Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження використання сіамської глибокої мережі з механізмом уваги для One-shot розпізнавання
(тема)

Виконав:
здобувач другого року навчання,
групи ДСм-23-1
Чернявський І. В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Науки про дані (Data Science)
(повна назва спеціалізації)
Керівник проф. Кулішова Н. Є.
(посада, прізвище, ініціали)

Допускається до захисту

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

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Науки про дані (Data Science)
(повна назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Чернявському Іллі Віталійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження використання сіамської глибокої мережі з механізмом уваги для One-shot розпізнавання

затверджена наказом університету від 22 листопада 20 24 р. № 1238Ст

2. Термін подання студентом роботи до екзаменаційної комісії 23 січня 20 25 р.

3. Вихідні дані до роботи Датасети Oryx image, Military Decision-Making Dataset, мова Python, бібіліотеки Scikit-learn, Pillow, Pandas, платформа Kaggle.

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі та постановка задачі дослідження
2) One-shot розпізнавання
3) Практична розробка та навчання моделі

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	виконано
2	Аналіз предметної галузі та постановка задачі	10.12.2024	виконано
3	Дослідження One-shot розпізнавання	20.12.2024	виконано
4	Попередня обробка даних	30.12.2024	виконано
5	Розробка потрібних інструментів для навчання	4.01.2025	виконано
	моделі		
6	Створення моделі	6.01.2025	виконано
7	Навчання моделі	10.01.2025	виконано
8	Аналіз результатів	12.01.2025	виконано
9	Оформлення пояснювальної записки	21.01.2025	виконано
10	Захист кваліфікаційної роботи	23.01.2025	виконано

Дата видачі завдання 25 листопада 2024 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Кулішова Н.Є.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 56 с., 23 рис., 1 дод., 9 джерел.

ATTENTION-BASED SIAMESE NEURAL NETWORK, CHANNEL-ATTENTION, CONVOLUTIONAL NEURAL NETWORK, INCEPTION V3, ONE-SHOT LEARNING, PYTORCH, RESNET 18, SPATIAL ATTENTION, SPATIAL GROUP-WISE ENHANCE, TRAIN SET,VAL SET.

Об’єкт дослідження – процес навчання розпізнавання об’єкту на зображенні, де датасет обмежений за кількістю прикладів цього класу.

Предмет дослідження – глибокі згорткові нейронні мережі в комбінації з механізмом уваги як архітектура сіамської нейронної мережі.

Мета роботи – створення системи розпізнавання техніки на зображенні на основі штучної нейронної мережі, котра буде здатна розпізнати конкретний клас техніки, маючи обмежену кількість навчальних прикладів.

Методи дослідження – аналіз наукових статей та іншої літератури за темою дослідження. Попередня обробка даних, попереднє навчання обраної моделі та її подальша модифікація механізмом уваги, налаштування гіпер параметрів, навчання фінальної сіамської моделі з різними функціями втрат та аналітична оцінка результатів.

В результаті проведених досліджень вирішено задачу розпізнавання різної техніки на зображенні при умовах малої кількості прикладів класів (Few-Shot Learning). Дана модель може базуватися на різноманітних згорткових архітектурах, модифікованих механізмом уваги, показує непогану точність. Обрана модель, з огляду на архітектуру, дає змогу в майбутньому швидко і не затратно додавати нові класи. В якості програмної платформи використано фреймворк Pytorch мови програмування Python.

ABSTRACT

Master’s thesis contains: 56 pp., 23 fig., 1 ann., 9 references.

ATTENTION-BASED SIAMESE NEURAL NETWORK, CHANNEL-ATTENTION, CONVOLUTIONAL NEURAL NETWORK, INCEPTION V3, ONE-SHOT LEARNING, PYTORCH, RESNET 18, SPATIAL ATTENTION, SPATIAL GROUP-WISE ENHANCE, TRAIN SET,VAL SET.

The object of research is the process of learning to recognize an object in an image, where dataset is limited by number of examples of this class.

The subject of research is deep convolutional neural networks in combination with the attention mechanism as architecture of a Siamese neural network.

The purpose of the work is to create a system for recognizing equipment in an image based on an artificial neural network, which will be able to recognize a specific class of equipment, having a limited number of training examples.

Research methods – analysis of scientific articles and other literature on the topic of the study. Pre-processing of data, pre-training of the selected model and its further modification by the attention mechanism, hyper-parameter tuning, training of the final Siamese model with different loss functions and analytical evaluation of the results.

As a result of the conducted research, the problem of recognition different techniques in an image with a small number of class examples (Few-Shot Learning) was solved. This model can be based on a variety of convolutional architectures modified by the attention mechanism and shows good accuracy. The chosen model, due to its architecture, allows adding new classes quickly and inexpensively in the future. The Pytorch framework of the Python programming language was used as a software platform.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CBAM – Convolutional Block Attention Module, комбінований блок уваги;

DataLoader – клас котрий завантажує дані;

Inception – модель що базується на модулях початку;

Pillow – бібліотека створена для машинного зору;

Q, K, V – query, key, value – умовні вектори в механізмі уваги;

ReLU – Rectified Linear Unit – функція активації;

ResNet – математична модель що використовує залишкові блоки;

SGE – Spatial Group-wise Enhance – групова увага;

Sklearn – scikit-learn бібліотека створена для машинного навчання;

Train set – дані для тренування;

Val set – дані для валідації.

ВСТУП

Задача розпізнавання техніки на зображенні у поточній ситуації є дуже актуальною задачею. Потреба в розпізнаванні класів об’єктів, що знаходяться на зображеннях, виникла уже давно, проте значних результатів вдалось досягти лише після створення згорткових нейронних мереж. Для класифікації було запропоновано багато гарних архітектур, найрозповсюдженіші з них це Inception та Resnet. Також можливо використовувати YOLO алгоритм для знаходження розташування та класифікації об’єкту. Проте, кожна з цих нейронних мереж для розпізнавання (класифікації) об’єктів не може в достатній мірі точності розрізняти класи, котрі представлені невеликою обмеженою кількістю прикладів, або навіть одним. За цих обставин, щоб вирішити задачу, що називається Навчанням на невеликій кількості зразків – Few-Shot Learning, в даній роботі пропонується використовувати модель типу Siamese Network. Основою для цієї нейронної мережі є зазвичай згорткова нейронна мережа, в даному випадку це буде попередньо навчені Inception або Resnet. Задля покращення результатів наступні моделі було модифіковано механізмом уваги. Як функції втрат пропонується застосувати Triplet Loss та Contrastive Loss, оскільки це одна з головних частин логіки сіамських мереж, та потребує різної підготовки даних. В даній роботі було досліджено обидві функції.

Метою дослідницької та практичної роботи є створення системи розпізнавання техніки на зображенні на основі штучної нейронної мережі, котра буде здатна розпізнати конкретний клас техніки, маючи обмежену кількість навчальних прикладів. Сферою застосування може бути будь-який застосунок для онлайн-аналізу зображень техніки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз сучасного стану в галузі розпізнавання об’єктів на зображеннях

За останнє десятиліття розпізнавання об'єктів досягло значного прогресу, головним чином завдяки наявності великих наборів даних та розвитку архітектур глибокого навчання. Однак отримання великих наборів маркованих даних для будь-якої задачі неможливе в багатьох практичних сценаріях. Ці обмеження спричинили появу one-shot і few-shot навчання. Механізми одноразового навчання намагаються узагальнити нові класи з дуже малою кількістю маркованих прикладів за допомогою геніальних конструкцій нейронних мереж та функцій втрат. Найвідомішою архітектурою є сіамські нейронні мережі, які є основою для багатьох рішень для однократного навчання. Цей факт також проливає світло на внесок суміжних методів, таких як triplet loss і contrastive loss, які є ефективними для навчання корисних моделей в режимах з невеликою кількістю даних.

Застосування однократного та багатократного навчання охоплює різні сфери, включаючи розпізнавання обличь, верифікацію підписів та класифікацію дрібнозернистих зображень. Крім сіамських нейронних мереж у цих технологіях часто використовуються методи мета навчання, які дозволяють навчити моделі швидко адаптуватися до нових завдань. Приклади включають: прототипові мережі, мережі зіставлення, модельно-агностичне мета навчання (MAML).

Незважаючи на прогрес, one-shot і few-shot навчання стикається з низкою проблем: по-перше, узагальнення дуже мінливих або складних наборів даних часто залишається проблемою. По-друге, моделі часто дають збої, коли завдання вимагає розрізнення дрібнозернистих ознак – наприклад, видів птахів або виразів обличчя.

Іншими словами, гармонія архітектур на кшталт сіамських нейромереж у поєднанні з точно налаштованими функціями втрат, такими як триплетні та контрастні втрати, обіцяє нові перспективи для розпізнавання об'єктів в умовах недостатнього обсягу даних. Межі таких досягнень зараз простягаються до галузей, які потребують потужних і ефективних систем розпізнавання, включаючи охорону здоров'я, робототехніку і безпеку.

1.2 Згорткові нейронні мережі

Згорткові нейронні мережі як вдосконалені аналізатори зображень, по суті, технологію, що дозволяє комп'ютерам «бачити» та інтерпретувати зображення і відео. Інтригує те, що, оскільки цифрове аудіо можна зобразити у вигляді двомірної карти, ці мережі також виявилися високоефективними для обробки звуку. Людський мозок працює пошарово, вловлюючи все більш складні деталі, коли людина спостерігає за чимось [1]. Згорткові нейронні мережі (CNN) функціонують подібним чином. Вони складаються з декількох шарів, кожен з яких працює для виконання певного завдання. Найпомітнішими серед них є шари, що відповідають за базовий аналіз, об'єднувальні шари та повністю пов'язані шари, які інтегрують усю виділену інформацію. Згорткові шари можна порівняти з командою інтелектуальних фільтрів, які досліджують зображення. У міру того, як мережа навчається, фільтри стають дедалі кращими у визначенні ключових особливостей, починаючи від простих елементів, таких як краї, і закінчуючи більш складними, такими як текстури та форми.

Кожен фільтр має так зване «рецептивне поле» (рисунок 1.1) – область, яку він може охоплювати в кожен момент часу. Під час навчання фільтри визначають важливі патерни, багаторазово аналізуючи різні частини зображення. Згортковий шар застосовує набір фільтрів до вхідного зображення або карти ознак. Кожен фільтр – це невелика матриця ваг.

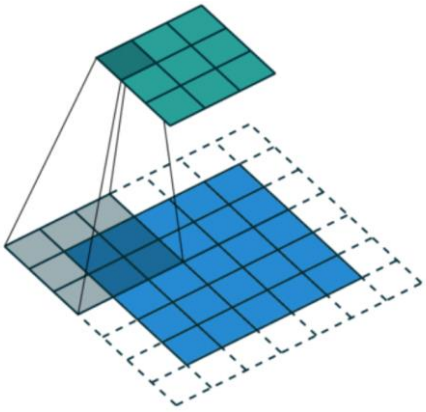


Рисунок 1.1 – Дія фільтру згортки

У згорткових нейронних мережах (CNN) операція згортки відіграє ключову роль у виділенні значущих ознак із вхідних даних, наприклад, із зображень. Операція згортки виконується шляхом накладання фільтра на відповідне рецептивне поле входу: елементи (ваги) фільтра перемножуються з відповідними значеннями вхідних даних, а потім підсумовуються. Рецептивне поле – це невелика область або фрагмент карти ознак. Переміщуючи фільтр по всій карті ознак, ми розраховуємо згортку в кожній просторовій точці, формуючи нову карту ознак.

Щоб контролювати процес згортки, використовуються такі параметри, як крок (stride) і заповнення (padding). Крок визначає, наскільки зміщується фільтр при кожному переміщенні, а padding додає нульові значення по краях карти ознак, запобігаючи її зменшенню при переході між рівнями. Крок визначає зсув, з яким фільтр рухається по карті, і остаточні розміри визначаються формулою

$$w_0 = \frac{w-F+2P}{s}, \tag{1.1}$$

де w_0 – вихідний розмір;

w – вхідний розмір;

F – розмір фільтру;

P – розмір падингу;

s – розмір кроку.

Математично операцію згортки можна визначити наступним чином:

$$Y[i,j] = \sum_m \sum_n X[i + m * s, j + n * s] \cdot K[m,n], \quad (1.2)$$

де Y представляє карту вихідних ознак;

X – карту вхідних ознак;

K позначає фільтр або ядро;

a i, j, m, n – просторові індекси;

s – крок.

Щоб обчислити одне значення в карті вихідної функції, фільтр K поелементно множить відповідні значення в карті вхідної функції X в межах сприйнятливого поля на відповідні значення в карті вхідної функції. Отримані добутки підсумовуються [2]. Процедура повторюється для всіх просторових точок вхідного сигналу, створюючи повну карту вихідної функції Y.

Ця операція згортки може дозволити мережі вловити появу просторових зв'язків і виділити деякі локальні патерни або особливості вхідних даних. У процесі навчання ваги фільтрів налаштовуються таким чином, щоб можна було виявити різні патерни, такі як краї, текстури або більш складні структури, залежно від завдань.

Накладаючи кілька згорткових шарів з різними фільтрами, нейронні мережі можуть поступово вивчати ієрархічні представлення: від низькорівневих ознак, захоплених на ранніх шарах, до високорівневих ознак, захоплених на більш глибоких шарах, які відповідають цілим частинам зображень. Таке ієрархічне виділення ознак дозволяє нейронним мережам ефективно моделювати складні візуальні патерни і робити точні прогнози в таких завданнях, як класифікація зображень, виявлення об'єктів і сегментація зображень.

Невід'ємною частиною згорткових нейронних мереж є шар об'єднання. Він використовується для зменшення вибірки карт особливостей, які виходять з шарів згортки. Операція об'єднання зменшує просторовий розмір карт ознак, зберігаючи найбільш репрезентативні ознаки, таким чином допомагаючи процесу абстрагування ознак на вищих рівнях.

Одна з найпоширеніших операцій об'єднання – max pooling:

$$Y[i,j] = \max_{mn}(X[i \cdot s + m, j \cdot s + n]), \tag{1.3}$$

де Y представляє карту вихідних об'єднаних ознак;

X позначає карту вхідних ознак;

s – крок (розмір кроку для переміщення вікна пулу);

a i, j, m, n – відповідні індекси.

При максимальному об'єднанні карта вхідних ознак розбивається на області, що не перетинаються, або вікна об'єднання, а потім з кожної області витягується максимальне значення [3]. Програма вибирає найсильнішу або найпомітнішу ознаку в кожному вікні об'єднання, вибираючи максимальне значення. Щоб знайти точне значення в об'єднаній карті ознак (Y), вікно об'єднання розміщується в потрібному місці на вхідній карті ознак (X). Фактично, вихідна карта ознак оновлюється максимальним значенням у вікні об'єднання, яке обчислюється як максимальне з відповідних елементів вхідної карти ознак. Це представлено у вихідній карті ознак (рисунок 1.2).

Використовуючи метод максимального об'єднання, шар об'єднання зменшує висоту і ширину карти особливостей, що, в свою чергу, зменшує обчислювальне навантаження на наступні шари. Крім того, максимальне об'єднання дає змогу досягти просторової інваріантності, тобто мережа може розпізнавати ті самі об'єкти, навіть якщо вони дещо зміщені або перевернуті на вхідному зображенні.

Крім максимального об'єднання, існують інші типи операцій об'єднання, наприклад, середнє об'єднання, при якому замість максимального значення пікселів у кожному вікні пулу обчислюється середнє значення пікселів. Такі операції об'єднання можуть демонструвати різні компроміси між збереженням космічної інформації та зменшенням розмірності простору. Однак, правильний варіант, який слід використовувати, залежить від завдання та архітектури мережі.

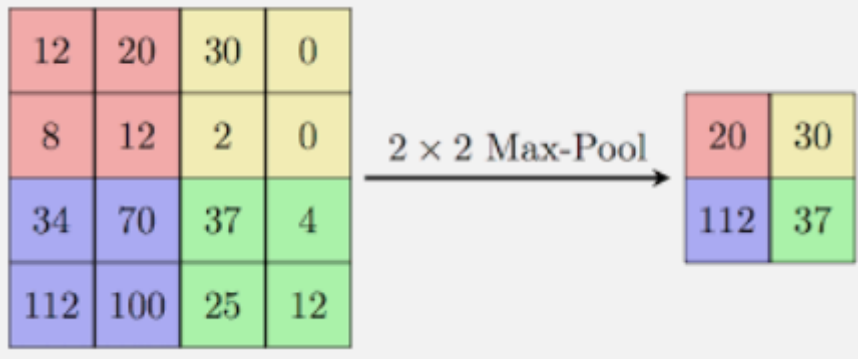


Рисунок 1.2 – Операція max-pooling

По суті, операції об'єднання в CNN зменшують роздільну здатність карт ознак, таким чином відкидаючи менш важливу інформацію і зберігаючи основні ознаки, які будуть використані мережею для захоплення високорівневих репрезентацій, і, як наслідок, можна досягти інваріантності перекладу.

Щільні шари, також відомі як повністю пов'язані шари, часто розміщуються на виході з архітектури мережі. Таким чином, нейрони з попереднього шару з'єднані з кожним нейроном поточного шару. Повністю пов'язані шари виконують обробку витягнутих елементів і дають кінцевий результат. Вони дозволяють мережі вивчати дуже складні взаємозв'язки між елементами і виконувати завдання класифікації та регресії.

Таким чином, ШНМ є дуже ефективною моделлю глибокого навчання, яка здатна аналізувати та видобувати особливості візуальних

даних. Їх ієрархічна структурна перевага і використання згорткових, об'єднаних і повністю пов'язаних шарів роблять згорткові мережі законодавцями моди в комп'ютерному зорі і поштовхом для подальшого розвитку в різноманітних додатках.

1.3 Механізми уваги

1.3.1 Multi-Head Attention

Scaled Dot-Product Attention схематично представлений на рисунок 1.3. Вхідні дані в такій системі поділяються на три частини: запит (query), ключі (key), котрі мають розмірність, яку позначимо як dk, та значень (value) з розміром dv [12].

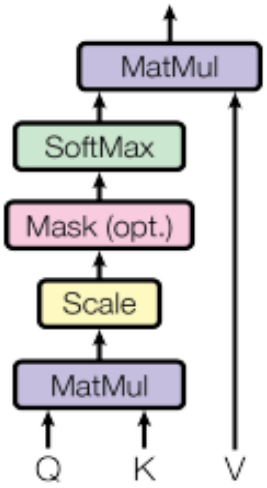


Рисунок 1.3 – Scaled Dot-Product Attention [2]

Наступним кроком потрібно отримати скалярний добуток (dot product) запиту з усіма ключами по черзі. Далі потрібно поділити отриманий результат на корінь від раніше згаданої розмірності dk. Останнім кроком застосовуємо функцію активації Softmax на отриманих результатах [4]. Щоб застосувати це на практиці, функцію уваги виконується одночасно на наборі

запитів, ключів, та значень, які консолідуються відповідні в матриці Q, K і V:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (1.4)$$

де Q – матриця запитів;

K – матриця ключів;

V – матриця значень;

d_k – розмір ключів.

Адитивна увага та увага з множинним добутком є двома найпопулярнішими функціями уваги. Скалярний добуток майже ідентичний описаному алгоритму, за винятком коефіцієнта масштабування $1/\sqrt{d_k}$, тому на практиці він швидший і ефективніший, оскільки може використовувати оптимізований код матричного множення. Адитивна увага, з іншого боку, обчислює функцію сумісності за допомогою мережі прямого поширення з одним прихованим шаром [5]. Хоча обидва механізми мають однакову теоретичну складність, скалярний добуток уваги працює краще для малих значень d_k. З іншого боку, коли d_k більше, адитивна увага перевершує увагу скалярного добутку без масштабування, через те, що для великих значень d_k скалярний добуток стає набагато більшим і, відповідно, це призводить до того, що функція softmax потрапляє в області з дуже малими градієнтами. Ми досягаємо цього шляхом масштабування скалярного добутку на $1/\sqrt{d_k}$.

В механізмі уваги Multi-Head Attention замість виконання єдиної функції звернення уваги за допомогою ключів, значень і запитів d-вимірів виявиляється корисним лінійне проектування запитів, ключів і значень h разів за допомогою різних навчених лінійних проекцій на виміри dk, dk і dv відповідно. Для кожної з цих спроектованих версій запитів, ключів і значень потім реалізується функція уваги паралельно, даючи dv-вимірні вихідні

значення. Вони об’єднуються та знову проектуються, в результаті чого виходять остаточні значення як показано на рисунок 1.4:

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^0, \quad (1.5)$$

$$head_i = Attention(QW^Q, KW^K, VW^V), \quad (1.6)$$

де W^0 – матриця параметрів;

Q, K, V – матриці запитів, ключів та значень;

і – індекс.

Multi-Head Attention дозволяє моделі спільно звертати увагу на інформацію з різних підпросторів представлення в різних позиціях. З однією головою уваги усереднення перешкоджає цьому.

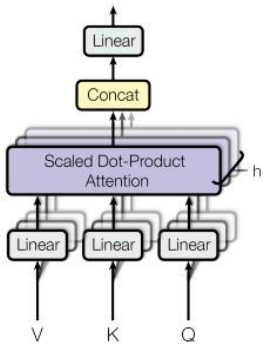


Рисунок 1.4 – Multi-Head Attention [2]

1.3.2 Механізми просторової і канальної уваги

Просторову увагу можна перефразувати так: «На яке місце на зображенні звертає увагу модель?»

Кожне зображення має частини різного значення. Наприклад, коли показано зображення кота, області навколо очей чи вух або візерунки шерсті, ймовірно, матимуть більше значення, ніж решта зображення.

Просторова увага надає ваги більш значущим областям шляхом вивчення змістовних взаємозв'язків у самому зображенні.

Просторова увага працює таким чином: спочатку створюється карта просторової уваги, яка вказує на важливість кожного пікселя чи регіону. Це робиться за допомогою операцій:

– pooling, котра агрегує інформацію про функції по виміру каналу;

– згорток в згорткових шарах для уточнення об'єднаних функцій під час створення карти уваги;

– перезважування: панорамне значення на карті уваги з оригінальними функціями, щоб модель могла приділяти більше уваги відповідним областям просторового розташування (рисунок 1.5).

Зрештою це призводить до ситуації, коли модель витрачає свої обчислювальні ресурси на частини зображення, які є більш ніж значущими та корисними для прийняття рішення.

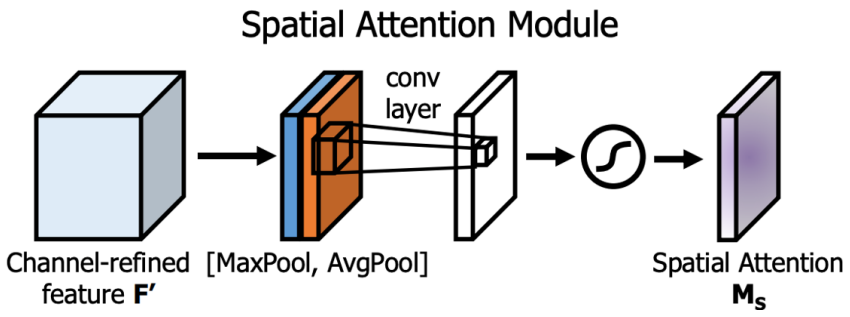


Рисунок 1.5 – Модуль просторової уваги

Канальна увага відповідає на питання: «Які функції найважливіші?»

Кожен із каналів у згортковій карті об'єктів представляє певний тип певної риси об'єкту, як-от краї, текстури чи кольори. Не всі функції будуть корисними для кожного завдання. Увага до каналу допомагає моделі визначити, які канали функцій будуть найбільш релевантними, і посилює їх (рисунок 1.6).

Це потребує трьох наступних кроків:

– глобальне об’єднання: інформація з усього зображення згортається в один дескриптор на канал за допомогою глобального середнього об’єднання або максимального об’єднання;

– повністю зв’язані рівні: ущільнена інформація надсилається через невеликі нейронні мережі, щоб дізнатися важливість кожного каналу;

– повторне зважування: вихідні карти функцій модулюються балами важливості каналу, щоб ключові функції були виділені, а неважливі – пригнічені.

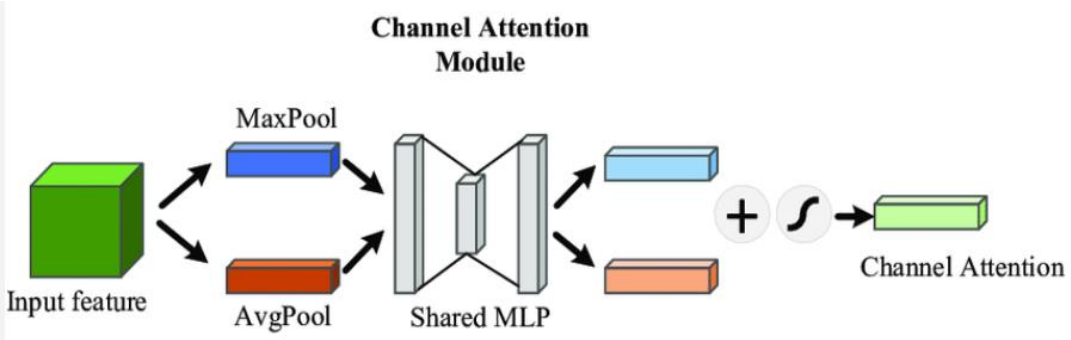


Рисунок 1.6 – Модуль канальної уваги

Канал уваги гарантує, що модель знає, які функції (наприклад, певні кольори чи текстури) важливі для завдання.

1.3.3 Комбіновані модулі уваги

Модуль Convolutional Block Attention Module (CBAM) послідовно комбінує модулі канальної та просторової уваги (рисунок 1.7). Перший модуль зважує карти каналів, потім модифікує їх модулем, котрий зважує конкретні зони зображень. Таким чином покращується здатність моделі аналізувати більш різнопланові зображення, знаходити закономірності вздовж різних розмірностей і концепцій [6].

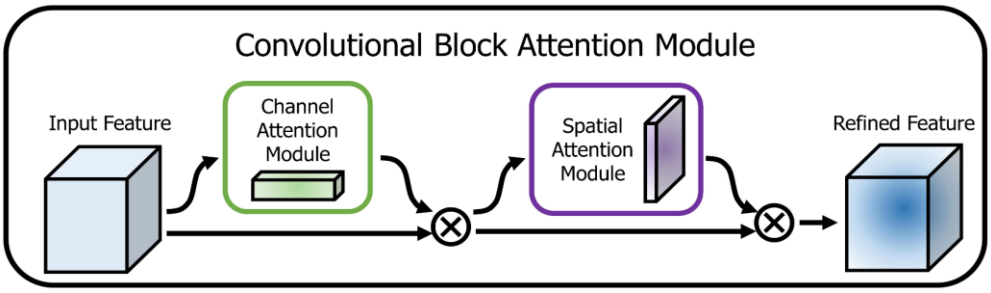


Рисунок 1.7 – Модуль Convolutional Block Attention Module (CBAM)

Ще одним ефективним модулем є SGE (Spatial Group-wise Enhance, групова увага) – це облегшений, з меншою кількістю ваг та обчислень, механізм уваги каналів для покращення репрезентації функцій у CNN [7]. Такі модулі працюють на першому етапі групування карт функцій уздовж виміру каналу з наступним обчисленням просторової уваги для кожної групи незалежно (рисунок 1.8). Модуль діє в три основні етапи:

– групування функцій: вхідна карта функцій групується в кілька груп по каналах;

– обчислення просторової уваги – просторові статистичні дані, такі як середнє або дисперсія для кожної групи, обчислюються для формування карти уваги;

– повторне зважування об’єктів – створена просторова увага застосовується до вихідної карти об’єктів, яка адаптивно покращує важливі просторові розташування.

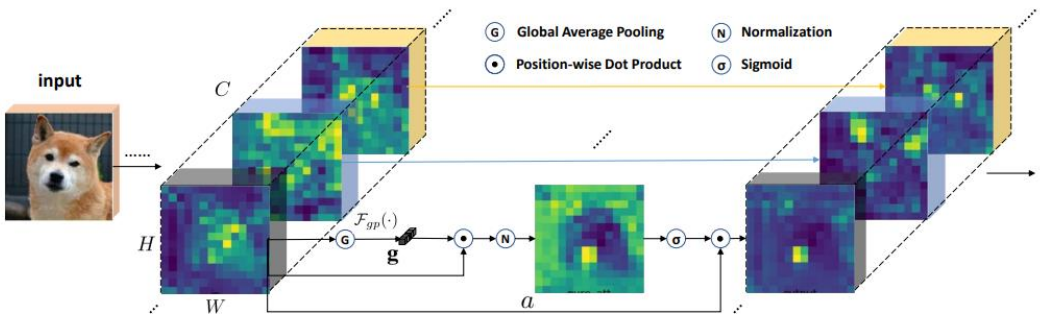


Рисунок 1.8 – Модуль Spatial Group-wise Enhance (SGE)

1.4 Архітектура моделі ResNet

ResNet, скорочення від Residual Network, вирішує одну з найважливіших проблем у навчанні глибоких нейронних мереж: проблему зникаючого градієнта, через яку класичним глибоким мережам важко ефективно поширювати інформацію між багатьма шарами. Цю проблему вдалося подолати, запропонувавши нову концепцію, яка називається залишковим навчанням.

Теоретично, більш глибокі нейронні мережі повинні моделювати більш складні об'єкти. Однак на практиці збільшення глибини призводить до збільшення помилки навчання і труднощів з оптимізацією. Це відбувається тому, що градієнти (сигнали помилок) настільки зменшуються, коли вони поширюються через шари, що це робить оновлення ваг неефективним у дуже глибоких мережах.

ResNet вирішує цю проблему за допомогою залишкових зв'язків. Замість того, щоб намагатися вивчити відображення безпосередньо, він змушує шари вивчати залишок (різницю) між входом і бажаним виходом:

$$y = F(x) + x, \tag{1.7}$$

де x – вхід до шару;

F(x) – функція залишку;

y – остаточний результат.

Архітектура ResNet модулюється на залишкові блоки, що складаються в стек. Типовий залишковий блок реалізований як згортка шарів для вивчення особливостей, пакетна нормалізація для швидшої та стабільнішої збіжності, активація ReLU для введення нелінійності, швидке з'єднання, яке безпосередньо додає вхідні дані до виходу блоку (рисунок 1.9).

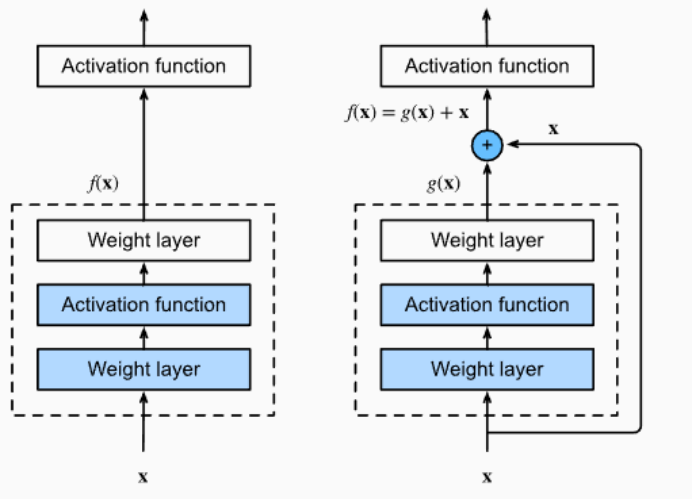


Рисунок 1.9 – Залишковий блок

ResNet – це орієнтир глибокого навчання, який дозволяє мережам масштабуватися до безпрецедентних глибин, не страждаючи від вузьких місць у навчанні, які були притаманні попереднім архітектурам. Її філософія проектування, особливо залишкове навчання, мала великий вплив на багато наступних розробок моделей, ставши, таким чином, основою для нових досягнень в галузі глибокого навчання.

1.5 Inception network

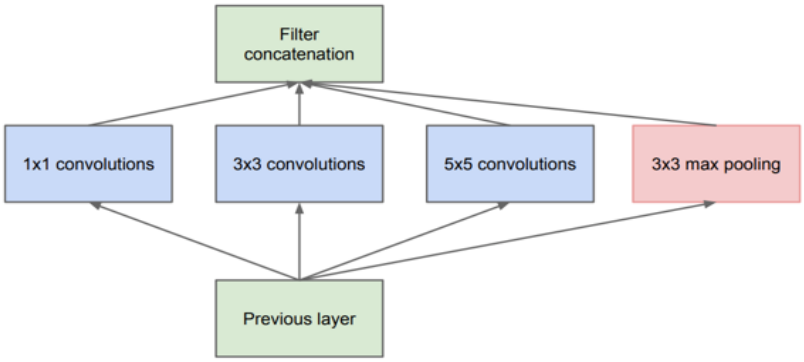
Архітектура Inception досить унікальна тим, що використовує багатомасштабний підхід до виділення функцій. Для згорткових мереж, організованих у шари, зазвичай потрібен лише один розмір ядра на кожному шарі. Inception NN робить це краще, маючи три різні розміри фільтрів (1x1, 3x3 і 5x5) на одному шарі одночасно. Це привносить у модель здатність охоплювати дрібні, середньо- та великомасштабні просторові особливості на зображеннях, створюючи дуже деталізовану карту функцій, де всі елементи зображення пов’язані між собою витонченим чином.

Початкові модулі реалізують три головні ідеї: кожен модуль обробляє вхідні дані, які він отримує через кілька згорток і об’єднань одночасно, де б

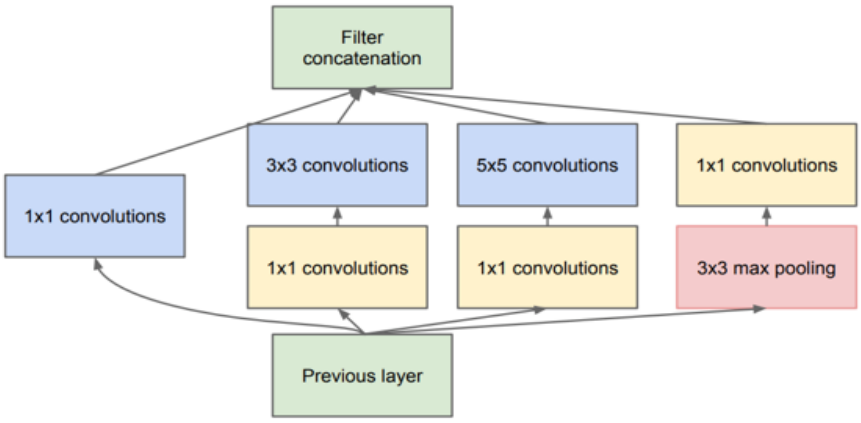
він не був, мережа повертає результати всіх цих робіт через один вихідний канал (рисунок 1.10а), ця концепція економить на вартість обробки та підвищує ефективність.

Згортки 1x1 застосовуються між попереднім і наступним звичайними шарами як шари розмірної ортогоналізації, вони мають першочергове значення для зменшення витрат на обчислення шляхом збільшення кількості параметрів, які потрібно використовувати, без шкоди для репрезентативних можливостей (рисунок 1.10б).

Спеціальні шляхи в модулі Inception відповідають за роботу з певними рівнями абстракції, що дає змогу моделі зосередитися на широкому діапазоні тонкощів зображення.



а)



б)

Рисунок 1.10 – Блок Inception а) початкова конфігурація; б) конфігурація для зменшення розмірності

Нейронна мережа Inception є однією з провідних моделей глибокого навчання, яка показує, як креативність в архітектурі може реагувати на виклики обчислення та продуктивності. Його модульний, багатомасштабний підхід забезпечує ефективний спосіб вилучення детальних і різноманітних функцій із зображень, таким чином показуючи шлях для подальшого розвитку дизайну нейронної мережі.

2 ONE-SHOT РОЗПІЗНАВАННЯ

2.1 Архітектура сіамської нейронної мережі

Сіамські нейронні мережі (Siamese NN) – це клас нейронних архітектур, розроблених для завдань, що вимагають вимірювання подібності або відмінності між парами вхідних даних. Точніше, замість того, щоб обробляти окремі точки даних так, як це зробила б традиційна нейронна мережа, сіамські мережі вивчають якесь спільне представлення між двома або більше входами. Ця єдина структура робить мережу здатною виконувати порівняння та знаходити зв’язки між парами вхідних даних, що виявляється добре в багатьох програмах, особливо для таких завдань, як розпізнавання техніки, обличчя та перевірка підпису (рисунок 2.1).

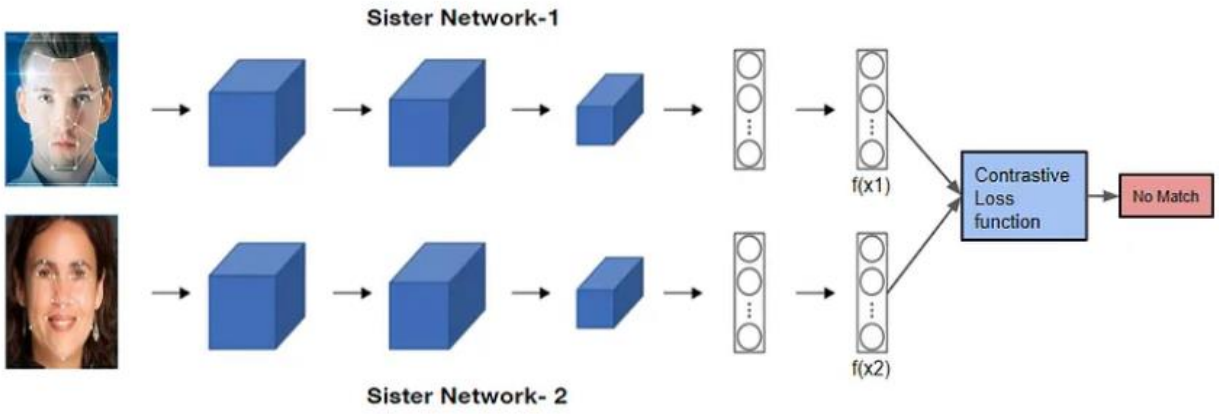


Рисунок 2.1 – Приклад архітектури сіамської мережі

Архітектура сіамської NN: мережа складається з двох або більше ідентичних підмереж з однаковими вагами та параметрами; кожен з них приймає одну з пар вхідних даних і створює вбудовування або вектор ознак у просторі меншої розмірності. Після того, як вхідні дані надходять в ідентичні підмережі, різниця між вбудовуваннями вимірюється за допомогою метрики: евклідової відстані або косинусної подібності. Мережа

навчена оптимізувати метрику відстані за допомогою contrastive loss або функції triplet loss, залежно від завдання. Його мета полягає в тому, щоб залучити в простір вбудовування пари входів, які схожі один на одного, і відштовхнути ті, які не схожі.

Такий розподіл ваги між підмережами має важливе значення для забезпечення того факту, що мережа навчається узгодженому перетворенню для всіх вхідних пар, не пов’язаних із конкретною точкою даних. Ця архітектура найкраще обслуговує багато реальних додатків, зокрема, якщо порівнювати або зіставляти вхідні дані, які не є ідентичними, але мають однакові функції, наприклад, перевіряючи, чи є два зображення однієї особи або чи розповідають два фрагменти тексту про те саме.

Методи навчання, які використовуються для сіамських нейронних мереж, включають визначення функції втрат так, що подібні вхідні пари пов’язані з високим балом, а різні вхідні пари – з низьким балом. Іншими популярними функціями втрат є контрастні втрати та триплетні втрати.

Мережі Simease навіть мають деякі переваги, особливо у випадках, коли доступні позначені дані обмежені. Використовуючи пари даних, сіамські мережі можуть вивчати змістовну дискримінаційну мову навіть на меншій кількості прикладів [4]. Крім того, їхня здатність бути інваріантною до таких варіацій, як точка зору, освітлення чи вираз у завданнях зображення, робить їх ще більш корисними на практиці.

З іншого боку, сіамські мережі мають свої недоліки. Ефективність моделі тісно пов’язана з використовуваною функцією втрат, наприклад, налаштування параметрів запасу має вирішальне значення для найкращої продуктивності. Крім того, незважаючи на те, що сіамські мережі можуть набувати дуже характерних особливостей, вони можуть не працювати задовільно в завданнях, які потребують великої кількості різноманітних даних, таких як завдання класифікації з багатьма різними категоріями.

2.2 Функції втрат в сіамській мережі

Сіамські нейронні мережі зазвичай застосовуються для завдань навчання подібності, де намагаються визначити, чи є два вхідних екземпляри схожими чи ні. Найпоширенішими функціями втрат, прийнятими в сіамських мережах, є контрастні втрати та триплетні втрати.

Контрастивні втрати: це функція втрат, яка штрафує мережу, якщо відстань між подібними парами велика або якщо відстань між різними парами мала. Звичайний спосіб визначення функції контрастних втрат показаний як:

$$Loss = 1/2N \sum {}^{N}_{i=1} ((1-y){d_i}^2 + y_i\, max(margin - d_i, 0)^2), \quad (2.1)$$

де d_i представляє відстань між двома векторами, які є виходом переднього шару;

y_i представляє мітку зразок пари;

0 означає, що два зображення належать до тієї самої категорії;

1 означає, що два зображення не пов’язані між собою.

Контрастна втрата добре поводиться у простіших випадках, коли парних відносин достатньо, щоб зафіксувати подібність. Переваги цієї втрати полягають у тому, що вона простіша та обчислювально дешевша, ніж триплетна втрата, а також її легше реалізувати та навчити для сценаріїв з двійковими мітками. Одним із недоліків є те, що вона не фіксує відносну різницю між кількома прикладами, такими як опорні та позитивні зразки.

Триплетні втрати – це тип функції втрат, який використовується, коли одночасно порівнюються три вхідні дані: опорний вхід, позитивний приклад (схожий на якір) і негативний приклад (не схожий на якір). Триплетна функція втрат гарантує, що з запасом a опорна точка буде ближче до позитивного прикладу, ніж до негативного прикладу:

$$Loss = 1/N \sum\nolimits_{i=1}^{N} max(0, d_{ap} - d_{an} + margin), \quad (2.2)$$

де d_{ap} – відстань між анкером і позитивними вкладеннями;

d_{an} – відстань між опорою та негативними вкладеннями;

m – маржа.

Це має такі переваги: триплетна функція втрат фіксує відносну подібність між кількома прикладами та є більш надійною для завдань, де важливі зв'язки між кількома класами. Її недоліки: потрібно ретельно відбирати триплети, інакше погана вибірка призведе до повільної конвергенції. Більш обчислювально дорогий підхід, оскільки потрібно обробляти трійки.

2.3 Модифікація моделі за допомогою механізму уваги

Щоб дізнатися про найкращу комбінацію механізмів уваги та архітектури нейронної мережі, було досліджено, як об'єднати різні модулі уваги за допомогою Inception V3 та ResNet 18. Точніше, в них було вбудовано механізми уваги: SE, SK, SGE, ECA. Оскільки механізми уваги не змінюють розміри карт ознак, їх можна вставити в будь-який згортковий шар. Однак, як компроміс між обчислювальною ефективністю та необхідною архітектурою базових мереж, було побудовано гібридну схему нейронної мережі на основі уваги.

Модель Inception V3 складається з 11 блоків, розділених на п'ять категорій, кожна з яких має кілька шарів згортки та об'єднання. Оскільки сама архітектура Inception V3 є настільки розвинутою, занадто багато модулів уваги можуть втрутитись в мережу та погіршити структуру мережі. Навіть додавання лише одного модуля уваги на блок призведе до додавання 11 нових модулів, що, безумовно, занадто багато. Щоб уникнути цього, додано один модуль уваги на тип блоку, що додало п'ять модулів уваги.

ResNet 18 містить стандартні шари згортки та об’єднання, а також, що більш важливо, два важливі залишкові блоки. Оскільки робота цих залишкових блоків залежить від пропуску з’єднань, вставлення модулів уваги безпосередньо в них порушить їх структуру. Замість цього кожна пара залишкових блоків розглядалася як один рівень, тож вийшло чотири шари. Тепер модуль уваги вставлено в кінці кожного з цих чотирьох рівнів, тобто, в архітектуру ResNet 18 додано чотири модулі уваги. Таким чином можливо оптимізувати інтеграцію цих мереж за допомогою стратегічного розміщення механізмів уваги з мінімальним впровадженням структурних збоїв і обчислювальних витрат.

3 ПРАКТИЧНА РОЗРОБКА ТА НАВЧАННЯ СИСТЕМИ

3.1 Набір використаних програмних інструментів

Розглянемо набір програмних інструментів, котрі були використані при розробці запропонованої моделі. Мовою програмування для вирішення цієї задачі є python. Середовищем програмування було обрано Jupyter notebook, оскільки він має зручну блочну систему та зрозумілий інтерфейс. За етап роботи з файлами датасету в директорії сховища відповідає бібліотека os. Завантажено дані за допомогою бібліотеки pillow для зображень та pandas для міток. Для обробки та подальшого використання використовувалися torchvision та pillow. Для збереження та завантаження обробленого датасету відповідає бібліотека pytorch. Для реалізації моделі та всіх пов’язаних з нею операцій було використано ефективний та гарний фреймворк для машинного навчання Pytorch. В кінці для побудування графіків були використані бібліотеки matplotlib та tensorboard.

3.2 План розробки

Для початку було обрано стартову модель, ResNet18, оскільки були обмежені ресурси. Задля кращого результати було вирішено обрати уже попередньо навчену її варіацію на датасеті ImageNet, котру модифіковано за допомогою механізмів уваги. Наступним кроком модель навчено на чистому та гарному датасеті класифікації військової техніки. Далі потрібно буде розробити сіамську нейронну мережу та спробувати до навчити модель на задачі розпізнавання конкретної техніки на більш реальному і місцями поганому датасеті. Звісно, перед цим потрібно буде обробити датасети та підготувати їх до використання в моделях. В кінці потрібно буде продемонструвати та порівняти отримані результати.

3.3 Попередня обробка даних та розробка потрібних інструментів для навчання моделі

Для попереднього навчання моделі обрано датасет Military Decision-Making Dataset з Kaggle. Він є досить чистий з зображеннями гарної якості для детекції об’єктів з різних класів. В датасеті представлено наступні класи військової техніки (рисунок 3.1).

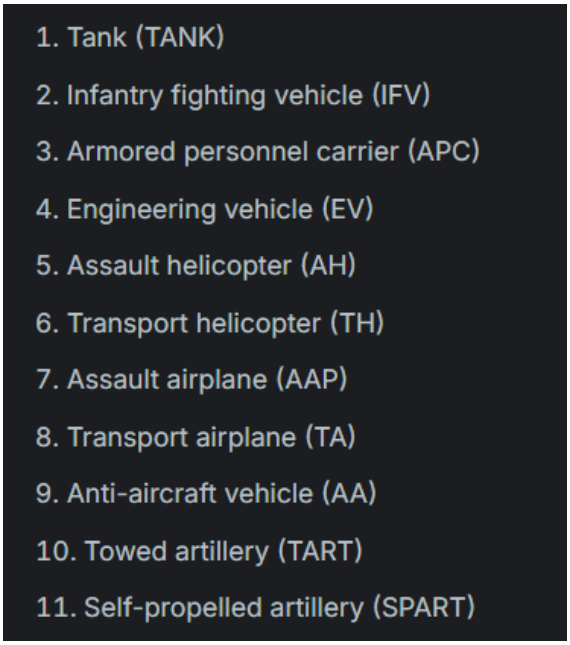


Рисунок 3.1 – Назви класів техніки на зображеннях та їх мітки

Для початку зображення завантажено за допомогою Pillow, з попередньою перевіркою валідності файлу. Оскільки датасет розроблений для детекції об’єктів, то мітки зберігаються в форматі для YOLO алгоритму. Отже, доведеться завчасно вирізати техніку по Bounding box (лістинг 3.1), якщо на картинці є більше одного об’єкту, а якщо менше, то ціле зображення та приводим їх до розміру 320 × 320 пікселів в трьох каналах.

Лістинг 3.1 – Програмний код функції вирізання картинок

```
def crop_and_resize(img, bbox, rescale_size):
```

Продовження лістингу 3.1

```
    img_width, img_height = img.size
    center_x, center_y, bbox_width, bbox_height = bbox
    center_x = int(center_x * img_width)
    center_y = int(center_y * img_height)
    bbox_width = int(bbox_width * img_width)
    bbox_height = int(bbox_height * img_height)
    left = max(0, center_x - bbox_width // 2)
    top = max(0, center_y - bbox_height // 2)
    right = min(img_width, center_x + bbox_width // 2)
    bottom = min(img_height, center_y + bbox_height // 2)
    cropped_img = img.crop((left, top, right, bottom))
    resized_img = custom_transform(cropped_img, rescale_size)
    return resized_img
```

Датасет вже розділений на навчальний та валідаційний. Приклад 10 зображень після обробки показаний на рисунку 3.2.

Рисунок 3.2 – Приклад зображень

На наступному кроці перераховано кількість прикладів кожного класу для того, щоб дізнатися чи присутній в датасеті дисбаланс класів (рисунок 3.3).

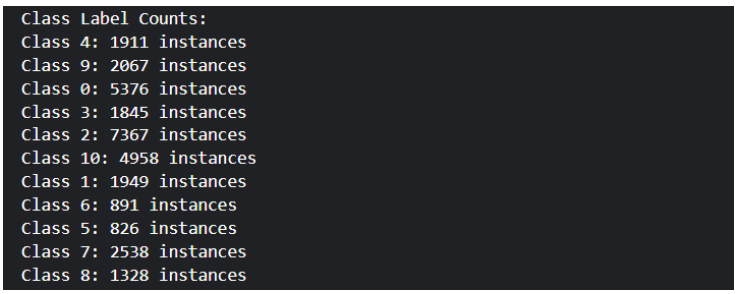


Рисунок 3.3 – Кількість прикладів кожного класу

Визначено присутність суттєвого дисбалансу класів, для вирішення цієї проблеми під час класифікації згенеровано ваги у залежності від кількості прикладів класу: чим більше прикладів класу в датасеті, тим менше ваги (рисунок 3.4).

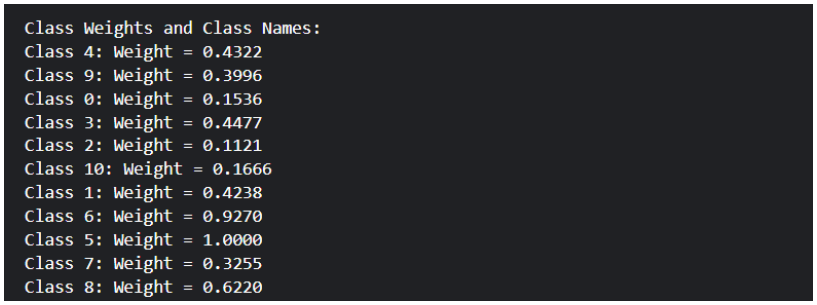


Рисунок 3.4 – Розраховані ваги кожного класу

Далі для нормалізації пораховано середнє значення та стандартне відхилення по каналам (лістинг 3.2). MEAN – (0.5097, 0.524, 0.5099), STD – (0.212, 0.212, 0.237).

Лістинг 3.2 – Mean і STD

```
channel_sum = np.zeros(3)
channel_squared_sum = np.zeros(3)
num_pixels = 0
batch_size = 100
for i in tqdm(range(0, len(train_images), batch_size)):
```

Продовження лістингу 3.2

```
    batch_images = train_images[i:i + batch_size]
    for img in batch_images:
        img_array = np.array(img) / 255.0  # Shape: (320, 320, 3)
        channel_sum += img_array.sum(axis=(0, 1))
        channel_squared_sum += (img_array ** 2).sum(axis=(0, 1))
        num_pixels += img_array.shape[0] * img_array.shape[1]
channel_mean = channel_sum / num_pixels
channel_std = np.sqrt(channel_squared_sum / num_pixels - channel_mean ** 2)
```

Оскільки в датасеті містяться близько 11800 картинок, гарною практикою є аугментація даних. Маючи ліміт по обсягу оперативної пам’яті, неможливо провести аугментацію заздалегідь, тому розроблено модуль для динамічної аугментації під час завантаження даних в модель (лістинг 3.3). Розберемо функції в трансформаторі зображень. Спочатку приведено Pillow зображення до тензору, потім з 25% ймовірністю зображення горизонтально обернено, змінено перспективу та нормалізовано.

Лістинг 3.3 – Динамічний обробник зображень

```
transform_v1 = transforms.Compose([
    transforms.ToTensor(),
    transforms.RandomHorizontalFlip(p=0.25),
    transforms.RandomRotation(degrees=25),
    transforms.RandomPerspective(distortion_scale=0.6, p=0.25),
    transforms.Normalize(MEAN, STD)])
```

Таким чином аугментація даних збільшить датасет та дозволить збільшити різноманітність даних, зображення будуть ніби з різних ракурсів.

Визначено клас «кастомний датасет», котрий буде використовувати DataLoader для завантаження даних. Кастомний датасет буде обирати по

індексам дані та переводити їх в потрібний формат, аугментувати зображення і повертати. DataLoader формує батчі (пакети), розділяє їх на багато потоків процесору та перемішує значення. Дані класи створено на 4 потоках та завантажено туди датасет (лістинг 3.4).

Лістинг 3.4 – Завантажувач даних

```
train_dataset = SimpsonsDataset(train_images, train_labels,
'train', transform_v1)
train_dataloader = DataLoader(train_dataset,
batch_size=BATCH_SIZE, num_workers=4, shuffle=True)
val_dataset = SimpsonsDataset(val_images, val_labels, 'val',
transform_v1)
val_dataloader = DataLoader(val_dataset,
batch_size=BATCH_SIZE, num_workers=4)
```

Створено клас метрик для збирання метрик по всьому валідаційному датасеті, котрий буде збирати відмінності між парами по кожному батчу (пакету), а потім за потреби обраховувати різні метрики (лістинг 3.5). В даному дослідженні за допомогою бібліотеки Sklearn розраховуються такі метрики, як точність, Recall, Precision, F1.

Лістинг 3.5 – Клас розрахунку метрик

```
class Metrics():
    def __init__(self):
        self.all_actual = np.array([])
        self.all_predicted = np.array([])
    def batch_step(self, actualv, predictedv):
        self.all_actual = np.concatenate([self.all_actual,
actualv.numpy(force=True)])
        self.all_predicted =
np.concatenate([self.all_predicted,
predictedv.numpy(force=True)])
    def get_metrics(self):
```

Продовження лістингу 3.5

```
        recall = recall_score(self.all_actual,
self.all_predicted, average='weighted', zero_division=0)
        precision = precision_score(self.all_actual,
self.all_predicted, average='weighted', zero_division=0)
        f1 = f1_score(self.all_actual, self.all_predicted,
average='weighted', zero_division=0)
        accuracy = accuracy_score(self.all_actual,
self.all_predicted)
        return recall, precision, f1, accuracy
```

3.4 Створення моделі

Як модель було обрано для реалізації ResNet18 через її невеликий розмір і високу швидкість навчання. Щоб не навчати її з нуля, було підвантажено ваги, навчені на датасеті ImageNet. Зображення датасетів ImageNet і military-equipment не дуже схожі, та, очевидно, мають різні розподіли. Все ж таки, модель, гарно навчена на одному датасеті, може розпізнавати прості образи та форми, наприклад автомобілей. Тому було розраховано нові параметри mean і std для нормалізації зображень на мілітарному датасеті.

Почнемо зі створення нових модулей уваги, спочатку створено CBAM як самий простий механізм (лістинг 3.6). Класи Channel та Spatial були також реалізовані для головної моделі CBAM. Для розумних стартових ваг застосовано ініціалізатор Ксав'єра.

Лістинг 3.6 – Клас CBAM

```
class CBAM(nn.Module):
    def __init__(self, in_channels, reduction_ratio=16):
        super(CBAM, self).__init__()
        self.channel_attention = ChannelAttention(in_channels,
reduction_ratio)
```

Продовження лістингу 3.6

```
        self.spatial_attention = SpatialAttention()
    def forward(self, x):
        x = self.channel_attention(x)
        x = self.spatial_attention(x)
        return x
```

Далі реалізовано модуль SGE, котрий на практиці показує кращі результати, та потребує менше обчислювальних потужностей (лістинг 3.7). Надалі він використовується як пріоритетний через значні обмеження в пристрої для обчислення.

Лістинг 3.7 – Реалізація класу SGE

```
class SpatialGroupEnhance(nn.Module):
    def __init__(self, groups = 64):
        super(SpatialGroupEnhance, self).__init__()
        self.groups = groups
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.weight = Parameter(torch.zeros(1, groups, 1, 1))
        self.bias = Parameter(torch.ones(1, groups, 1, 1))
        self.sig = nn.Sigmoid()
    def forward(self, x): # (b, c, h, w)
        b, c, h, w = x.size()
        x = x.view(b * self.groups, -1, h, w)
        xn = x * self.avg_pool(x)
        xn = xn.sum(dim=1, keepdim=True)
        t = xn.view(b * self.groups, -1)
        t = t - t.mean(dim=1, keepdim=True)
        std = t.std(dim=1, keepdim=True) + 1e-5
        t = t / std
        t = t.view(b, self.groups, h, w)
        t = t * self.weight + self.bias
        t = t.view(b * self.groups, 1, h, w)
        x = x * self.sig(t)
```

Продовження лістингу 3.7

```
        x = x.view(b, c, h, w)
        return x
```

Наступним кроком створено модель на основі ResNet18, модифіковану кастомними модулями уваги. Шари, що називались layerN, були перенесені з оригінальної моделі ResNet, layer0 було зібрано з оригінальних шарів згортки, пакетної норми, Relu та максимального пулінгу. Останній середній пулінг був залишений оригінальним та змінений останній fully connected шар для класифікації нових об’єктів. В кінці розставлено чотири нові модулі після шарів layerN (лістинг 3.8). На всі оригінальні шари було завантажено попередньо навчені ваги, а нові ініціалізовані за допомогою уніформи Ксав’єра. Потрібно також зазначити, що кількість каналів повинна ділитися на кількість груп в модулях SGE. Проте, розмір зображення до і після цього модуля не змінюється.

Лістинг 3.8 – метод нового класу ResNet18SGE

```
def forward(self, x):
    x = self.layer0(x)
    x = self.layer1(x)
    x = self.module1(x)
    x = self.layer2(x)
    x = self.module2(x)
    x = self.layer3(x)
    x = self.module3(x)
    x = self.layer4(x)
    x = self.module4(x)
    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.fc(x)
    return x
```

3.5 Навчання моделі класифікації техніки

Створимо модель, котра повинна класифікувати 11 класів, дивлячись на кількість каналів, модулі уваги поділено на (2, 4, 4, 4) групи. Налаштовано learning rate для шарів, котрі використовуються попередньо навченими. Цей параметр буде менше (5e-5), а для нових більше (5e-4). Як оптимайзер було обрано AdamW, котрий модифікований розпадом ваг. В стандартну крос ентропійну функцію втрат для класифікації було додано ваги.

Функцію навчання було розроблено досить лінійно, і стандартно для класифікації. Вона ітерується по лоадеру та побатчево завантажує дані в модель; рахує функцію втрат, проводить зворотне поширення, запускає функцію оптимайзеру (лістинг 3.9). Також вона рахує час та різні метрики та повертає значення лосу. Для валідаційного датасету не обчислюється зворотне поширення і не проводиться навчання оптимайзером, проте на ньому підраховуються всі метрики та будуються графіки результатів навчання. Також ваги моделі зберігаються для її подальшого використання.

Лістинг 3.9 – Функція тренування

```
def train_step(model, loss_fn, opt, loader, batch_size):
    loss_per_batches = 0
    for i, data in tqdm(enumerate(loader),
total=len(train_labels)//batch_size):
        features, labels = data
        features, labels = features.to(device),
labels.to(device)
        opt.zero_grad()
        y_pred = model(features)
        loss = loss_fn(y_pred, labels)
        loss.backward()
        opt.step()
        loss_per_batches += loss
```

Продовження лістингу 3.9

```
    return loss_per_batches/(i+1)
```

З початку навчання прийнято батч 64, learning rate 5e-5 та 5e-4, кількість груп 2, 4, 4, 4. Модель навчилася досить гарно, про що можливо судити з графіку втрат (рисунок 3.5), та таких метрик, як F1, Recall і Precision (рисунок 3.6). Проте після 30 епохи модель почала видавати Nan через занадто малі значення. Отже, оптимально навченою можна вважати модель після 20+ епохи.

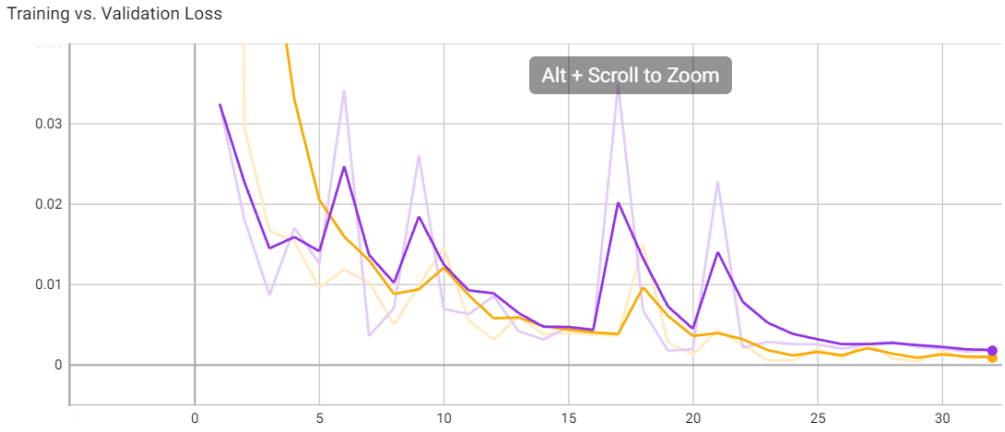


Рисунок 3.5 – Графік втрат

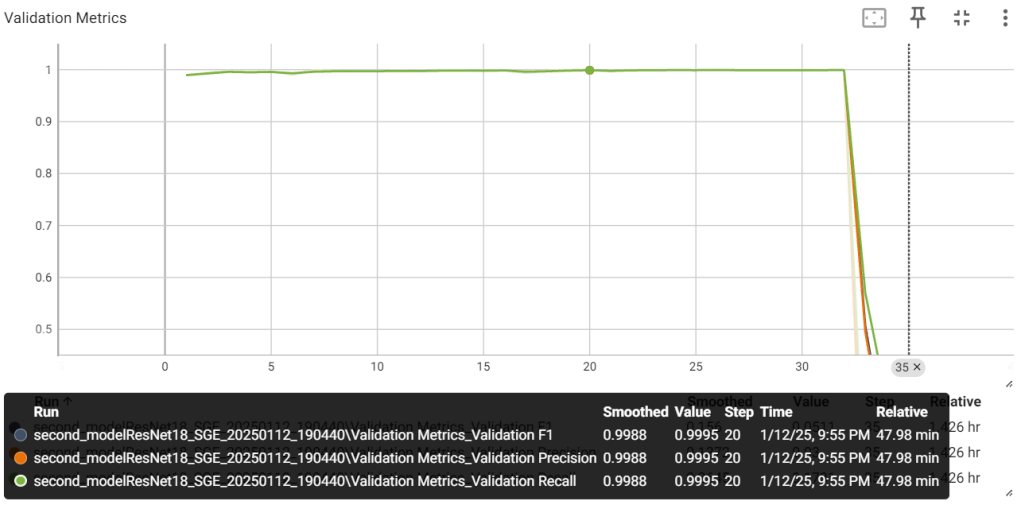


Рисунок 3.6 – Графік метрик

Далі будуть використовувані ваги, збережені після 20 епохи. Можна побачити цікаву тенденцію, коли отримано Nan числа в результаті навчання, проте модель уже встигла добре навчитися раніше, тому можна взяти більш ранній варіант моделі. Для подальшої роботи збережено модель з 20 епохи, з наступними валідаційними метриками LOSS = 0.00217, Accuracy = 0.9991, Recall = 0.9990, Precision = 0.9991, F1 = 0.9990.

3.6 Підготовка даних для навчання сіамської нейронної мережі

Було обрано два датасети – один, знятий на полігоні з добре розрізнювальною технікою і класами, котрі гарно відрізняються. Другий датасет є більш реалістичний – Oryx dataset з реальними даними, знятими з дронів та камер в бойових умовах, багато техніки є повністю знищеною або пошкодженою, на деяких картинках дуже багато техніки зниженої поруч та такої, розлетілася на частини. Багато знімків з висоти та змазаних (рисунок 3.7). Таким чином, було прийняте рішення, що техніку потрібно обирати власноруч, та відсіювати дуже значні викиди та аномалії. Також потрібно правильно обрати класи.

```
[10]: equipment
      Infantry_Fighting_Vehicles                175
      Trucks,_Vehicles,_and_Jeeps               150
      Tanks                                     133
      Armoured_Fighting_Vehicles                 74
      Engineering_Vehicles_And_Equipment         67
      Reconnaissance_Unmanned_Aerial_Vehicles    59
      Towed_Artillery                            24
```

Рисунок 3.7 – Розподіл прикладів навчання по класам

Наступним кроком датасет було розділено на тренувальний та валідаційний, також потрібно було впевнитися, що в валідаційному і тестовому датасеті присутні пари для кожного елементу. Після цього фото масштабовані до розміру 320×320 та 3 каналів (лістинг 3.10).

Лістинг 3.10 – Завантаження та попередня обробка даних

```
try:
    img = Image.open(img_path)
    if img.mode != 'RGB':
        img = img.convert('RGB')
    resized_img = custom_transform(img, RESCALE_SIZE)
    image_list.append(resized_img)
    label_list.append(row[target_name])
```

Наступним кроком буде кодування таргетів за допомогою LabelEncoder з бібліотеки sklearn. Спочатку його навчено на списку класів, щоб потім можна було швидко закодувати та розкодувати мітки.

Також було знову вжито попередній динамічний обробник даних з попереднього навчання (лістинг 3.3), та Mean та STD з попереднього навчання.

Далі потрібно створити клас «датасет» для створення триплетів, цей клас містить в собі зображення, мітки та словник з використаними зображеннями, розподіленими по класам. Отже, щоб надалі коректно зробити триплети, потрібно правильно зробити батчі (пакети), щоб вони містили позитивні картинки для всіх якорів (лістинг 3.11). Спочатку проводиться пошук всіх зображень того ж класу в датасеті, звірка їх з уже використаними картинками, вибір серед них випадкового не використаного елементу того самого класу. В результаті створено батч, та вдвічі більший батч позитивних пар. Зображення обробляються за допомогою динамічного обробника зображень.

Лістинг 3.11 – Кастомний датасет для триплетів

```
def __getitem__(self, index):
    img1 = self.transform(self.images[index])
    label = self.labels[index]
    indices = np.where(self.labels == label)[0]
    if len(self.used_ind[label]) == len(indices):
```

Продовження лістингу 3.11

```
        self.used_ind[label] = []
    indices = np.delete(indices, np.where(indices == index))
    if len(self.used_ind[label]) == len(indices) and index not
in self.used_ind[label]:
        self.used_ind[label] = []
    else:
        indices = np.delete(indices, np.where(np.isin(indices,
self.used_ind[label])))
    positive_idx = random.choice(indices)
    img2 = self.transform(self.images[positive_idx])
    self.used_ind[label].append(positive_idx)
    if self.mode == 'test':
        return img1, img2
    else:
        label = self.labels[index]
        return img1, img2, torch.tensor(label,
dtype=torch.long)
```

Для автоматичного формування триплетів використовується hard batching [8]. Тобто, в парах визначається відстань між елементами, створюється позитивна та негативна маска. Далі знаходять самий дальній елемент з позитивною міткою до якоря та самий ближчий з негативною (лістинг 3.12). Таким чином формують триплети і передають їх в триплетну функцію втрат.

Лістинг 3.12 – Створювач триплетів

```
def TripletLoss(loss_fn, embeddings, labels, mode='train',
batch_size=None):
    distances = torch.cdist(embeddings, embeddings, p=2)
    labels = labels.unsqueeze(1)
    mask_positive = labels == labels.T
    mask_negative = ~mask_positive
    distances_positive = distances.clone()
```

Продовження лістингу 3.12

```
    distances_positive[~mask_positive] = -float('inf')
    hard_positive_indices = distances_positive.argmax(dim=1)
    distances_negative = distances.clone()
    distances_negative[~mask_negative] = float('inf')
    hard_negative_indices = distances_negative.argmin(dim=1)
    anchor = embeddings
    positive = embeddings[hard_positive_indices]
    negative = embeddings[hard_negative_indices]
    if mode == 'val':
        anchor = anchor[:batch_size]
        positive = positive[:batch_size]
        negative = negative[:batch_size]
    return loss_fn(anchor, positive, negative)
```

Переробимо клас метрик з класу (лістинг 3.5) для сіамської нейронної мережі. По-перше, потрібно додати повний датасет до даних. Головна діагональ заповнюється дуже великими числами, щоб уникнути само вибору. Зображення відсортовано за мінімальною відстанню, і створено ранжовані мітки. Далі проведено класифікацію за методом KNN для обрання класів. Також потрібно порахувати бінарну відповідність пар та обрахувати рахунок, для обрахування mean average precision score (MAP). MAP є основною метрикою для оцінювання сіамських нейронних мереж, оскільки вона враховує не тільки правильність або не правильність класу, а ще й ступень відхилення, у нашому випадку це відстань [9]. Тому MAP є найоптимальнішою мірою для оцінки моделей даного типу. В кінці знайдено середнє значення для average precision score та розраховано recall, precision, f1, accuracy (лістинг 3.13).

Лістинг 3.13 – Обрахування метрик

```
distance_matrix = pairwise_distances(self.all_embeddings,
np.concatenate([self.all_embeddings, train_emb]),
```

Продовження лістингу 3.13

```
metric="euclidean", force_all_finite=False)
np.fill_diagonal(distance_matrix, np.inf)
ranked_indices = np.argsort(distance_matrix, axis=1)
all_labels = np.concatenate([self.all_actual, train_lbl])
ranked_labels = all_labels[ranked_indices]
ap_scores = []
for i in range(len(self.all_actual)):
    y_true = (ranked_labels[i][:-1]==
self.all_actual[i]).astype(int)
    y_score = -distance_matrix[i][ranked_indices[i]][:-1]
    ap_scores.append(average_precision_score(y_true, y_score))
y_pred = np.array(y_pred)
mAP = np.mean(ap_scores)
```

Наступним проапгрейдом буде створення окремого класу, котрий буде запам’ятовувати всі метрики всіх епох, виводити їх в консоль під час навчання, записувати результат в SummaryWriter для побудування графіків під час навчання. Також цей клас буде відміряти час епохи, слідкувати за ними та виводити найкращий результат.

В кінці роботи з даними та написання допоміжних функцій створено екземпляри триплетного датасету та дата лоадеру, котрий його використовує (лістинг 3.14). Буде застосовано 4 потоки, дані для тренувального датасету будуть перемішані.

Лістинг 3.14 – Створення даталоадерів

```
train_dataset = TripletDataset(train_images, enc_tlabels,
'train', transform_v1)
train_dataloader = DataLoader(train_dataset,
batch_size=BATCH_SIZE, num_workers=4, shuffle=True)
val_dataset = TripletDataset(val_images, enc_vlabels, 'val',
transform_v1)
```

Продовження лістингу 3.14

```
val_dataloader = DataLoader(val_dataset,
batch_size=BATCH_SIZE, num_workers=4)
```

3.7 Створення моделі та навчання сіамської мережі

Оскільки сама ідея сіамської нейронної мережі полягає в використанні виходів однієї нейронної мережі, формування з її виходів триплетів та обчислення втрат, то розумно буде використати попередньо навчену на попередньому датасеті модель для класифікації. Щоб використовувати модель (лістинг 3.8) її модифіковано наслідуванням від класу попередньої моделі (лістинг 3.15), тим самим виключено з обчислень класифікаційний шар, в кінці отримано ембедінги, зберігаючи всі інші властивості і методи, крім перевизначеного методу forward.

Лістинг 3.15 – Модифікована модель

```
class ResNet18WithSGEFeatureExtractor(ResNet18WithSGE):
    def __init__(self, num_classes, groups, pretrained=True):
        super(ResNet18WithSGEFeatureExtractor,
self).__init__(num_classes, groups, pretrained)
    def forward(self, x):
        x = self.layer0(x)
        x = self.layer1(x)
        x = self.module1(x)
        x = self.layer2(x)
        x = self.module2(x)
        x = self.layer3(x)
        x = self.module3(x)
        x = self.layer4(x)
        x = self.module4(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        return x
```

Функцію навчання модифіковано для використання з сіамською нейронною мережею. Головною відмінністю є вид даних і те, що їх обсяг збільшено вдвічі для створення пар, та зменшено, коли потрібно порахувати метрики, щоб вони не були розраховані двічі. Також використовуються класи метрик та клас MetricsWriter для запису виводу на екран метрик і збереження моделі.

Останнім кроком буде створення об’єкту моделі, завантаження вагів попередньо навченої моделі, створення Writer та налаштування тріплет втрат та оптимізатора. При створені моделі потрібно задавати таку ж саму кількість класифікаційних нейронів, щоб правильно було завантажено ваги з попереднього навчання, проте, далі не буде використовуватися шар класифікації (лістинг 3.16). Модель переводиться на потрібний пристрій, та завантажується потрібний чекпоїнт моделі, навченої на попередньому датасеті.

Лістинг 3.16 – Створення моделі та всіх необхідних інструментів

```
model = ResNet18WithSGEFeatureExtractor(num_classes=11,
groups=[2, 4, 4, 4], pretrained=True)
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
model = model.to(device)
checkpoint_path = '/kaggle/input/model-
weights/second_model_ResNet18_SGE_20250112_190440_20'
checkpoint = torch.load(checkpoint_path, map_location=device)
model.load_state_dict(checkpoint)
mw = MetricsWriter(model, model_name='capt20_1-5_1_try_model',
save_treshold=10)
loss_func = nn.TripletMarginWithDistanceLoss(margin=0.5)
optimizer = optim.AdamW(model.parameters(), lr=1e-4,
weight_decay=1e-4)
```

3.8 Навчання та аналіз результатів.

Враховуючи обчислювальні спроможності, була обрана для експериментів найменша з моделей, описаних вище ResNet18. Модуль уваги був обраний такий, що потребує менше параметрів для навчання SGE. Досліджуючи датасети, можливо помітити, що датасет на полігоні має більше прийнятних зображень, оскільки реальний датасет Oryx має дуже багато неякісних фото для підтвердження знищення техніки. Тож, довелось в ручному режимі перебирати і підбирати для навчання моделі приблизно 700 екземплярів.

Почнемо навчання з датасету military-equipment, котрий є дуже чистим, техніка знаходиться в основному на полігоні і її гарно видно. Налаштовано margin = 1.0, та learning rate = 1e-4, значення втрати наведено на рисунку 3.8.

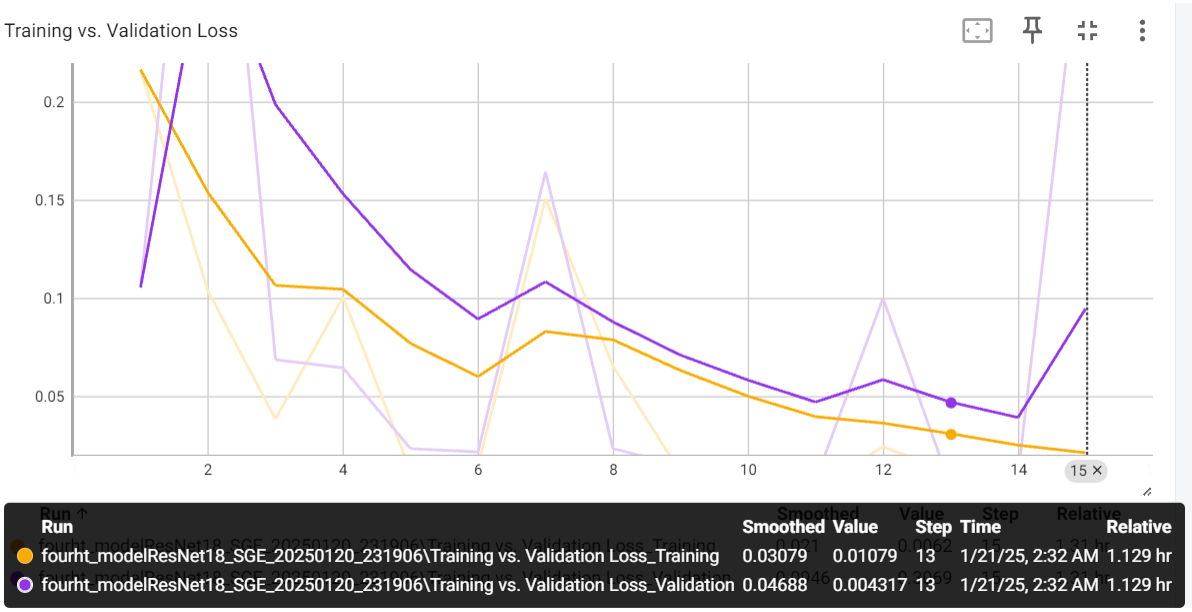


Рисунок 3.8 – Графік втрат

Видно, що значення втрат гарно сходяться приблизно на 14 епосі. Далі побудовано графік MAP для демонстрації того, як добре справляється з

завданням модель, збалансована не тільки за втратами, а і за відстанню між зображеннями. Судячи з цієї метрики (рисунок 3.9), модель дійсно гарно навчилася розпізнаванню військової техніки.

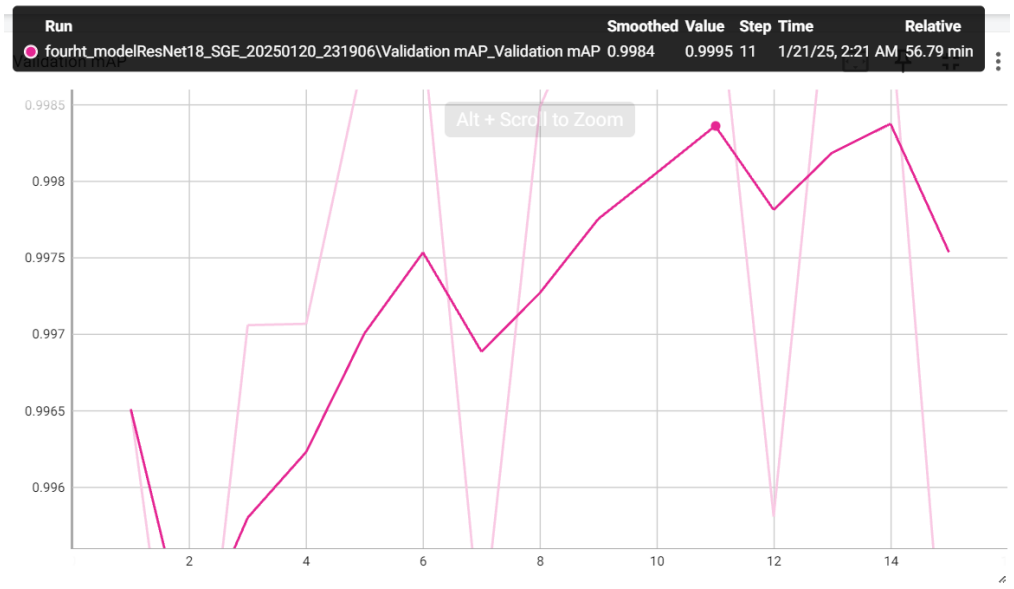


Рисунок 3.9 – Графік MAP

Перейдемо до датасету з реальними зображеннями поля бою. Підготовлених в ручному режимі даних в даному випадку дуже мало, близько 700 прикладів. На цих прикладах моделі буде складно навчатися через різноплановість фото, наявність великої кількості зайвих об’єктів на одному зображенні, згорілої та підірваної техніки, котра розташовується поруч, та, навіть, просто зображень, знятих з дуже великої висоти або змазаних.

Налаштуємо гіперпараметри, а саме margin та learning rate (рисунок 3.10). Для досліду, котрий представлений фіолетовою лінією, гіперпараметри мають значення m = 0.7, lr = 1e-5. Він показав непогану динаміку навчання, проте кінцевий параметр бажано бачити вищим, ніж 0.49. Помаранчевий графік показує другий запуск з гіперпараметрами m = 0.5, lr = 1e-4 з результатом приблизно 0.52. Третій експеримент було проведено з m = 1.5, lr = 1e-4, він позначений зеленим кольором.

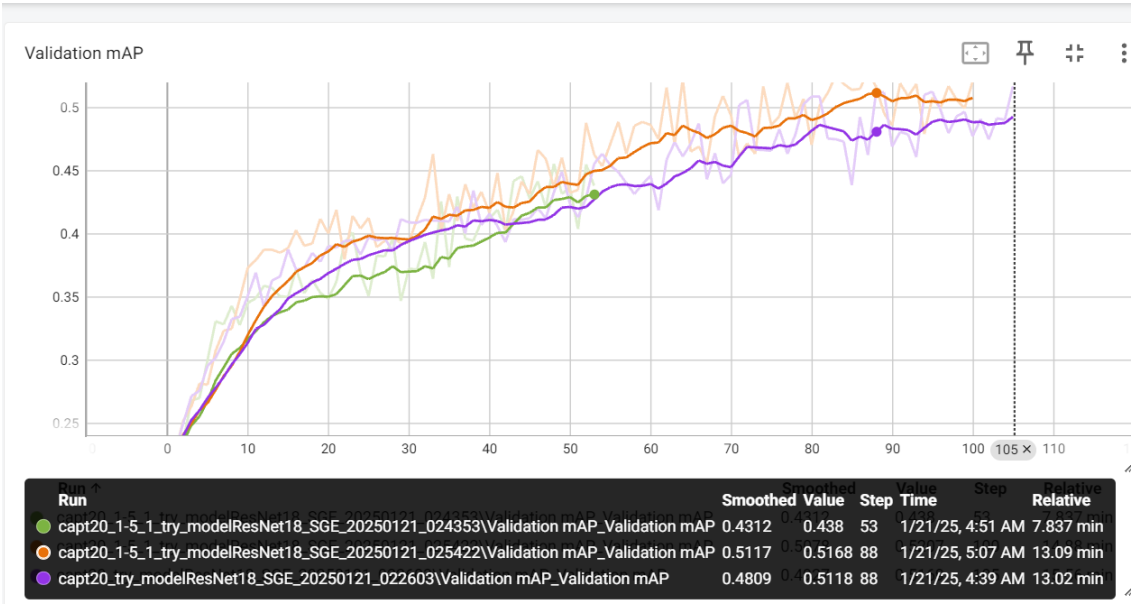


Рисунок 3.10 – Графік mAP

Судячи з графіку функції втрат, найбільш успішним виявився також другий експеримент (рисунок 3.11), хоча те, що розмір margin найменший, могло вплинути на функцію втрат.

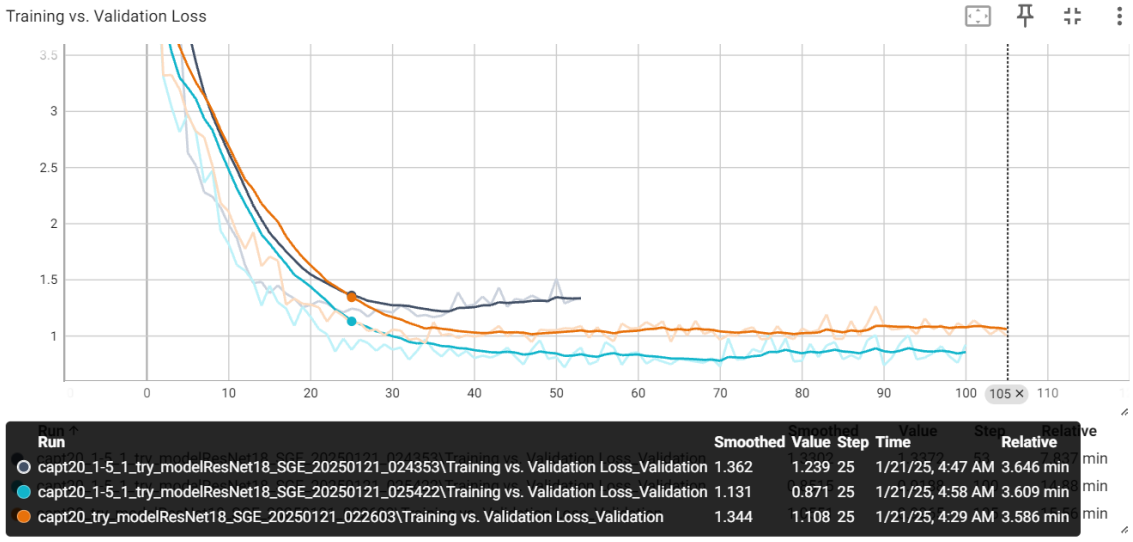


Рисунок 3.11 – Графік функції втрат

Останньою метрикою буде точність (рисунок 3.12), на графіку помітна тенденція, що всі три експерименти мають схожу середню точність.

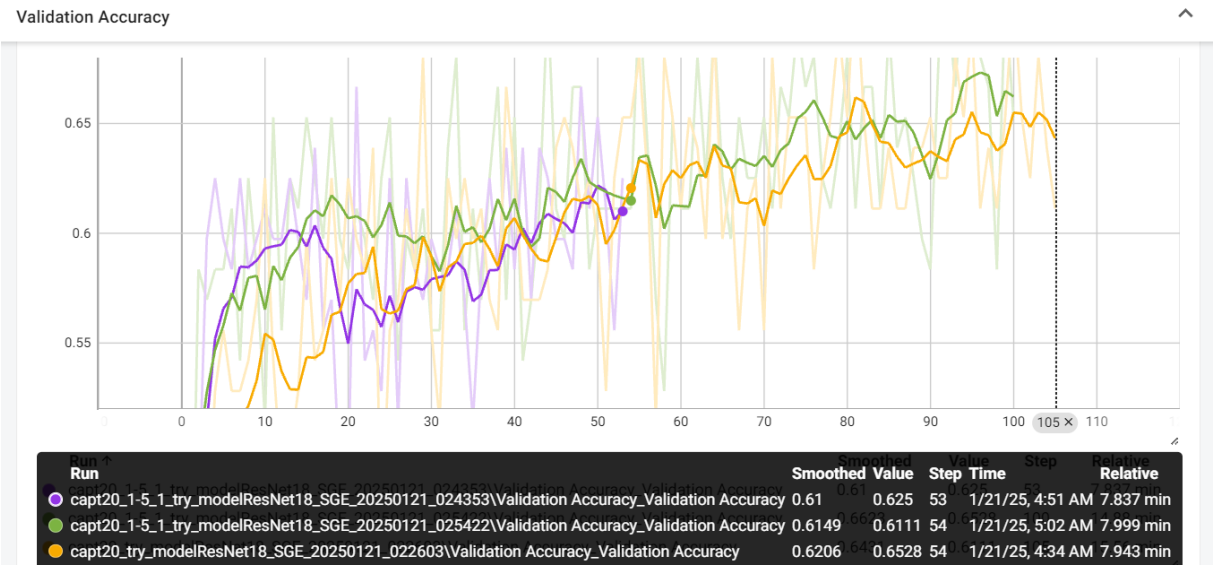


Рисунок 3.12 – Графік точності навчання модифікованої моделі

Отже на чистому, полігонному датасеті military-equipment модель навчилася на відмінно, а на реальному бойовому моделі не вистачило як кількості, так і якості зображень. Проте, Oryx – це єдиний бойовий датасет у відкритому доступі, тому, щоб надалі покращувати модель, його потрібно повністю перебирати.

ВИСНОВКИ

У ході проведеного дослідження розпізнавання об'єктів на зображенні при умові невеликої кількості тренувальних прикладів було виконано аналіз попередніх напрацювань по цій темі. Після цього було прийнято рішення обрати сіамську нейронну мережу, модифікувати її та виконати експерименти з різними функціями втрат під час навчання.

Для цього було проведено теоретичне ознайомлення з предметною сферою. Були вивчені методи попередньої обробки зображень. Наступним кроком було вивчення теоретичного обгрунтування побудови моделі та принципу її навчання, після чого модель з модифікаціями було реалізовано на практиці та проведено ряд експериментів.

Конкретний метод вирішення задачі було обрано через його простоту, сіамські мережі показали гарні результати в подібних задачах. Тому було вирішено модифікувати саме сіамську нейронну мережу механізмами уваги та взяти за основу перевірену та стабільну модель класифікації об’єктів на зображенні ResNet18.

Проведене дослідження підтвердило ефективність обраного методу для вирішення задачі One-shot розпізнавання. Отримані результати можуть бути використані в подальших дослідженнях та розробках у сфері розпізнавання об’єктів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mayank Mishra. Convolutional Neural Networks, Explained. Published on Aug 26, 2020. URL: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939 (дата звернення: 05.11.2024).

2. Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. arXiv:1706.03762v5 [cs.CL] 6 Dec. 2017, pp. 5999- 6009. URL: https://arxiv.org/pdf/1706.03762.pdf (дата звернення: 06.11.2024).

3. Wanshun Wong. What is Residual Connection? Published on Dec 19, 2021. URL: https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55 (дата звернення: 22.10.2024).

4. Souvik Mandal. Power of Siamese Networks and Triplet Loss: Tackling Unbalanced Datasets Published on May 17, 2023 https://medium.com/@mandalsouvik/power-of-siamese-networks-and-triplet-loss-tackling-unbalanced-datasets-ebb2bb6efdb1 (дата звернення: 03.11.2024).

5. Zihao Huang, Yue Wang, Application of attention-based Siamese composite neural network in medical image recognition Published on 15 Mar 2024 https://arxiv.org/pdf/2304.09783 (дата звернення: 09.10.2024).

6. Sanghyun Woo, Jongchan Park, Joon-Young Lee, In So Kweon CBAM: Convolutional Block Attention Module Published 18 Jul 2018 https://arxiv.org/abs/1807.06521 (дата звернення: 28.12.2024).

7. Xiang Li, Xiaolin Hu, Jian Yang Spatial Group-wise Enhance: Improving Semantic Feature Learning in Convolutional Networks Published 25 May 2019 https://arxiv.org/abs/1905.09646 (дата звернення: 28.12.2024).

8. Alexander Hermans, Lucas Beyer, Bastian Leibe: In Defense of the Triplet Loss for Person Re-Identification Published 21 Nov 2017 https://arxiv.org/pdf/1703.07737 (дата звернення: 15.01.2025)

9. Deval Shah: What is Mean Average Precision (mAP), how to calculate it, and why is it important for evaluating models' performance? Published on Mar

7, 2022 URL: https://www.v7labs.com/blog/mean-average-precision (дата звернення: 05.01.2025).