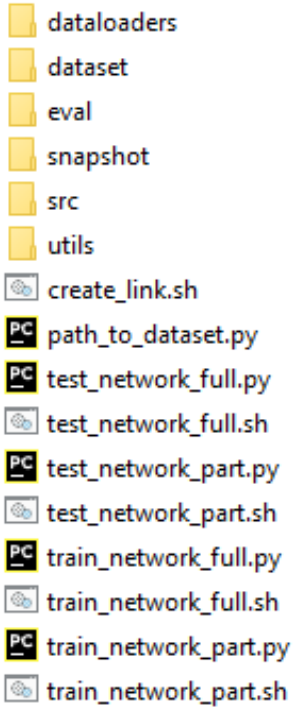


Рисунок 3.5 – Структура проєкту з нейронною мережею

Директорія *dataloders* містить класи типу Dataloader бібліотеки Pytorch. Вони призначені для підготовки даних з датасету до подальшого їх використання у нейронній мережі. За їх допомогою можна розбивати дані на пакети, перемішувати їх, об’єднувати дані з декількох датасетів. Також вони мають важливе значення для збільшення продуктивності тренування та тестування мережі, адже класи Dataloader дозволяють використовувати багатопроцесорну обробку даних та завантажувати дані безпосередньо на тензори CUDA.

Директорія *dataset* містить кадри відео з датасету MoCa-Mask. Вона має дві піддиректорії: *TestDataset_per_sq* та *TrainDataset_per_sq*. Перша призначена для тестування мережі, друга – для тренування мережі. Обидві директорії містять

відеофрагменти, розбиті по кадрам у форматі jpg, та бінарні маски цих кадрів у форматі png. У директорії TestDataset_per_sq містяться 16 відеофрагментів, а у TrainDataset_per_sq – 71 відеофрагмент.

Директорія *eval* містить файли для програмного забезпечення MATLAB. За допомогою цих файлів проводяться розрахунки ефективності мережі за показниками, що зазначені у бенчмарку. Результати розрахунків поміщуються у піддиректорію *Results*.

Директорія *src* є найголовнішою в проєкті. Саме в ній знаходиться програмна реалізація модулів нейронної мережі для детектування малопомітних рухомих об’єктів.

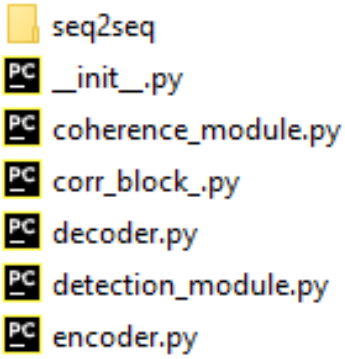


Рисунок 3.6 – Зміст директорії src проєкту з нейронною мережою

У директорії *utils* містяться різноманітні допоміжні функції та класи, що використовуються в інших компонентах. Наприклад файл Hybrid_Eloss.py містить функцію розрахунку гібридної функції витрат, що використовується при тренування мережі.

У *кореневій* директорії також знаходяться скрипти для тренування та тестування нейронної мережі для детектування малопомітних рухомих об’єктів. Файли, що містять в назві «part» відносяться до моделі мережі, що містить лише перший модуль – модуль детектування. Файли, що містять в назві «full» відносяться до мережі з повноцінною архітектурою.

3.5.2 Реалізація модуля детектування

Компоненти модуля детектування викладені у файлах encoder.py (кодувальник), detection_module.py (головний файл), corr_block.py (кореляційний блок) та decoder.py (декодер).

detection_module.py

```python
import torch
import torch.nn as nn

from src.corr_block_ import CorrelationBlock
from src.encoder import Network
from src.decoder import Decoder

class BasicConv2d(nn.Module):
    def __init__(self, in_planes, out_planes, kernel_size, stride=1, padding=0, dilation=1):
        super(BasicConv2d, self).__init__()
        self.conv = nn.Conv2d(in_planes, out_planes,
                              kernel_size=kernel_size, stride=stride,
                              padding=padding, dilation=dilation, bias=False)
        self.bn = nn.BatchNorm2d(out_planes)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        return x

class CorrelationPyramid(nn.Module):
    def __init__(self, in_planes, pyramid_type='conv'):
        super(CorrelationPyramid, self).__init__()
        self.pyramid_type = pyramid_type
        self.layer1 = CorrelationBlock(in_planes * 2, None, False, scale=2)
        self.layer2 = CorrelationBlock(in_planes * 2, None, False, scale=4)
        self.layer3 = CorrelationBlock(in_planes * 2, None, False, scale=8)

        self.conv1 = BasicConv2d(in_planes*2, in_planes, 3, padding=1)
        self.conv2 = BasicConv2d(in_planes*2, in_planes, 3, padding=1)
        self.conv3 = BasicConv2d(in_planes*2, in_planes, 3, padding=1)

    def forward(self, fea1, fea2):
        #pdb.set_trace()
        out1 = self.conv1(self.layer1(torch.cat([fea1[0], fea2[0]], dim=1)))
        out2 = self.conv2(self.layer2(torch.cat([fea1[1], fea2[1]], dim=1)))
        out3 = self.conv3(self.layer3(torch.cat([fea1[2], fea2[2]], dim=1)))
        return out1, out2, out3
```

Рисунок 3.7 – Файл detection_module.py з модулем детектування (повний код наведений у Додатку Б)

У файлі detection_module.py містяться класи BasicConv2d, CorrelationPyramid VideoModel. Клас BasicConv2d визначає двовимірну згортку для даного модуля. У класі CorrelationPyramid описується кореляційна піраміда. Вона має три шари з різним масштабуванням, визначених змінними layer.

```
83      def forward(self, x):
84          image1, image2, image3 = x[0],x[1],x[2]
85          f_map1=self.backbone.feat_net(image1)
86          f_map2=self.backbone.feat_net(image2)
87          f_map3=self.backbone.feat_net(image3)
88
89          corr_vol12 = self.pyramid(f_map1, f_map2)
90          corr_vol13 = self.pyramid(f_map1, f_map3)
91
92          out12 = self.decoder(corr_vol12)
93          out13 = self.decoder(corr_vol13)
94
95          concat = torch.cat([out12[-1], out13[-1]], dim=1)
96          out = self.fusion_conv(concat)
97
98          return out12, out13, out
```

Рисунок 3.8 – Метод *forward()* класу VideoModel

Клас VideoModel описує всю структуру модуля детектування з усіма його компонентами: кодувальником, кореляційною пірамідою та декодером.

У PyTorch метод *forward()* класів nn.Module відповідає за обчислення, які здійснюються при кожному виклику модуля. Саме тому метод *forward()* наглядно показує роботу модуля детектування (рис. 3.7). В якості входу він приймає три послідовні кадри (змінні image). Далі, виконується будування карт ознак цих кадрів (змінні f_map) за допомогою кодувальника, екземпляр якого є атрибутом класу VideoModel під назвою backbone (у нейронних мережах, реалізованих у PyTorch, backbone вказує на базову модель архітектури). Після цього розраховується кореляційний об’єм пар кадрів: першого та другого (змінна corr_vol12), першого та третього(змінна corr_vol13). Наступним кроком за допомогою декодера на основі

ознак з кореляційних об'ємів будуються покращені карти ознак для пар кадрів (змінні out12 та out13) та загальна карта ознак out.

3.5.3 Реалізація модуля узгодженості

Модуль узгодженості реалізований у файлі coherence_module.py. Допоміжні файли, що необхідні для підмодуля seq2seq, розташовані в однойменній директорії.

coherence_module.py | detection_module.py

```python
import torch
import torch.nn as nn

from src.encoder import Network
from src.seq2seq.PNS_Module import NS_Block
from src.decoder import Decoder

class BasicConv2d(nn.Module):
    def __init__(self, in_planes, out_planes, kernel_size, stride=1, padding=0, dilation=1):
        super(BasicConv2d, self).__init__()
        self.conv = nn.Conv2d(in_planes, out_planes,
                              kernel_size=kernel_size, stride=stride,
                              padding=padding, dilation=dilation, bias=False)
        self.bn = nn.BatchNorm2d(out_planes)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        return x

class Seq2Seq(nn.Module):
    def __init__(self, in_planes, pyramid_type='conv'):
        super(Seq2Seq, self).__init__()
        self.pyramid_type = pyramid_type

        self.NSB_1 = NS_Block(in_planes)
        self.NSB_2 = NS_Block(in_planes)

    def forward(self, fea, origin_shape):
        fea1 = fea[1].view([*origin_shape[:2], *fea[1].shape[1:]])
        high_feature_1 = self.NSB_1(fea1) + fea1
        high_feature_2 = self.NSB_2(high_feature_1) + high_feature_1
        out2 = fea1 + high_feature_2
        out2 = out2.view(-1, *out2.shape[2:])
        return fea[0], out2, fea[2]

class VideoModel(nn.Module):
    def __init__(self, args):
        super(VideoModel, self).__init__()
```

Рисунок 3.9 – Файл detection_module.py з модулем узгодженості (повний код наведений у Додатку Б)

Файл модуля узгодженості coherence_module.py має схожу структуру с файлом модуля детектування detection_module.py, адже архітектури цих модулів дуже схожі.

Програмна реалізація модуль узгодженості складається з трьох класів: BasicConv2d, Seq2Seq та VideoModel. Клас BasicConv2d визначає двовимірну згортку для даного модуля і є ідентичним однойменному класу в файлі з модулем детектування. У класі Seq2Seq реалізовано підмодуль seq2seq. Він складається з двох блоків нормалізованої самоуваги (клас NS_Block), імпортованих з файлу PNS_Module з директорії Seq2Seq. Як видно з метода *forward()* класу Seq2Seq, цей компонент в якості входу бере карту ознак та вилучає ознаки вищого рівня (змінні high_feature_1 та high_feature_2) за допомогою використання блоків нормалізованої самоуваги.

```
76      def forward(self, x):
77          if x.shape[2] == 1:
78              x_in = torch.cat([x, x, x], 2)
79          else:
80              x_in = x
81
82          origin_shape = x_in.shape
83          x_in = x_in.view(-1, * origin_shape[2:])
84
85          if x.shape[2] == 4:
86              x_in = self.first_conv(x_in)
87
88          f_map=self.backbone.feat_net(x_in)
89          corr_vol = self.s2s(f_map, origin_shape)
90          out = self.decoder(corr_vol)
91          return out
```

Рисунок 3.10 – Метод *forward()* класу VideoModel

Клас VideoModel містить безпосередню реалізацію модуля узгодженості. В якості базової моделі (атрибут backbone) використовується кодувальник. Як видно з методу *forward()* класу VideoModel спочатку для вилучення ознак застосовується

кодувальник. Далі, йде визначення ознак вищого рівня за допомогою підмодуля seq2seq (атрибут s2s). Результат попередньої операції оброблюється за допомогою декодера, який вже дає на виході остаточну маску об’єкта.

3.5.4 Перевірка роботи мережі на тестовому датасеті MoCa-Mask

Здійснимо перевірку роботи нейронної мережі для детектування малопомітних рухомих об’єктів на тестовому датасеті MoCa-Mask, що знаходиться у директорії dataset та містить 16 відеофрагментів, розбитих по кадрам (рис. 3.10).

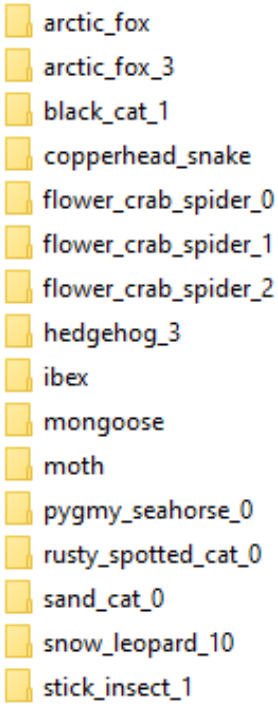


Рисунок 3.11 – Відеофрагменти тестового датасета, розбиті по кадрам

Скрипт для тестування повної натренованої мережі знаходиться у кореневій директорії та називається test_network_full.py. Файл містить клас test_dataset, в якому регулюється робота з датасетом, допоміжні функції test_dataloader та count_parameters_in_MB, і головну функцію.

Головна функція містить команди парсингу аргументів запуску, завантаження даних, налаштування та запуску нейронної мережі відповідно до отриманих аргументів.

test_network_full.py

```python
import torch
import torch.nn.functional as F
import torch.utils.data as data
import torchvision.transforms as transforms
import numpy as np
import os
from PIL import Image
from src import VideoModel_coherence as Network
from path_to_dataset import Path
from glob import glob
import os.path as osp
import imageio
import pdb

def count_parameters_in_MB(model):
    return np.sum(np.prod(v.size()) for name, v in model.named_parameters() if "aux" not in name)/1e6

class test_dataset:
    def __init__(self, dataset='MoCA', split='MoCA-Video-Test',
                 input_length=10, fsampling_rate=1):
        self.input_length = input_length
        self.fsampling_rate = fsampling_rate
        self.image_list = []
        self.extra_info = []

        if dataset == 'MoCA':
            root = Path.db_root_dir('MoCA')
            img_format = '*.jpg'

        data_root = osp.join(root, split)

        for scene in os.listdir(osp.join(data_root)):
            images  = sorted(glob(osp.join(data_root, scene, 'Pred', '*.png')))

            clip_size = self.input_length
            skip_size = self.input_length
            out = False

            video_len = len(images)
            for i in range(1, video_len, skip_size):
```

Рисунок 3.12 – Файл test_network_full.py (повний код наведено в Додатку Б)

Для полегшеного запуску Python-скрипту для тестування нейронної мережі для детектування малопомітних рухомих об’єктів test_network_full.py разом з усіма необхідними аргументами запуску створено Shell-скрипт test_network_full.sh.

test_network_full.sh

```
CUDA_VISIBLE_DEVICES=0 python test_network_full.py --testsplit 'MoCA-Video-Test' \
--pth_path './snapshot/Net_epoch_MoCA_full.pth' --fsampling_rate 1 --input_length 2
```

Рисунок 3.13 – Shell-скрипт запуску тестування нейронної мережі test_network_full.sh

В результаті виконання скрипту для тестування нейронної мережі в директоріях за шляхом *./res/MoCA/network_full/[назва відеофрагменту]/Pred/* з’являються побудовані бінарні маски детектованих малопомітних рухомих об’єктів.

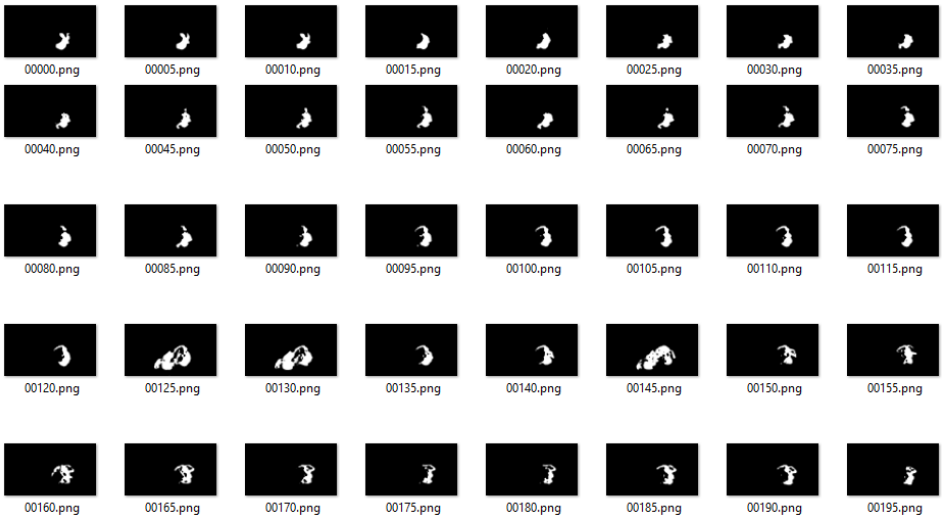


Рисунок 3.14 – Приклад побудованих масок об’єкта: відеофрагмент під назвою flower_crab_spider_1

3.6 Оцінка ефективності роботи нейронної мережі

Оцінка ефективності роботи нейронної мережі для детектування малопомітних рухомих об’єктів відбувається за допомогою розрахунку кількісних показників з бенчмарку MoCA-Mask. Ці розрахунки здійснюються за допомогою файлів-скриптів середовища MATLAB. У проєкті з нейронною мережею вони знаходяться у директорії eval.

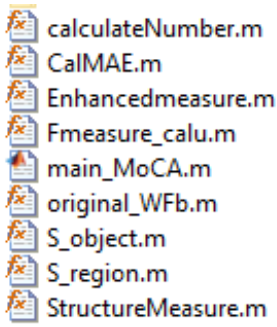


Рисунок 3.15 – Зміст директорії src проєкту з нейронною мережою

main_MoCA.m

```
clear; close; clc;
% set the path of sal/gt/results
salDir = '../res/MoCA/'; %MoCA
Models = {'network_full'};
gtDir = '../dataset/MoCA-Mask/';
Datasets = {'TestDataset_per_sq'}; %ValDataset_per_sq TestDataset_per_sq
Results_Save_Path = './Result/MoCA-Mask_eva/';


Thresholds = 1:-1/255:0;

for  m = 1:length(Models)

    modelName = Models{m}

    resVideoPath = '../res/MoCA/network_full/';  % video saliency map for evaluated models

    videoFiles = dir(gtDir);

    videoNUM = length(videoFiles);

    [video_Sm, video_wFm, video_mae] = deal(zeros(1,videoNUM));
    [video_column_E, video_column_Sen, video_column_Spe, video_column_Dic, video_column_IoU] = deal(zeros(videoNUM,256))

    for videonum = 1:length(Datasets)

        videofolder = Datasets{videonum}

        filePath = [Results_Save_Path modelName '/'];   % results save path

        if ~exist(filePath, 'dir')
            mkdir(filePath);
        end

        fileID = fopen([filePath modelName '_' videofolder '_result.txt'], 'w');


        seqPath = [gtDir videofolder '/'];  % gt sequence Path
        seqFiles = dir(seqPath);

        seqNUM = length(seqFiles)-2;

        [seq_Sm, seq_wFm, seq_mae] = deal(zeros(1,seqNUM));
```

Рисунок 3.16 – Скрипт середовища MATLAB mainMoCA.m (повний код наведено в Додатку Б)

Головним файлом є скрипт mainMoCA.m. За його допомогою виконується порівняння отриманих в результаті тестування нейронної мережі бінарних масок об’єктів з цільовими (англ. ground-truth, GT) масками. Перевірюється наскільки відповідає отриманий результат «правильному», бажаному.

Для кожного відеофрагменту вимірюється S-показник S_α, зважений F-показник F_β^w, показник покращеного вирівнювання E_ϕ (середнє та максимальне значення) та середня абсолютна похибка M, коефіцієнт Дайса Dic (середнє та максимальне значення) та відношення перетину до об'єднання IoU (середнє та максимальне значення).

Також виконується розрахунки середніх та максимальних значень таких показників як чутливість Sen та специфічність Spe. В контексті класифікації пікселів, чутливість Sen (також відома як відгук або частота істинних позитивних результатів) оцінює можливості виявлення істинних позитивних результатів, а специфічність Spe (також відома як частота істинних негативних результатів) оцінює можливості правильної ідентифікації класів істинних негативних результатів (наприклад, фон зображення).

Останнім кроком виконується розрахунок середніх арифметичних значень вище наведених показників по всім тестовим відеофрагментам.

Результат зроблених у MATLAB розрахунків з файлу mainMoCA.m генерується в текстовий файл з назвою *network_full_TestDataset_per_sq_result.txt* за шляхом *./eval/Result/MoCA-Mask_eva/network_full*. Цей файл містить таблицю, рядками якої є тестові відеофрагменти, а стовпцями – показники з бенчмарку MoCA-Mask.

```
(arctic_fox) 0.622 0.765 0.858 0.024 0.701 0.608 0.867 0.717 0.627 0.887 1.000 0.988 0.993
(arctic_fox_3) 0.388 0.725 0.654 0.016 0.441 0.356 0.687 0.455 0.371 0.679 1.000 0.818 0.822
(black_cat_1) 0.368 0.677 0.782 0.023 0.444 0.327 0.826 0.448 0.329 0.588 1.000 0.899 0.914
(copperhead_snake) 0.460 0.697 0.851 0.018 0.556 0.390 0.963 0.562 0.402 0.591 1.000 0.999 0.995
(flower_crab_spider_0) 0.098 0.530 0.437 0.076 0.160 0.091 0.453 0.193 0.111 0.542 1.000 0.892 0.903
(flower_crab_spider_1) 0.632 0.774 0.906 0.023 0.723 0.572 0.953 0.739 0.597 0.907 1.000 0.991 0.997
(flower_crab_spider_2) 0.623 0.783 0.905 0.027 0.701 0.559 0.946 0.704 0.562 0.843 1.000 0.987 0.998
(hedgehog_3) 0.260 0.557 0.677 0.060 0.330 0.254 0.739 0.335 0.257 0.444 1.000 0.588 0.633
(ibex) 0.128 0.628 0.488 0.017 0.187 0.116 0.546 0.208 0.133 0.652 1.000 0.780 0.786
(mongoose) 0.326 0.628 0.690 0.019 0.414 0.277 0.755 0.447 0.304 0.807 1.000 0.869 0.875
(moth) 0.390 0.634 0.754 0.013 0.425 0.313 0.791 0.427 0.318 0.518 1.000 0.695 0.697
(pygmy_seahorse_0) 0.205 0.581 0.810 0.014 0.231 0.171 0.919 0.229 0.170 0.262 1.000 0.396 0.396
(rusty_spotted_cat_0) 0.235 0.566 0.878 0.023 0.304 0.226 0.958 0.331 0.250 0.354 1.000 0.504 0.504
(sand_cat_0) 0.194 0.610 0.879 0.023 0.211 0.178 0.893 0.221 0.190 0.246 1.000 0.375 0.377
(snow_leopard_10) 0.013 0.500 0.590 0.011 0.019 0.014 0.674 0.019 0.013 0.036 1.000 0.103 0.132
(stick_insect_1) 0.021 0.492 0.927 0.022 0.030 0.019 0.914 0.047 0.031 0.039 1.000 0.389 0.486
(Dataset) wFm Sm meanEm MAE meanDic meanIoU maxEm maxDice maxIoU meanSen maxSen meanSpe maxSpe
 (TestDataset_per_sq) 0.310 0.634 0.755 0.026 0.367 0.280 0.795 0.394 0.301 0.525 1.000 0.695 0.720
```

Рисунок 3.17 – Файл з результатами розрахунків ефективності роботи нейронної мережі network_full_TestDataset_per_sq_result.txt

Отримані результати (рис. 3.11) слід порівняти з іншими моделями нейронних мереж для детектування малопомітних рухомих об’єктів за такими ж показниками. Результати порівняння внесені у Таблицю 3.1 (символи біля показників означають: ↑ – чим вище, тим краще; ↓ – чим нижче, тим краще). Дані тестування аналогічних моделей взяті з [9].

Таблиця 3.1 – Порівняння нашої нейронної мережі з аналогічними моделями нейронних мереж.

Модель	$S_\alpha \uparrow$	$F_\beta^w \uparrow$	$E_\phi \uparrow$	$M \downarrow$	$mDic \uparrow$	$mIoU \uparrow$
PNS-Net	0.544	0.097	0.510	0.033	0.121	0.101
RCRNet	0.555	0.138	0.527	0.033	0.171	0.116
MG	0.530	0.168	0.561	0.067	0.181	0.127
SLT-Net	0.631	0.311	0.759	0.027	0.360	0.272
Наша мережа	0.634	0.310	0.755	0.026	0.367	0.280

Як видно з Таблиці 3.1, побудована нейронна мережа перевершує аналогічні моделі нейронних мереж за такими показниками, як: S_α , $M, mDic$, $mIoU$; проте програє за показниками F_β^w та E_ϕ.

Висновок до розділу 3

Для програмної реалізації нейронної мережі для детектування малопомітних рухомих об'єктів було застосовано таке програмне забезпечення: мова програмування Python, фреймворк машинного навчання PyTorch, інтегроване середовище розробки Pycharm. Розрахунки ефективності роботи нейронної мережі були виконані у програмному забезпеченні MATLAB.

Директорія з програмною реалізацією нейронної мережі має розгалуджену структуру. Головною є директорія src, в якій знаходиться .py файли з реалізацією всіх модулів, підмодулів та компонентів нейронної мережі. Модуль детектування міститься у файлі detection_module.py, а модель узгодженості – у coherence_module.py.

Тренування та тестування роботи нейронної мережі виконується за допомогою скриптів train_network та test_network. Для тренування та тестування нейронної мережі використовується відеофрагменти з датасету MoCA-Mask.

Результатом тестування роботи на тестовому датасеті MoCA-Mask нейронної мережі є колекція побудованих масок для всіх кадрів відеофрагментів датасету. Для

перевірки ефективності роботи нейронної мережі ці побудовані бінарні маски об’єктів порівнюються з цільовими за допомогою розрахунку показників бенчмарку MoCA-Mask у середовищі MATLAB.

В результаті порівняння ефективності нашої нейронної мережі для детектування малопомітних рухомих об’єктів з аналогічними моделями нейронних мереж було визначено, що побудована нейронна мережа перевершує аналогічні моделі нейронних мереж за такими показниками, як: S-показник S_α, середня абсолютна похибка M, середнє значення коефіцієнту Дайса $mDic$, середнє значення відношення перетину до об'єднання IoU; одночасно з цим вона програє за зваженим F-показником F_β^w та показником покращеного вирівнювання E_ϕ.

ВИСНОВКИ

Метою даної бакалаврської кваліфікаційної роботи була реалізація нейронної мережі для детектування малопомітних рухомих об’єктів шляхом поєднання таких архітектур нейронних мереж, як згорткова нейронна мережа та трансформер.

Першим кроком було проведено аналіз об’єкту та предмету дослідження, а саме – технологій комп’ютерного зору та нейронної мережі детектування малопомітних рухомих об’єктів. Було проаналізовано наукову літературу на дану тему та визначено найкращі технологічні рішення у цій сфері. Здобута при цьому інформацію була використана при проєктуванні та реалізації нашої нейронної мережі. Хід та результати даного етапу виконання бакалаврської кваліфікаційної роботи було викладено у першому розділі.

Наступним кроком було виконано проєктування нейронної мережі. Було спроєктовано архітектуру нейронної мережі, що складається з двох модулів: модуля детектування та модуля узгодженості, які поєднують елементи згорткових нейронних мереж та трансформеру. Також було визначено математичний апарат, що використовувався при розробці нейронної мережі, а саме: застосування математичної операції згортки та мінімізації функції витрат нейронної мережі за допомогою застосування ітеративного методу стохастичного градієнтного спуску. Було вибрано апаратне забезпечення для апаратно-програмного комплексу з нашою нейронною мережею. Дані кроки були детально описані у другому розділі бакалаврської кваліфікаційної роботи.

У третьому розділі був описаний останній етап побудови нейронної мережі для детектування малопомітних рухомих об’єктів – її програмна реалізація. Для цього спочатку було виконано аналіз відповідного програмного забезпечення. В результаті даного аналізу було прийнято рішення використовувати мову програмування Python та фреймворк PyTorch. Саме за допомогою даних програмних засобів було реалізовано нейронну мережу для детектування малопомітних рухомих об’єктів.

У кінці було проведено оцінку ефективності побудованої нейронної мережі на тестовому датасеті MoCA-Mask та порівняно з аналогічними моделями нейронних мереж. Проведені розрахунки показали, що реалізована нейронна мережа для детектування малорухомих непомітних об’єктів знаходиться на рівні з аналогами, а по деяким показникам перевершує їх.

Побудована нейронна мережа для детектування малопомітних рухомих об’єктів зберігає потенціал для подальшого удосконалення. Основним способом підвищення ефективності роботи нейронної мережі є використання для тренування датасету зі значно більшою кількістю відеофрагментів.

Практичні та теоретичні напрацювання, що були зроблені під час виконання даної бакалаврської роботи, можуть бути застосовані у багатьох сферах, де потрібне знаходження малопомітного рухомого об’єкта у складному середовищі: для пошуково-рятувальних задач, для систем розумного відеоспостереження, для знаходження та захисту рідкісних видів тварин, для візуальних задач дослідження океану та космосу, для систем медичного візуального аналізу та ін.

Дана бакалаврська кваліфікаційна робота була апробована на науковій конференції «Ольвійський форум – 2023: стратегії країн Причорноморського регіону в геополітичному просторі».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Le T.N., Nguyen T.V., Nie Z.L., Tran M.T., Sugimoto A. Anabranch Network for Camouflaged Object Segmentation. 2019. *Journal of Computer Vision and Image Understanding*. Vol. 184, P. 45-56. DOI: https://doi.org/10.1016/j.cviu.2019.04.006.

2. Fan D.-P., Ji G.-P., Sun G., Cheng M.-M., Shen J., Shao L. Camouflaged Object Detection. 2020. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. P. 2774-2784. DOI: https://doi.org/10.1109/CVPR42600.2020.00285.

3. Mei H., Ji G.-P., Wei Z., Yang X., Wei X., Fan D.-P. Camouflaged Object Segmentation with Distraction Mining. 2021. *IEEE/CVF Conference on Computer Vision and Pattern Recognition.* P. 8768-8777. DOI: https://doi.org/10.1109/CVPR46437.2021.00866.

4. Zhai Q., Li X., Yang F., Chen C., Cheng H., Fan D.-P. Mutual Graph Learning for Camouflaged Object Detection. 2021. *IEEE/CVF Conference on Computer Vision and Pattern Recognition.* P. 12992-13002. DOI: https://doi.org/10.1109/CVPR46437.2021.01280.

5. Fan D.-P., Ji G., Cheng M., Shao L. Concealed Object Detection. 2021. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 44. P. 6024-6042. DOI: https://doi.org/10.1109/TPAMI.2021.3085766.

6. Bideau P., Learned-Miller E. It's Moving! A Probabilistic Model for Causal Motion Segmentation in Moving Camera Videos. 2016. *ECCV 2016. Lecture Notes in Computer Science()*. Vol. 9912. Springer, Cham. P. 433–449. DOI: https://doi.org/10.1007/978-3-319-46484-8_26.

7. Bideau P., Menon R.R., & Learned-Miller E.G. MoA-Net: Self-supervised Motion Segmentation. *ECCV 2018. Lecture Notes in Computer Science().* Vol. 11134. P. 715–730. DOI: https://doi.org/10.1007/978-3-030-11024-6_55.

8. Lamdouar H., Yang C., Xie W., Zisserman A. Betrayed by Motion: Camouflaged Object Discovery via Motion Segmentation. 2021. *ACCV 2020. Lecture*

Notes in Computer Science(). Vol. 12623. DOI: https://doi.org/10.1007/978-3-030-69532-3_30.

9. Cheng X., Xiong H., Fan D.-P., Zhong Y., Harandi M.T., Drummond T., Ge Z. Implicit Motion Handling for Video Camouflaged Object Detection. 2022. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* P. 13854-13863. DOI: https://doi.org/10.1109/CVPR52688.2022.01349.

10. Animal Camouflage Analysis: CHAMELEON Database. Politechnika Śląska. URL: https://www.polsl.pl/rau6/chameleon-database-animal-camouflage-analysis (Last accessed: 12.05.2023).

11. Wang W., Xie E., Li X. et al. PVT v2: Improved baselines with Pyramid Vision Transformer. 2022. *Comp. Visual Media*. Vol. 8. P. 415–424. DOI: https://doi.org/10.1007/s41095-022-0274-8.

12. Ji G.-P. et al. Progressively Normalized Self-Attention Network for Video Polyp Segmentation. 2021. MICCAI 2021. *Lecture Notes in Computer Science()*. Vol. 12901. P. 142–152. DOI: https://doi.org/10.1007/978-3-030-87193-2_14.

13. Wei J., Wang S., Huang Q. F3net: Fusion, feedback and focus for salient object detection. 2020. AAAI Conference on Artificial Intelligence 2020. P. 12321–12328. DOI: https://doi.org/10.1609/aaai.v34i07.6916.

14. Zhao J., Liu J.-J., Fan D.-P., Cao Y., Yang J., Cheng M.-M. EGNet: Edge Guidance Network for Salient Object Detection. 2019. IEEE/CVF International Conference on Computer Vision (ICCV) 2019. P. 8778-8787. DOI: https://doi.org/10.1109/ICCV.2019.00887.

15. Ma J. Segmentation Loss Odyssey. 2020. *ArXiv, abs/2005.13449*. DOI: https://doi.org/10.48550/arXiv.2005.13449.

16. Rahman M. A. Wang, Y. Optimizing intersection-over-union in deep neural networks for image segmentation. 2016. *Advances in Visual Computing. ISVC 2016. Lecture Notes in Computer Science()*. Vol. 10072. P. 234–244. Springer. DOI: https://doi.org/10.1007/978-3-319-50835-1_22

17. Fan D.-P., Gong C., Cao Y., et al. Enhanced-alignment measure for binary foreground map evaluation. 2018. *Proceedings of International Joint Conference on Artificial Intelligence 2018.* P. 698–704. DOI: https://doi.org/10.24963/ijcai.2018/97.

18. Fan D.-P., Ji G.-P., Qin X., Cheng M.-M. Cognitive Vision Inspired Object Segmentation Metric and Loss Function. 2021. *SCIENTIA SINICA Informationis*. Vol. 51. DOI: https://doi.org/10.1360/SSI-2020-0370.

19. Fan D.-P., Cheng M.-M., Liu Y., Li T., Borji A. Structure-measure: A New Way to Evaluate Foreground Maps. 2017. *Proceedings of the IEEE International Conference on Computer Vision 2017*. P. 4548–4557. DOI: https://doi.org/10.48550/arXiv.1708.00786.

20. Margolin R., Zelnik-Manor L., Tal A. How to evaluate foreground maps? 2014. *2014 IEEE Conference on Computer Vision and Pattern Recognition*. P. 248–255. DOI: https://doi.org/10.1109/CVPR.2014.39.

21. Perazzi F., Krähenbühl P., Pritch Y., Hornung A. Saliency filters: Contrast based filtering for salient region detection. 2012. 2012 IEEE Conference on Computer Vision and Pattern Recognition. P. 733-740. DOI: https://doi.org/10.1109/CVPR.2012.6247743.

22. Calin O. Deep Learning Architectures: A Mathematical Approach. Springer, 2020. 760 p.

23. Difference Between the Cost, Loss, and the Objective Function. *Baeldung on Computer Science*. URL: https://www.baeldung.com/cs/cost-vs-loss-vs-objective-function (Last accessed: 20.05.2023).

24. Raspberry Pi 4 Model B specifications. *Raspberry Pi*. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/ (Last accessed: 20.05.2023).

25. Jetson Nano Developer Kit. *NVIDIA Developer*. URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (Last accessed: 20.05.2023).

26. Raspberry Pi 4 Model B 4GB. *Arduino в Україні*. URL: https://arduino.ua/prod3308-raspberry-pi-4-4gb (дата звернення: 20.05.2023).

27. Плата міні-комп'ютера NVIDIA Jetson Nano Developer Kit (V3). *Arduino в Україні*. URL: https://arduino.ua/prod3564-nvidia-jetson-nano-developer-kit (дата звернення: 20.05.2023).

28. IMX477-160 12.3MP Camera, 160° FOV, Applicable For Jetson Nano / Compute Module. *Waveshare*. URL: https://www.waveshare.com/imx477-160-12.3mp-camera.htm (Last accessed: 20.05.2023).

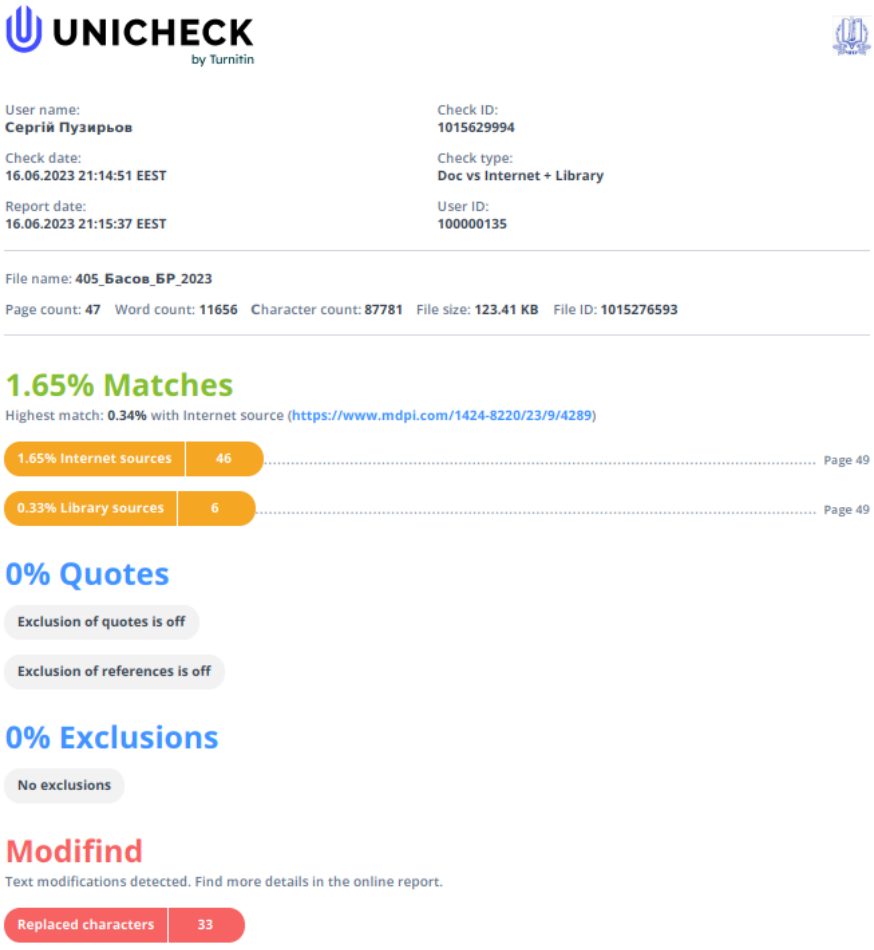
29. Камера IMX477-160 12.3Мп з ширококутним 160° об'єктивом для Jetson Nano/Compute Module. *Arduino в Україні*. URL: https://arduino.ua/prod4863-imx477-160-12-3mp-camera-160-fov-applicable-for-jetson-nano-compute-module (дата звернення: 20.05.2023).

ДОДАТОК А

ДОВІДКА

про перевірку на унікальність пояснювальної записки
кваліфікаційної бакалаврської роботи
на тему: «Побудова нейронної мережі для детектування малопомітних рухомих об’єктів»
студента спеціальності 123 «Комп’ютерна інженерія»,
групи 405
Басов Даніїл Євгенійович
прізвище, ім’я, по-батькові

Перевірку тексту здійснено сервісом: онлайн-сервіс Unicheck.
Результат перевірки тексту роботи: схожість складає 1,65 %.


UNICHECK
by Turnitin

User name: Сергій Пузирьов
Check ID: 1015629994
Check date: 16.06.2023 21:14:51 EEST
Check type: Doc vs Internet + Library
Report date: 16.06.2023 21:15:37 EEST
User ID: 100000135

File name: 405_Басов_БР_2023
Page count: 47 Word count: 11656 Character count: 87781 File size: 123.41 KB File ID: 1015276593

1.65% Matches
Highest match: 0.34% with Internet source (https://www.mdpi.com/1424-8220/23/9/4289)
1.65% Internet sources 46 Page 49
0.33% Library sources 6 Page 49

0% Quotes
Exclusion of quotes is off
Exclusion of references is off

0% Exclusions
No exclusions

Modifind
Text modifications detected. Find more details in the online report.
Replaced characters 33

Студент
Басов
Д. Є. Басов
підпис ініціали, прізвище

Керівник
канд. фіз.-мат. наук, доцент
С. В. Пузирьов
підпис ініціали, прізвище

Дата: «___»_____2023 р.

ДОДАТОК Б

Код нейронної мережі для детектування малопомітних рухомих об’єктів

Лістинг 1. detection_module.py

```
import torch
import torch.nn as nn

from src.corr_block_ import CorrelationBlock
from src.encoder import Network
from src.decoder import Decoder

class BasicConv2d(nn.Module):
    def __init__(self, in_planes, out_planes, kernel_size, stride=1, padding=0,
dilation=1):
        super(BasicConv2d, self).__init__()
        self.conv = nn.Conv2d(in_planes, out_planes,
                              kernel_size=kernel_size, stride=stride,
                              padding=padding, dilation=dilation, bias=False)
        self.bn = nn.BatchNorm2d(out_planes)
        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        return x

class CorrelationPyramid(nn.Module):
    def __init__(self, in_planes, pyramid_type='conv'):
        super(CorrelationPyramid, self).__init__()
        self.pyramid_type = pyramid_type
        self.layer1 = CorrelationBlock(in_planes * 2, None, False, scale=2)
        self.layer2 = CorrelationBlock(in_planes * 2, None, False, scale=4)
        self.layer3 = CorrelationBlock(in_planes * 2, None, False, scale=8)

        self.conv1 = BasicConv2d(in_planes*2, in_planes, 3, padding=1)
        self.conv2 = BasicConv2d(in_planes*2, in_planes, 3, padding=1)
        self.conv3 = BasicConv2d(in_planes*2, in_planes, 3, padding=1)

    def forward(self, fea1, fea2):
        #pdb.set_trace()
        out1 = self.conv1(self.layer1(torch.cat([fea1[0], fea2[0]], dim=1)))
        out2 = self.conv2(self.layer2(torch.cat([fea1[1], fea2[1]], dim=1)))
        out3 = self.conv3(self.layer3(torch.cat([fea1[2], fea2[2]], dim=1)))
        return out1, out2, out3

class VideoModel(nn.Module):
    def __init__(self, args, **kwargs):
        super(VideoModel, self).__init__()

        self.args = args
        self.extra_channels = 0
        print("Select mask mode: concat, num_mask={}".format(self.extra_channels))

        self.backbone = Network(pvtv2_pretrained=False, imgsize=self.args.trainsize)
        if self.args.pretrained_cod10k is not None:
            self.load_backbone(self.args.pretrained_cod10k )
```

```
            self.pyramid = CorrelationPyramid(in_planes=32, pyramid_type='conv')

            self.decoder = Decoder

            self.fusion_conv = nn.Sequential(nn.Conv2d(2, 32, 3, 1, 1),
                                             nn.Conv2d(32, 32, 3, 1, 1),
                                             nn.Conv2d(32, 1, 3, 1, 1),
                )

            self.freeze_bn()

        def load_backbone(self, pretrained):
            pretrained_dict = torch.load(pretrained)
            model_dict = self.state_dict()
            print("Load pretrained cod10k parameters from {}".format(pretrained))
            for k, v in pretrained_dict.items():
                if (k in model_dict):
                    print("load:%s"%k)
                else:
                    print("jump over:%s"%k)

            pretrained_dict = {k: v for k, v in pretrained_dict.items() if (k in
model_dict)}
            model_dict.update(pretrained_dict)
            self.load_state_dict(model_dict)

        def freeze_bn(self):
            for m in self.backbone.named_modules():
                if isinstance(m[1], nn.BatchNorm2d):
                    m[1].eval()

        def forward(self, x):
            image1, image2, image3 = x[0],x[1],x[2]
            f_map1=self.backbone.feat_net(image1)
            f_map2=self.backbone.feat_net(image2)
            f_map3=self.backbone.feat_net(image3)

            corr_vol12 = self.pyramid(f_map1, f_map2)
            corr_vol13 = self.pyramid(f_map1, f_map3)

            out12 = self.decoder(corr_vol12)
            out13 = self.decoder(corr_vol13)

            concat = torch.cat([out12[-1], out13[-1]], dim=1)
            out = self.fusion_conv(concat)

            return out12, out13, out
```

Лістинг 2. coherence_module.py

```
import torch
import torch.nn as nn

from src.encoder import Network
from src.seq2seq.PNS_Module import NS_Block
from src.decoder import Decoder

class BasicConv2d(nn.Module):
```

```
        def __init__(self, in_planes, out_planes, kernel_size, stride=1, padding=0,
dilation=1):
            super(BasicConv2d, self).__init__()
            self.conv = nn.Conv2d(in_planes, out_planes,
                                  kernel_size=kernel_size, stride=stride,
                                  padding=padding, dilation=dilation, bias=False)
            self.bn = nn.BatchNorm2d(out_planes)
            self.relu = nn.ReLU(inplace=True)

        def forward(self, x):
            x = self.conv(x)
            x = self.bn(x)
            return x

    class Seq2Seq(nn.Module):
        def __init__(self, in_planes, pyramid_type='conv'):
            super(Seq2Seq, self).__init__()
            self.pyramid_type = pyramid_type

            self.NSB_1 = NS_Block(in_planes)
            self.NSB_2 = NS_Block(in_planes)

        def forward(self, fea, origin_shape):
            fea1 = fea[1].view([*origin_shape[:2], *fea[1].shape[1:]])
            high_feature_1 = self.NSB_1(fea1) + fea1
            high_feature_2 = self.NSB_2(high_feature_1) + high_feature_1
            out2 = fea1 + high_feature_2
            out2 = out2.view(-1, *out2.shape[2:])
            return fea[0], out2, fea[2]

    class VideoModel(nn.Module):
        def __init__(self, args):
            super(VideoModel, self).__init__()

            self.args = args
            self.extra_channels = 0
            print("Select mask mode: concat, num_mask={}".format(self.extra_channels))

            self.backbone = Network(pvtv2_pretrained=False, imgsize=self.args.trainsize)
            if self.args.detection_pretrained is not None:
                self.load_backbone(self.args.detection_pretrained)

            self.s2s = Seq2Seq(in_planes=32, pyramid_type='conv')
            self.first_conv = nn.Conv2d(4, 3, 1)

            self.decoder = Decoder

            self.freeze_bn()

        def load_backbone(self, pretrained):
            pretrained_dict = torch.load(pretrained)
            model_dict = self.state_dict()
            print("Load pretrained parameters from {}".format(pretrained))
            for k, v in pretrained_dict.items():
                if (k in model_dict):
                    print("load:%s"%k)
                else:
                    print("jump over:%s"%k)
```

```
            pretrained_dict = {k: v for k, v in pretrained_dict.items() if (k in
model_dict)}
            model_dict.update(pretrained_dict)
            self.load_state_dict(model_dict)

        def freeze_bn(self):
            for m in self.backbone.named_modules():
                if isinstance(m[1], nn.BatchNorm2d):
                    m[1].eval()

        def forward(self, x):
            if x.shape[2] == 1:
                x_in = torch.cat([x, x, x], 2)
            else:
                x_in = x

            origin_shape = x_in.shape
            x_in = x_in.view(-1, * origin_shape[2:])

            if x.shape[2] == 4:
                x_in = self.first_conv(x_in)

            f_map=self.backbone.feat_net(x_in)
            corr_vol = self.s2s(f_map, origin_shape)
            out = self.decoder(corr_vol)
            return out
```

Лістинг 3. test_network_full.py

```
import torch
import torch.nn.functional as F
import torch.utils.data as data
import torchvision.transforms as transforms
import numpy as np
import os
from PIL import Image
from src import VideoModel_coherence as Network
from path_to_dataset import Path
from glob import glob
import os.path as osp
import imageio
import pdb

def count_parameters_in_MB(model):
    return np.sum(np.prod(v.size()) for name, v in model.named_parameters() if "aux"
not in name)/1e6

class test_dataset:
    def __init__(self, dataset='MoCA', split='MoCA-Video-Test',
                 input_length=10, fsampling_rate=1):
        self.input_length = input_length
        self.fsampling_rate = fsampling_rate
        self.image_list = []
        self.extra_info = []

        if dataset == 'MoCA':
            root = Path.db_root_dir('MoCA')
            img_format = '*.jpg'
```

```
            data_root = osp.join(root, split)

            for scene in os.listdir(osp.join(data_root)):
                images  = sorted(glob(osp.join(data_root, scene, 'Pred', '*.png')))

                clip_size = self.input_length
                skip_size = self.input_length
                out = False

                video_len = len(images)
                for i in range(1, video_len, skip_size):
                    clip_im = []
                    indices   =   list(range(i,   min(i+clip_size*(self.fsampling_rate),
video_len), self.fsampling_rate))
                    if len(indices) < clip_size:
                        continue
                    for j in indices:
                        clip_im.append(images[j])
                    self.image_list.append(clip_im)

                # add last one
                for   i   in   range(video_len-1,   video_len-self.input_length-2,   -
fsampling_rate):

                    clip_im = []
                    indices   =   list(range(i,   min(i+clip_size*(self.fsampling_rate),
video_len), self.fsampling_rate))
                    if len(indices) < clip_size:
                        continue
                    for j in indices:
                        clip_im.append(images[j])
                    self.image_list.append(clip_im)

                # add first one
                clip_im = []
                indices=list(range(self.input_length, -1, -self.fsampling_rate))
                for j in indices:
                    clip_im.append(images[j])
                self.image_list.append(clip_im)


            if len(self.image_list) == 0:
                raise
            # transforms
            self.transform = transforms.Compose([
                transforms.Resize((256, 448)),
                transforms.ToTensor()])

            self.index = 0
            self.size = len(self.image_list)

        def load_data(self):
            imgs = []
            shts = []
            names= []
            IMG = None
            PRED= None
            LABEL = None
            # forward
```

```
            for i in range(len(self.image_list[self.index])):
            # backward
            # for i in range(len(self.image_list[self.index])-1, -1, -1):
                if 'MoCA-Video-Test' in self.image_list[self.index][i]:
                    rgb_name = self.image_list[self.index][i].replace('Pred','Frame')
                else:
                    rgb_name = self.image_list[self.index][i].replace('Pred','Imgs')
                rgb_name = rgb_name.replace('.png','.jpg')
                imgs += [self.rgb_loader(rgb_name)]
                shts += [self.binary_loader(self.image_list[self.index][i])]
                names+= [self.image_list[self.index][i].split('/')[-1]]

            img_size = imgs[0].size

            for i in range(len(imgs)):
                imgs[i] = self.transform(imgs[i]).unsqueeze(0)
                shts[i] = self.transform(shts[i]).unsqueeze(0)

            scene= self.image_list[self.index][0].split('/')[-3]
            self.index += 1
            self.index = self.index % self.size

            for idx, (img, sht) in enumerate(zip(imgs, shts)):
                if IMG is not None:
                    IMG[idx, :, :, :] = img
                    PRED[idx, :, :, :] = sht
                else:
                    IMG = torch.zeros(len(imgs), *(img.shape))
                    PRED = torch.zeros(len(imgs), *(sht.shape))
                    IMG[idx, :, :, :] = img
                    PRED[idx, :, :, :] = sht
            return IMG, PRED, img_size, names, scene

        def rgb_loader(self, path):
            with open(path, 'rb') as f:
                img = Image.open(f)
                return img.convert('RGB')

        def binary_loader(self, path):
            with open(path, 'rb') as f:
                img = Image.open(f)
                return img.convert('L')
        def __len__(self):
            return self.size

    def test_dataloader(args):
        test_loader = test_dataset(dataset=args.dataset,
                                   split=args.testsplit,
                                   input_length=args.input_length,
                                   fsampling_rate=args.fsampling_rate)
        print('Test with %d image pairs' % len(test_loader))
        return test_loader

    if __name__ == '__main__':
        import argparse

        parser = argparse.ArgumentParser()
        parser.add_argument('--frame_gap', type=int, default=1, help='epoch number')
        parser.add_argument('--input_length', type=int, default=5, help='epoch number')
```

```
        parser.add_argument('--fsampling_rate',    type=int,    default=1,    help='epoch
number')
        parser.add_argument('--batchsize',  type=int,  default=1,  help='training  batch
size')
        parser.add_argument('--dataset', type=str, default='MoCA')
        parser.add_argument('--testsplit', type=str, default='MoCA-Video-Test')
        parser.add_argument('--testsize',  type=int,  default=352,  help='testing  frame
size')
        parser.add_argument('--trainsize',  type=int,  default=352,  help='testing  frame
size')
        parser.add_argument('--pth_path',                                        type=str,
default='./snapshot/COD10K/Net_epoch_best.pth')
        parser.add_argument('--detection_pretrained',      type=str,      default=None,
help='train from detection module')
        opt = parser.parse_args()

        test_loader = test_dataloader(opt)
        name_e = opt.pth_path.split('/')[-1].split('_')[-1]
        save_root    =    './res/{}/longterm_{}_f{}/'.format(opt.dataset,    name_e[:-4],
opt.input_length)
        model = Network(opt)
        model = torch.nn.DataParallel(model).cuda()
        model.load_state_dict(torch.load(opt.pth_path))
        model.cuda()
        model.eval()

        # compute parameters
        print('Total Params = %.2fMB' % count_parameters_in_MB(model))

        for i in range(test_loader.size):
            images, shorts, gt_shape, names, scene = test_loader.load_data()
            save_path=save_root+scene+'/Pred/'

            if not os.path.exists(save_path):
                os.makedirs(save_path)

            inputs = torch.cat([images, shorts], 2)
            preds = model(inputs)

            for res, name in zip(preds[-1][:], names[:]):
                if name[-5] in ['0','5']:
                    # pdb.set_trace()
                    res  =  F.upsample(res.unsqueeze(0),  size=(gt_shape[1],gt_shape[0]),
mode='bilinear', align_corners=False)
                    res = res.sigmoid().data.cpu().numpy().squeeze()
                    res = (res – res.min()) / (res.max() – res.min() + 1e-8)
                    print('> ')
                    # name =names[index].replace('jpg','png')
                    imageio.imwrite(save_path+name, res)
```

Лістинг 4. mainMoCA.m

```
clear; close; clc;
% set the path of sal/gt/results
salDir = '../res/MoCA/'; %MoCA
Models = {'network_full'};
gtDir = '../dataset/MoCA-Mask/';
Datasets = {'TestDataset_per_sq'}; %ValDataset_per_sq TestDataset_per_sq
Results_Save_Path = './Result/MoCA-Mask_eva/';
```

```
      Thresholds = 1:-1/255:0;

      for  m = 1:length(Models)

          modelName = Models{m}

          resVideoPath = '../res/MoCA/network_full/';  % video saliency map for evaluated
models

          videoFiles = dir(gtDir);

          videoNUM = length(videoFiles);

          [video_Sm, video_wFm, video_mae] = deal(zeros(1,videoNUM));
          [video_column_E,    video_column_Sen,    video_column_Spe,    video_column_Dic,
video_column_IoU] = deal(zeros(videoNUM,256));

          for videonum = 1:length(Datasets)

              videofolder = Datasets{videonum}

              filePath = [Results_Save_Path modelName '/'];   % results save path

              if ~exist(filePath, 'dir')
                  mkdir(filePath);
              end

              fileID = fopen([filePath modelName '_' videofolder '_result.txt'], 'w');


              seqPath = [gtDir videofolder '/'];  % gt sequence Path
              seqFiles = dir(seqPath);

              seqNUM = length(seqFiles)-2;

              [seq_Sm, seq_wFm, seq_mae] = deal(zeros(1,seqNUM));
              [seq_column_E,     seq_column_Sen,     seq_column_Spe,     seq_column_Dic,
seq_column_IoU] = deal(zeros(seqNUM,256));

              for seqnum = 1: seqNUM

                  seqfolder = seqFiles(seqnum+2).name;

                  gt_imgPath = [seqPath seqfolder '/GT/'];
                  [fileNUM, gt_imgFiles, fileExt] = calculateNumber(gt_imgPath); %index of
stop frame


                  resPath = [resVideoPath seqfolder '/Pred/']; %'/Pred/'
                  fileNUM = fileNUM-2; %remove last two frame to match the video results

                  [threshold_Fmeasure,     threshold_Emeasure,     threshold_IoU]     =
deal(zeros(fileNUM, length(Thresholds)));
                  [threshold_Sensitivity,   threshold_Specificity,   threshold_Dice]   =
deal(zeros(fileNUM, length(Thresholds)));

                  [Smeasure, wFmeasure, MAE] =deal(zeros(1,fileNUM));
```

```
                tic;
                for i = 1:fileNUM %skip the first and last gt file for some of the light-
flow based method

                    name = char(gt_imgFiles{i});
                    fprintf('[Processing Info] Model: %s, Dataset: %s, SalSeq: %s
(%d/%d), SalMapName: %s (%d/%d)\n',modelName, videofolder, seqfolder, seqnum, seqNUM, name,
i, fileNUM);

                    %load gt
                    gt = imread([gt_imgPath name]);

                    if (ndims(gt)>2)
                        gt = rgb2gray(gt);
                    end

                    if ~islogical(gt)
                        gt = gt(:,:,1) > 128;
                    end

                    %load resMap
                    %resname = strrep(name,'png', 'jpg');
                    resmap  = imread([resPath name]);
                    %check size
                    if size(resmap, 1) ~= size(gt, 1) || size(resmap, 2) ~= size(gt, 2)
                        resmap = imresize(resmap,size(gt));
                        imwrite(resmap,[resPath name]);
                        fprintf('Resizing have been operated!! The resmap size is not
math with gt in the path: %s!!!\n', [resPath name]);
                    end

                    resmap = im2double(resmap(:,:,1));

                    %normalize resmap to [0, 1]
                    resmap = reshape(mapminmax(resmap(:)',0,1),size(resmap));

                    % S-meaure metric published in ICCV'17 (Structure measure: A New Way
to Evaluate the Foreground Map.)
                    Smeasure(i) = StructureMeasure(resmap,logical(gt));

                    % Weighted F-measure metric published in CVPR'14 (How to evaluate the
foreground maps?)
                    wFmeasure(i) = original_WFb(resmap,logical(gt));

                    MAE(i) = mean2(abs(double(logical(gt)) - resmap));
                    [threshold_E,     threshold_F,     threshold_Pr,     threshold_Rec,
threshold_Iou]  = deal(zeros(1,length(Thresholds)));
                    [threshold_Spe, threshold_Dic]  = deal(zeros(1,length(Thresholds)));
                    for t = 1:length(Thresholds)
                        threshold = Thresholds(t);
                        [threshold_Pr(t),      threshold_Rec(t),      threshold_Spe(t),
threshold_Dic(t),             threshold_F(t),             threshold_Iou(t)]             =
Fmeasure_calu(resmap,double(gt),size(gt),threshold);
                        Bi_resmap = zeros(size(resmap));
                        Bi_resmap(resmap>=threshold)=1;
                        threshold_E(t) = Enhancedmeasure(Bi_resmap, gt);
                    end
                    toc;
                    threshold_Emeasure(i,:) = threshold_E;
```

```
                    threshold_Fmeasure(i,:) = threshold_F;
                    threshold_Sensitivity(i,:) = threshold_Rec;
                    threshold_Specificity(i,:) = threshold_Spe;
                    threshold_Dice(i,:) = threshold_Dic;
                    threshold_IoU(i,:) = threshold_Iou;

                end
                toc;

                %MAE
                seq_mae(seqnum) = mean2(MAE);
                %Sm
                seq_Sm(seqnum) = mean2(Smeasure);
                %wFm
                seq_wFm(seqnum) = mean2(wFmeasure);
                %E-m
                seq_column_E(seqnum,:) = mean(threshold_Emeasure, 1);
                seq_meanEm = mean(seq_column_E(seqnum,:));
                seq_maxEm = max(seq_column_E(seqnum,:));
                %Sensitivity
                seq_column_Sen(seqnum,:) = mean(threshold_Sensitivity, 1);
                seq_meanSen = mean(seq_column_Sen(seqnum,:));
                seq_maxSen = max(seq_column_Sen(seqnum,:));
                %Specificity
                seq_column_Spe(seqnum,:) = mean(threshold_Specificity, 1);
                seq_meanSpe = mean(seq_column_Spe(seqnum,:));
                seq_maxSpe = max(seq_column_Spe(seqnum,:));
                %Dice
                seq_column_Dic(seqnum,:) = mean(threshold_Dice,1);
                seq_meanDic = mean(seq_column_Dic(seqnum,:));
                seq_maxDic = max(seq_column_Dic(seqnum,:));
                %IoU
                seq_column_IoU(seqnum,:) = mean(threshold_IoU,1);
                seq_meanIoU = mean(seq_column_IoU(seqnum,:));
                seq_maxIoU = max(seq_column_IoU(seqnum,:));

                fprintf(fileID, '(%s) %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f
%.3f %.3f %.3f\n',...

seqfolder,seq_wFm(seqnum),seq_Sm(seqnum),seq_meanEm,seq_mae(seqnum),seq_meanDic,seq_meanIo
U,seq_maxEm,seq_maxDic,seq_maxIoU,seq_meanSen,seq_maxSen,seq_meanSpe,seq_maxSpe);
                fprintf('(Dataset) seq_wFm   seq_Sm   seq_meanEm   seq_MAE seq_meanDic
seq_meanIoU   seq_maxEm   seq_maxDice   seq_maxIoU   seq_meanSen   seq_maxSen   seq_meanSpe
seq_maxSpe\n (%s) %.3f %.3f %.3f %.3f  %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f \n',...

seqfolder,seq_wFm(seqnum),seq_Sm(seqnum),seq_meanEm,seq_mae(seqnum),seq_meanDic,seq_meanIo
U,seq_maxEm,seq_maxDic,seq_maxIoU,seq_meanSen,seq_maxSen,seq_meanSpe,seq_maxSpe);
            end

            %MAE
            video_mae(videonum) = mean2(seq_mae);
            %Sm
            video_Sm(videonum) = mean2(seq_Sm);
            %wFm
            video_wFm(videonum) = mean2(seq_wFm);
            %E-m
            video_column_E(videonum,:) = mean(seq_column_E, 1);
            meanEm = mean(video_column_E(videonum,:));
            maxEm = max(video_column_E(videonum,:));
```

```
            %Sensitivity
            video_column_Sen(videonum,:) = mean(seq_column_Sen, 1);
            meanSen = mean(video_column_Sen(videonum,:));
            maxSen = max(video_column_Sen(videonum,:));
            %Specificity
            video_column_Spe(videonum,:) = mean(seq_column_Spe, 1);
            meanSpe = mean(video_column_Spe(videonum,:));
            maxSpe = max(video_column_Spe(videonum,:));
            %Dice
            video_column_Dic(videonum,:) = mean(seq_column_Dic,1);
            meanDic = mean(video_column_Dic(videonum,:));
            maxDic = max(video_column_Dic(videonum,:));
            %IoU
            video_column_IoU(videonum,:) = mean(seq_column_IoU,1);
            meanIoU = mean(video_column_IoU(videonum,:));
            maxIoU = max(video_column_IoU(videonum,:));

            fprintf(fileID, '(Dataset) wFm Sm meanEm MAE meanDic meanIoU maxEm maxDice
maxIoU meanSen maxSen meanSpe maxSpe\n (%s) %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f
%.3f %.3f %.3f %.3f\n',...

videofolder,video_wFm(videonum),video_Sm(videonum),meanEm,video_mae(videonum),meanDic,mean
IoU,maxEm,maxDic,maxIoU,meanSen,maxSen,meanSpe,maxSpe);
            fprintf('(Dataset) wFm  Sm  meanEm  MAE meanDic meanIoU maxEm maxDice maxIoU
meanSen maxSen meanSpe maxSpe\n (%s) %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f
%.3f %.3f\n',...
                videofolder,video_wFm(videonum),video_Sm(videonum),meanEm,
video_mae(videonum),meanDic,meanIoU,maxEm,maxDic,maxIoU,meanSen,maxSen,meanSpe,maxSpe);

        end
        fclose(fileID);

    end
```