**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу**

**Кафедра штучного інтелекту**

До захисту допущено:

В. о. завідувачки кафедри

_____ Ірина ДЖИГИРЕЙ

«___»_____20__ р.

# Дипломна робота

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Системи і методи штучного інтелекту»**

**спеціальності 122 «Комп'ютерні науки»**

**на тему: «Використання методів штучного інтелекту для задач розпізнавання військових об'єктів на аерофотознімках»**

Виконав:
студент IV курсу, групи КІ-02
Дмитерчук Віталій Олександрович _____

Керівник:
доцент, к.ф.-м.н., доц. Тимошенко Ю. О. _____

Консультант з економічного розділу:
професор, д.е.н., проф. кафедри ЕК ФММ,
Шевчук Олена Анатоліївна _____

Консультант з нормоконтролю:
фахівець першої категорії кафедри ШІ,
Кравець П. В. _____

Рецензент:
доцент, к.т.н., доц. кафедри ІБ ФТІ,
Гальчинський Л. Ю. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Навчально-науковий інститут прикладного системного аналізу**
**Кафедра штучного інтелекту**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Системи і методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В. о. завідувачки кафедри
_____ Ірина ДЖИГИРЕЙ
«31» січня 2024 р.

# ЗАВДАННЯ

**на дипломну роботу студенту**

**Дмитерчуку Віталію Олександровичу**

1. Тема роботи «Використання методів штучного інтелекту для задач розпізнавання військових об'єктів на аерофотознімках», керівник роботи Тимошенко Олександр Юрійович, к.ф-м.н., доцент, затверджені наказом по університету від «31» травня 2024 р. №2240-С.

2. Термін подання студентом роботи «10» червня 2024 року.

3. Вихідні дані до роботи: зображення, отримані в результаті аерофотозйомки.

4. Зміст роботи: дослідження предметної області, вивчення теоретичних основ систем розпізнавання і класифікації об’єктів на зображенні, аналіз та модифікація вхідних даних, реалізація системи розпізнавання та класифікації об’єктів, функціонально-вартісний аналіз програмного продукту.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація.

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Економічний | Шевчук Олена Анатоліївна, професор, д. е. н. | | |

7. Дата видачі завдання «  » 2024 року.

Календарний план

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Дослідження предметної області | 20.04.2024 | Виконано |
| 2 | Підготовка першого розділу | 27.04.2024 | Виконано |
| 3 | Підготовка другого розділу | 04.05.2024 | Виконано |
| 4 | Підготовка даних до виконання задачі | 11.05.2024 | Виконано |
| 5 | Розробка програмного продукту | 18.05.2024 | Виконано |
| 6 | Підготовка третього розділу | 25.05.2024 | Виконано |
| 7 | Підготовка четвертого розділу | 30.05.2024 | Виконано |
| 8 | Підготовка економічної частини | 05.06.2024 | Виконано |
| 9 | Оформлення роботи відповідно до вимог нормоконтролю | 08.06.2024 | Виконано |
| 10 | Підготовка презентації роботи | 10.06.2024 | Виконано |

Студент Віталій ДМИТЕРЧУК

Керівник Юрій ТИМОШЕНКО

# РЕФЕРАТ

Дипломна робота: 97 с., 30 рис., 6 табл., 18 посилань, 1 додаток.

ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, ЗГОРТКОВІ НЕЙРОННІ МЕРЕЖІ, РОЗПІЗНАВАННЯ, КЛАСИФІКАЦІЯ, ВІЙСЬКОВА ТЕХНІКА, УКРІПЛЕННЯ, АЕРОРОЗВІДКА, БПЛА.

Об’єкт дослідження – методи автоматизованого розпізнавання та класифікації об’єктів на зображеннях.

Предмет дослідження – моделі згорткових нейронних мереж для задач розпізнавання й класифікації військових об’єктів на аерофотознімках.

Мета роботи – проаналізувати існуючі моделі згорткових нейронних мереж, порівняти їхню ефективність та застосувати отриману інформацію для розробки програми, що розпізнає та класифікує військові об’єкти на аерофотознімках.

Актуальність цієї дипломної роботи зумовлена поширенням застосування безпілотних літальних апаратів для виконання задач бойової розвідки та високим практичним потенціалом застосування методів автоматичного розпізнавання військових об’єктів на отриманих таким чином даних. Це дозволить значно прискорити процес передачі інформації щодо розташування та дій сил ворога, що в свою чергу підвищить ефективність бойової роботи всіх родів військ.

В результаті виконання роботи було проаналізовано структурні компоненти згорткових нейронних мереж та види моделей, що можуть бути використані для розпізнавання й класифікації об’єктів. Навчено й протестовано дві такі моделі та реалізовано програмне забезпечення для цієї задачі.

# ABSTRACT

Master's thesis: 97 p., 30 figures, 6 tables, 18 references, 1 appendix.

ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, CONVOLUTIONAL NEURAL NETWORKS, RECOGNITION, CLASSIFICATION, MILITARY TECHNOLOGY, FORTIFICATION, AERIAL RECONNAISSANCE, UAV.

The object of the study is methods of automated recognition and classification of objects in images.

The subject of research is models of convolutional neural networks for the recognition and classification of military objects on aerial photographs.

The purpose of the work is to analyze the existing models of convolutional neural networks, compare their effectiveness and obtain the information obtained for the development of programs that recognize and classify military objects on aerial photographs.

The relevance of this thesis is due to the expansion of the use of unmanned aerial vehicles for the performance of combat reconnaissance tasks and the high practical potential of applying the methods of automatic recognition of military objects to the data obtained in this way. This will significantly speed up the process of transmitting information about the location and actions of the enemy's forces, which in its version will increase the effectiveness of the combat work of all types of troops.

As a result of the work, the structural components of convolutional neural networks were analyzed and models that can be used for object recognition and classification are visible. Two such models were trained and tested, and software for this task was implemented.

ЗМІСТ

# ВСТУП

В умовах ведення будь-яких бойових дій виявлення техніки, укріплень та особового складу противника є першочерговою задачею. Що раніше встановлено місцезнаходження, тип озброєння, кількість об’єктів, то більше залишається часу для підготовки гідної оборони або необхідних маневрів власних сил. Ця задача набуває критичного значення під час повномасштабної війни з ворогом, що має значну чисельну, логістичну та економічну перевагу. Для досягнення мети з якнайшвидшого виявлення ворожих сил в кожній бригаді та батальйоні ЗСУ існують відповідно розвідувальні рота й взвод з необхідним для цього оснащенням, зокрема з безпілотними авіаційними комплексами.

Російсько-українська війна стала першим в історії озброєним протистоянням з широким використанням безпілотних апаратів всіх типів обома сторонами конфлікту. Основними задачами таких апаратів наразі є:

- розвідка з повітря як безпосередньо лінії фронту, «сірої зони» між укріпленнями сторін, так і тилу противника;
- коригування дружньої артилерії під час обстрілу ворожих цілей;
- знищення техніки, особового складу, укріплень шляхом доставки боєприпасів різних типів й або їх прицільного скиду, або зіткнення безпілотного апарату з ціллю, або використання спорядженої на апараті вогнепальної зброї;
- дистанційне встановлення або нейтралізування мінних загороджень без привертання уваги ворога до переміщень особового складу;
- доставка боєприпасів та спорядження за небезпечних умов (наприклад, під час артилерійського обстрілу) або у важкодоступні місця;
- ретранслювання радіосигналу на необхідній висоті для забезпечення максимального покриття місцевості.

В цій роботі увагу зосереджено на задачі повітряної розвідки з використанням безпілотних авіаційних комплексів (БпАК), а точніше на автоматизації частини робочого процесу оператора БпАК. Безпілотні літальні апарати, що входять в склад зазначених комплексів, під час польоту здійснюють високоякісну фото- або відеозйомку за допомогою встановлених камер, записуючи отримані матеріали на файловий носій. Після приземлення оператор аналізує фото або відео, відшукуючи та позначаючи на них ворожі об’єкти, аби потім передати дані про їх розміщення до командування. Очевидно, цей процес є кропітким і часомістким та міг би бути автоматизований за умови високої точності програми, що його виконує, звільнивши час та сили військовослужбовців на виконання інших задач або відпочинок. Така програма, що також має бути обчислювально нескладною й мати функціонал для визначення типу об’єкта, може бути реалізована з використанням технологій штучного інтелекту. Нейронні мережі, що лежать в основі цих технологій, за умови якісного навчання здатні швидко й з високою точністю визначати розташування та клас об’єктів на зображенні.

Ця дипломна робота присвячена аналізу можливостей застосування методів згорткових нейронних мереж для вирішення задач розпізнавання й класифікації військових об’єктів на аерофотознімках. Метою роботи є створення системи розпізнавання військових об'єктів на аерофотознімках з використанням технологій згорткових нейронних мереж та машинного навчання, а також порівняння якості роботи різних архітектур нейромереж та вибір найкращої з них.

# РОЗДІЛ 1 ПОСТАНОВКА ЗАДАЧІ АВТОМАТИЗАЦІЇ РОЗПІЗНАВАННЯ ВІЙСЬКОВИХ ОБ’ЄКТІВ

## 1.1 Актуальність проблеми

Як було зазначено у попередньому розділі, швидкість виявлення розташування та реакції на дії сил ворога є ключовим елементом успіху будь-якої воєнної операції, особливо в умовах поточного повномасштабного вторгнення. Тому такі ресурси як час та уважність розвідника, який по суті є першою ланкою процесу виявлення ворога, мають високу ціну й не витрачаються без вагомої на то причини. Ця робота покликана створити дієвий метод економії цих ресурсів шляхом автоматизації рутинного часомісткого процесу та за рахунок прискорення комунікації підвищити бойову ефективність всіх дотичних підрозділів на лінії бойового зіткнення.

## 1.2 Огляд основних типів об’єктів військового призначення

В цій роботі буде розглянуто 5 узагальнених категорій військових об’єктів, що є поширеними на лінії бойового зіткнення (ЛБЗ). Вибір та розподіл за типами було здійснено на основі знань та досвіду, отриманих в районі бойових дій. Визначені категорії мають достатні відмінності між собою, аби об’єкти вважались цілями або загрозами різного масштабу, але в той же час є достатньо загальними, аби включати в себе безліч підвидів та подібних між собою варіацій. Отже, об’єкти поділено на такі типи:

- капонір – стаціонарне одиничне укріплення порівняно великої площі зі штучним заглибленням нижче рівня землі та насипом по периметру. Призначені в першу чергу для укриття техніки від

ураження осколками снарядів, а також для її маскування. Приклад капоніру показано на рис. 1.1;

- окоп – лінійні укріплення витягнутої форми зі штучним заглибленням та насипом, призначені для укриття особового складу від куль, осколків тощо; дозволяють порівняно безпечно переміщуватись вздовж ЛБЗ та за необхідності ефективно оборонятись від піхоти противника. Приклад такого укріплення позначено на рис. 1.2;
- броньована техніка – ця категорія включає в себе зокрема танки, БМП, БТР та САУ – іншими словами, всю важку техніку військового призначення. Один з порівняно небагатьох помічених об’єктів цього типу показано на рисунку 1.3;
- вантажні автомобілі – вантажівки військового призначення, позначено на рис. 1.4;
- цивільні автомобілі – ця категорія включена до типів об’єктів, що потребують визначення, по причині того, що такі транспортні засоби часто використовуються особовим складом ворога з метою непомітного траснпортування вантажів військового призначення або скритного переміщення невеликого підрозділу. До цього типу відносяться зокрема й «буханки» – малотонажні фургони підвищеної прохідності марки УАЗ, – що дуже часто використовуються військовими силами РФ для цілої низки задач. Приклад позначення об’єктів цієї категорії показано на рис. 1.5.

**Рисунок 1.1** – Засніжені капоніри вздовж дороги, прикриті деревами

**Рисунок 1.2** – Розгалужена мережа окопів, розмічена рамками для подальшого навчання нейронної мережі

**Рисунок 1.3** – Позначені зеленими рамками засніжені танки біля дороги

**Рисунок 1.4** – Вантажівка посеред галявини

**Рисунок 1.5** – Скупчення цивільних авто біля будівлі

### 1.3 Аналіз існуючих підходів

З очевидних причин у військовий час складно або неможливо знайти у відкритому доступі інформацію про розробки програмного забезпечення за спрямуванням, подібним до теми цієї роботи. Тим не менш, деякі інші способи застосування нейронних мереж для розпізнавання об’єктів використовуються як нашою, так і ворожою армією, та є загальновідомими у військових колах.

Прикладом такої технології може слугувати візуальне та електромагнітне автонаведення ударних БпЛА на ціль, коли вона знаходиться в зоні прямої видимості. Подібною системою відомий російський дрон-камікадзе літакового типу «Ланцет» – приклад його процесу захоплення цілі наведено на рис. 1.6.


**Рисунок 1.6** – «Ланцет» за візуальними ознаками наводиться на гармату ЗСУ

Незважаючи на високу ефективність ударних БпЛА, обладнаних подібною системою автонаведення, в цієї технології є суттєвий недолік, що полягає в недостатньо високій швидкості реакції на зміщення захопленої цілі. Якщо така ціль несподівано зникає з поля зору, апарат без корекцій з боку пілота залишається на тому ж курсі й зазвичай промахується. Окрім цього,

роздільна здатність його камери та архітектура нейронної мережі розпізнавання об’єктів не розраховані на виокремлення дрібних деталей, тому може спрацювати на фальшивий макет техніки або змусити БпЛА врізатись у заздалегідь розтягнуту сітку, провокуючи детонацію бойової частини.

Окрім зазначеної технології, існує подібна до неї, що застосовується в розвідувальних БпЛА й дозволяє їм в автоматичному режимі супроводжувати рухому ціль або зависати над нерухомою в умовах відсутності зв’язку з оператором.

## 1.4 Причини вибору технології згорткових нейронних мереж для вирішення поставленої задачі

Як було зазначено наприкінці розділу 1.1, процес розпізнавання об’єктів визначених класів може бути автоматизовано програмними засобами на кшталт нейронних мереж. Цей вибір ґрунтується на таких причинах:

- наявність великої кількості фото- та відеоматеріалів з ворожими військовими об’єктами, що придатні для тренування нейронної мережі;
- порівняно низька складність реалізації програми за рахунок автоматичного навчання й виокремлення характерних візуальних ознак об’єктів різних типів;
- можливість постійного донавчання моделі на нових зображеннях об’єктів для її адаптації до змінних умов довкілля та засобів маскування противника;
- простота масштабування при додаванні нового типу об’єктів;
- можливість обрати архітектуру, що не вимагатиме значних обчислювальних потужностей під час використання після завершення тренування.

### 1.5 Постановка задачі

Завданнями на дипломну роботу є:

- узагальнення інформації про будову та принципи, що лежать в основі роботи згорткової нейронної мережі на кожному з етапів обробки зображення;
- дослідження методів обробки й розширення датасету;
- дослідження двох моделей, що базуються на згортковій нейронній мережі й часто використовуються в задачах розпізнавання й класифікації декількох об’єктів на зображенні – RetinaNet та YOLOv8;
- реалізація програмного застосунку, що визначає клас військових об’єктів на наданому аерофотознімку.

### Висновки до розділу 1

Своєчасне виявлення та ураження живої сили й техніки ворога є ключем до успіху у війні. Автоматизація процедури аналізу даних з розвідувального дрона здатна кратно прискорити потік стратегічно важливої інформації від підрозділів розвідки до центру прийняття рішень. В умовах повномасштабного вторгнення вирішення цієї задачі набуває критичного значення.

Для розуміння специфіки проблеми досліджено й розбито на класи поширені об’єкти військового призначення, а саме капоніри, окопи, важка броньована техніка, вантажівки та цивільні авто, що можуть використовуватись військовими. Така класифікація дозволяє побудувати ефективну модель розпізнавання представлених об’єктів.

Окрім того, у розділі 1.1 пояснено причини вибору архітектури згорткових нейронних мереж, а також здійснено постановку мети дипломної роботи та її визначення її завдань.

# РОЗДІЛ 2 ДОСЛІДЖЕННЯ ОСНОВНИХ СТРУКТУРНИХ ЕЛЕМЕНТІВ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ ТА ПРОЦЕСУ ЇХ НАВЧАННЯ

З технологічним прогресом стрімко зростає кількість і важливість задач, спрямованих на розпізнавання й класифікацію об’єктів. Такі задачі є часто унікальними за своїм спрямуванням, і єдине, що їх об’єднує – наявність великої кількості класифікованих візуальних даних. Очевидно, що розробка окремих рішень під кожну проблему подібного масштабу потребуватиме складного процесу ручного виокремлення характерних ознак об’єктів, використання масивного математичного апарату та неймовірну кількість часу. Окрім того, масштабування такого програмного продукту для розпізнавання додаткового класу об’єктів або незначна візуальна зміна цього об’єкту будуть дуже ресурсовитратними задачами. Закономірно виникла потреба винайти спосіб автоматизувати цей процес, змусити програму самостійно підлаштовуватись до своєї задачі.

Базуючись на досягненнях попередників – персептроні Ф. Розенблатта та багатошаровому персептроні Дж. Гінтона, Д. Румельхарта, Р. Вільямса, – а також надихнувшись тогочасними відкриттями у сфері нейробіології щодо будови зорової кори людини, Ян ЛеКун з колегами у 1980-х формує поняття про згорткові нейронні мережі (convolutional neural networks, CNN), а вже в 1990-х успішно застосовує їх в технології LeNet-5 [1]. Архітектура була спрямована на розпізнавання рукописних цифр та застосовувалась для автоматизації обробки чеків у банках. З того часу фундаментальна будова згорткових нейромереж не змінилась: ключовими складовими залишаються згорткові (convolution), підвибіркові (pooling) та повнозв’язні (fully connected) шари. Ці та інші елементи CNN буде описано в цьому розділі.

## 2.1 Загальна будова згорткової нейронної мережі

Оскільки згорткові нейронні мережі використовуються здебільшого для обробки фото- та відеоматеріалів, на вхід дана мережа приймає піксельне зображення, виражене як тривимірний масив виду (висота зображення, ширина зображення, кількість каналів зображення). Таким чином, наприклад, кольорове зображення формату JPEG з роздільною здатністю $600 \times 400$ буде записано як масив $600 \times 400 \times 3$, де кожна двовимірна матриця $600 \times 400$ відповідає інтенсивності одного з кольорів (червоний, зелений, синій) для кожного пікселя. На рис. 2.1 проілюстровано розбиття зображення на 3 кольорові канали та конвертація їхніх значень у чорно-білі зображення.

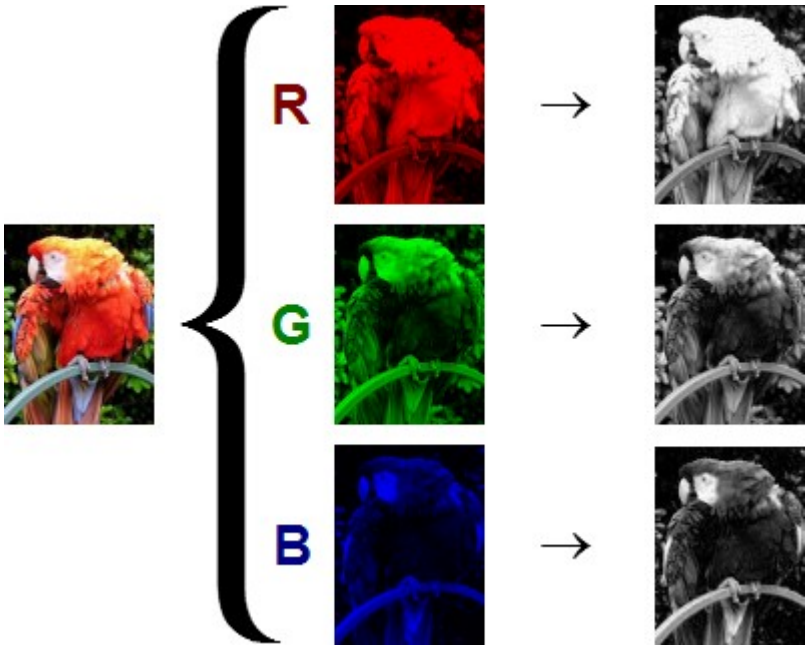



**Рисунок 2.1** – Розбиття зображення на кольорові канали та конвертація їх значень у чорно-білі моноканальні зображення[1]

[1]Джерело зображення:
https://upload.wikimedia.org/wikipedia/commons/5/56/RGB_channels_separation.png?20110219015028

Основною рисою і фундаментальним процесом всередині згорткових нейронних мереж є власне згортка (рис. 2.2, Convolution). Вона полягає в застосуванні простої матричної операції між ядром згортки (kernel) та невеликою обраною ділянкою зображення. Ця операція повторюється через “прохід” ядра по кожній ділянці зображення відповідного розміру. З отриманих результатів формується новий масив – карта ознак (feature map), що містить лінійні комбінації вхідних значень.

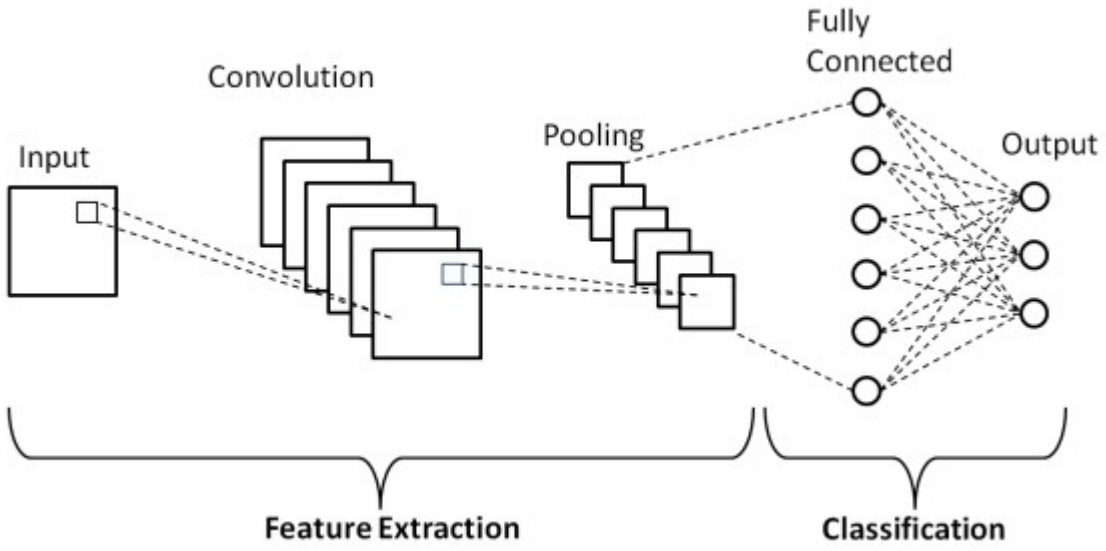



**Рисунок 2.2** – Схема основних структурних елементів CNN[2]

Задля внесення нелінійності в модель та уможливлення її навчання на складних патернах в даних отримана карта значень передається до функції активації (activation function). На цьому етапі до кожної комірки масиву застосовується деяка заздалегідь визначена нелінійна математична операція.

Наступним кроком обробки зображення є операція підвибірки (pooling). Метою застосування цього шару (рис. 2.2, Pooling) є кратне зменшення розмірів масиву, що сприяє зниженню обчислювальної складності моделі, узагальненню та вилученню найбільш значущих ознак, отриманих після згортки, та запобігає

[2]Джерело зображення:
https://www.researchgate.net/publication/336805909/figure/fig1/AS:817888827023360@1572011300751/Schematic-diagram-of-a-basic-convolutional-neural-network-CNN-architecture-26.ppm

перенавчанню.

Описані 3 етапи можемо об’єднати в процес виокремлення ознак (feature extraction). Він може повторюватись декілька разів залежно від розміру вхідного зображення та специфіки задачі, потенційно з різними функціями активації, способами згортки та методами підвибірки.

Після необхідної кількості згорток і вибірок отримано тривимірний масив виду $(x, y, n)$, де:

- $x, y$ – роздільна здатність карт ознак після всіх операцій згортки та підвибірки, що залежить від кількості відповідних шарів, розміру фільтрів (kernel size), кроку зсуву (stride length), типу заповнення (padding) та кратності вікна підвибірки;
- $n$ – кількість карт ознак, що залежить від кількості фільтрів на останньому згортковому шарі.

Наступним кроком є операція згладжування (flattening). Її задачею є конвертація тривимірного масиву в одновимірний вектор, що буде прийнятий повнозв’язними шарами. Таким чином, наприклад, з масиву карт ознак розміром $32 \times 32$ у кількості 8 штук отримуємо вектор з 8192 елементами. Це перетворення допомагає зменшити необхідну для обробки кількість нейронів у повнозв’язних шарах, а також дозволяє їм аналізувати взаємозв’язки між різними рівнями абстракції (картами ознак, отриманих від різних згорткових шарів).

Завершальним етапом обробки зображення є повнозв’язні шари (рис. 2.2, Fully-connected). Метою цих шарів є надання кінцевої відповіді на питання класифікації або регресії – визначення наявності, класу об’єкта або його розташування. Їхня основна риса полягає в тому, що кожен нейрон шару пов’язаний з кожним нейроном з попереднього шару. Він приймає всі їхні сигнали, виконує лінійну операцію зваженого сумування цих сигналів з відповідними вагами та передає отримане значення до нелінійної функції активації, а звідти – до нейронів наступного повнозв’язного шару або ж як вихідний результат роботи мережі [2, 3].

Розглянемо ці та інші елементи й аспекти будови згорткової нейронної мережі детальніше.

## 2.2 Основні параметри згорткового шару

Операція згортки (рис. 2.3) – застосування ковзаючого фільтру або ядра (kernel) розміру $n \times n$ поступово на кожному каналі зображення – має такі кроки:

1. Обрати ділянку каналу розміром $n \times n$.
2. Помножити елементи матриці ядра на відповідні елементи матриці зображення – таким чином, отримано $n^2$ добутків.
3. Додати добутки – отримано значення ядра для цього каналу.
4. Повторити кроки 1-3 для кожного каналу, не змінюючи ділянку.
5. Додати значення ядра для всіх каналів – отримано значення фільтру відповідної ділянки розміром $n \times n$.
6. Записати це значення до відповідної комірки матриці цього фільтра.
7. Перейти до наступної ділянки зображення відповідно до довжини зсуву.

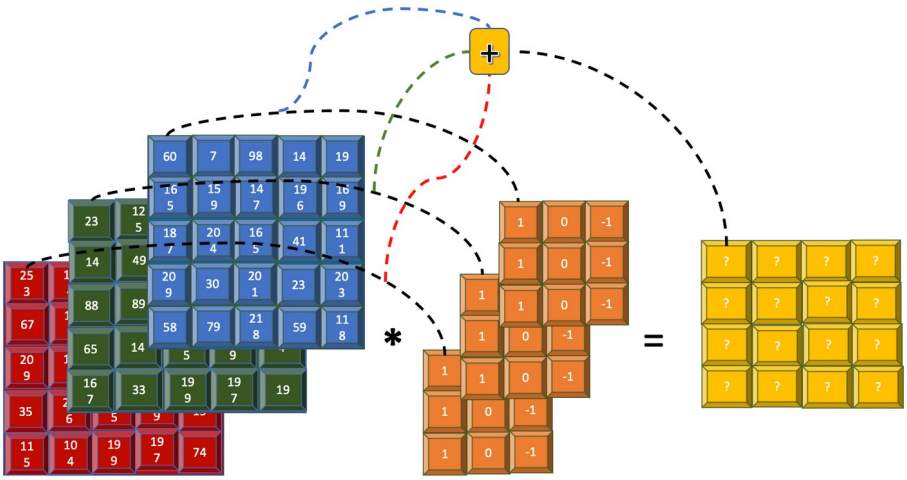


**Рисунок 2.3** – Схема операції згортки з застосуванням одного однакового ядра розмірністю $3 \times 3$ до кожного з каналів вхідного зображення[3]

### *2.2.1 Конфігурація ядра згортки*

Розмір та значення комірок матриці мають значний вплив на точність та швидкість роботи моделі, а також на виокремлення деталей різного роду. Загалом діють такі закономірності:

- фільтри великого розміру ($5 \times 5$, $7 \times 7$) охоплюють більшу область зображення, що дозволяє моделі краще виявляти більш глобальні ознаки й взаємозв’язки, а також виділяти складні або великі структури;

  тим не менш, використання таких фільтрів знижує чутливість моделі до дрібніших деталей;
- малі фільтри ($3 \times 3$) краще зосереджуються на дрібних деталях та локальних ознаках, а використання кількох шарів з малими фільтрами може забезпечити високу точність моделі;
- зі збільшенням розміру фільтра зменшується кількість операцій,

[3]Джерело зображення: https://developer.ibm.com/developer/default/articles/introduction-to-convolutional-neural-networks/images/RGB_Convolutions.png

необхідних для покриття зображення, але збільшується кількість параметрів, які потрібно навчити – іншими словами, що більший фільтр, то довший час потребує модель для навчання, але тим швидше відбувається обробка зображення.

В сучасних моделях часто використовують комплексний підхід – поєднання фільтрів кількох різних розмірів на різних етапах обробки даних або паралельно для ефективного виокремлення деталей різного масштабу [8].

Для визначення різних патернів існують різні матриці ядер. На рис. 2.4 представлено ядра (оператори) типів (a) Roberts Cross, (b) Sobel, (c) Prewitt, що використовуються для визначення меж об’єктів. Вони всі мають X- та Y-складову, що застосовуються окремо як різні фільтри. В результаті отримуємо дві карти ознак: $G_X$ відповідає вертикальним межам об’єктів відповідно до горизонтальних градієнтів, $G_Y$ – горизонтальним межам відповідно до вертикальних градієнтів. Приклади застосування даних операторів зображено на рис. 2.5.

| | |
|---|---|
| +1 | 0 |
| 0 | -1 |

Gx

| | |
|---|---|
| 0 | +1 |
| -1 | 0 |

Gy

(a)

| | | |
|---|---|---|
| -1 | 0 | +1 |
| -2 | 0 | +2 |
| -1 | 0 | +1 |

Gx

| | | |
|---|---|---|
| +1 | +2 | +1 |
| 0 | 0 | 0 |
| -1 | -2 | -1 |

Gy

(b)

| | | |
|---|---|---|
| -1 | 0 | +1 |
| -1 | 0 | +1 |
| -1 | 0 | +1 |

Gx

| | | |
|---|---|---|
| +1 | +1 | +1 |
| 0 | 0 | 0 |
| -1 | -1 | -1 |

Gy

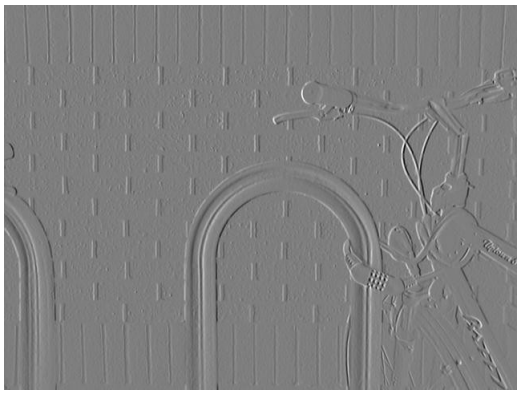
(c)

**Рисунок 2.4** – Матриці операторів

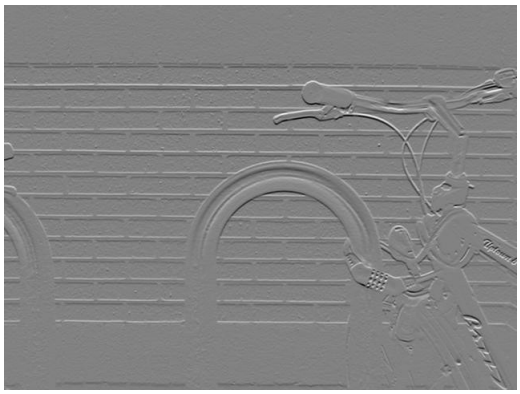
( a – Roberts Cross; b – Sobel; c – Prewitt[4])

(a)

(b)



(c)



(d)

[4]Джерела зображень:

(a) http://lab.must.or.kr/GetFile.aspx?File=/DIP/edge/roberts_operator.png

(b) http://lab.must.or.kr/GetFile.aspx?File=/DIP/edge/sobel_operator.png

(c) http://lab.must.or.kr/GetFile.aspx?File=/DIP/edge/prewitt_operator.png

(e)

(f)

**Рисунок 2.5** – Результати застосування різних операторів до зображення (a – оригінальне зображення; b – оператор Sobel; c – нормалізований горизонтальний градієнт $G_X$ оператора Sobel; d – нормалізований вертикальний градієнт $G_Y$ оператора Sobel; e – оператор Roberts Cross; f – оператор Prewitt [5])

### *2.2.2 Крок зсуву та заповнення*

Зсув (stride) – пересування до нової ділянки зображення вікна фільтру після завершення операції згортки на попередній. Відповідно крок зсуву визначає, на скільки пікселів вправо або вниз зсунеться ядро. В цілому його обирають з тих самих міркувань, що й розмір фільтру: що більший крок, то швидше навчання й менший час обробки зображень, але й тим більша втрата точності. Часто використовують комбінований підхід: на початкових шарах, де важливо виокремити максимум деталей, обирають мінімальний зсув (1 піксель), в той час як ближче до повнозв’язних шарів, коли важливішою стає оптимізація обчислень моделі та зменшення карти ознак, крок збільшують до 2 пікселів.

[5]Джерела зображень:

(a) https://upload.wikimedia.org/wikipedia/commons/3/3f/Bikesgray.jpg

(b) https://upload.wikimedia.org/wikipedia/commons/2/24/Bikesgray_sobel.JPG

(c) https://upload.wikimedia.org/wikipedia/commons/6/60/Bikesgraygv.jpg

(d) https://upload.wikimedia.org/wikipedia/commons/8/8a/Bikesgraygh.jpg

(e) https://upload.wikimedia.org/wikipedia/commons/6/62/Bikesgray_roberts.JPG

(f) https://upload.wikimedia.org/wikipedia/commons/3/3e/Bikesgray_prewitt.JPG

Заповнення (padding) – операція розширення вхідного зображення по краях додатковими значеннями для того, щоб після згортки розміри карти ознак відповідали розмірам початкового зображення або були скорочені у передбачуваний спосіб. Це також дозволяє уникнути втрати інформації, що знаходиться близько до меж зображення, оскільки ядро згортки може обробити відповідну область без спотворень. Використовують такі методи заповнення:

- нульове (zero padding) – додаткові комірки навколо зображення заповнюються нулями (рис. 2.6);
- відбивання (reflect padding) – значення пікселів на краях зображення дублюються в зворотному порядку;
- повторення (replicate padding) – значення крайніх пікселів дублюються.

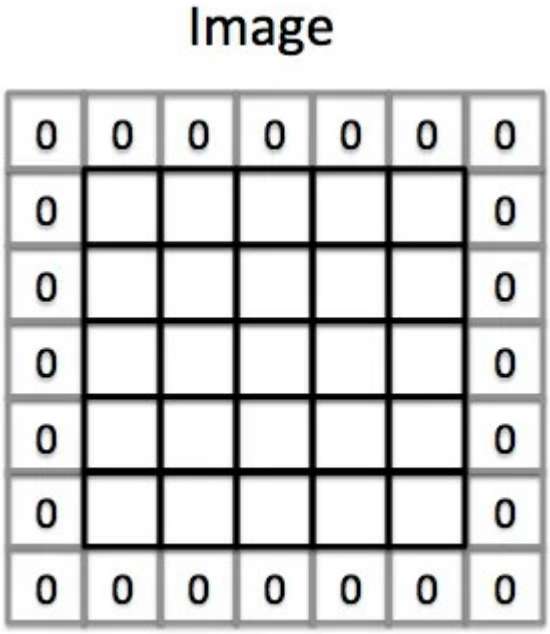



**Рисунок 2.6** – Нульове заповнення[6]

## 2.3 Функція активації

[6]Джерело зображення: https://miro.medium.com/v2/resize:fit:640/format:webp/0*edMCGFv8acvCx8oP.png

Основною метою застосування функцій активації до шарів нейронної мережі є внесення нелінійності в процес обробки вхідних даних. Без цієї операції всі трансформації всередині моделі зводилися б до лінійного перетворення, що має обмежену ефективність у задачах виявлення складних закономірностей та деталей з високим рівнем абстракції.

У згорткових шарах функцію активації використовують для поелементного перетворення карти ознак у карту активації. У повнозв’язних шарах функція застосовується окремо до виходу кожного нейрона. На виході останнього шару згорткової мережі функція активації дозволяє перетворити результат роботи моделі на дискретне вирішення задачі класифікації.

Виділяють такі поширені функції активації [4, 5]:

- ReLU – найбільш розповсюджена й проста функція, виражається наступною формулою:

$$f(x)=max(0,x)$$

Основною її перевагою є ненасичення активації, що дозволяє запобігти проблемі затухаючих градієнтів (vanishing gradient problem), яка полягає в тому, що при тренуванні перші шари отримують незначні або й нульові зміни до ваг нейронів, призводячи до сильного сповільнення тренування моделі та зниження її ефективності.

Основний недолік – проблема “dying ReLU”, зумовлена нулем у формулі функції. Через нього нейрони, що не були активовані на ранніх шарах мережі (видали сигнал “0”), відповідно не передають корисного навантаження й на наступні. В результаті може статись, що в процесі опрацювання зображення значна кількість нейронів може не запуститись зовсім.

- Leaky ReLU – покращена версія ReLU з однією ключовою зміною, має формулу:

$$f(x)=max(\alpha x,x)$$

Завдяки заміні 0 на $\alpha x$ ця функція запобігає проблемі “dying ReLU”,

але дозволяє близькі до 0 негативні вихідні значення при негативних вхідних. Гіперпараметр $\alpha$ визначається експериментально відповідно до поставленої задачі, стандартне значення $\alpha = 0.01$ .

- Sigmoid – функція з вихідними значеннями в діапазоні (0;1). Виражається формулою:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Корисна для задач бінарної класифікації, але окрім як в останніх шарах моделей використовується нечасто через проблему затухаючих градієнтів, що зумовлена прямуючою до 0 похідною при великих значеннях$x$.

- Tanh – гіперболічний тангенс, має вихідні значення в діапазоні (-1;1). Виражається формулою:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Подібно до Sigmoid має проблему з затухаючими градієнтами, тому широко не використовується.

- ELU – заснована на ReLU кусково задана функція:

$$f(x) = x, x > 0$$
$$f(x) = \alpha(e^{x} - 1), x \leq 0$$

Вирішує проблему “dying ReLU” завдяки обробці негативних вхідних значень, а також спрямовує вихідні значення ближче до 0, що сприяє пришвидшенню навчання. Значення $-\alpha$ є асимптотою, до якої прямують негативні вихідні значення.

- Swish – порівняно нова функція, заснована на Sigmoid. Виражається формулою:

$$f(x)=\frac{x}{1+e^{-\beta x}}$$

відсутність проблеми “dying ReLU” та плавніша активація, що може покращити здатність мережі до навчання на складніших патернах.

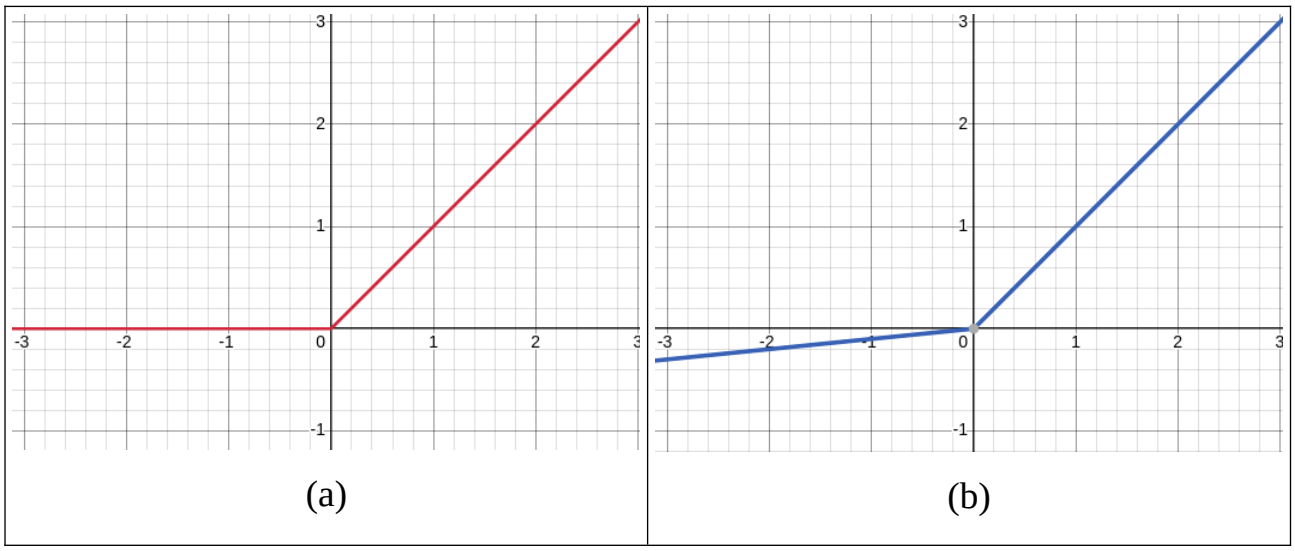
- Softmax – використовується на виході моделі для задач багатокласової класифікації. Виражається формулою:

$$f(x_i)=\frac{e^{x_i}}{\sum_j e^{x_j}},$$

де $x_i$ – вихідне значення нейрона і.

Ця функція перетворює вектор виходів на ймовірності, сума яких дорівнює 1, тому вони легко інтерпретуються як ймовірності належності до певного класу.

Графіки поширених функцій активації зображено на рис. 2.7.




(a) (b)

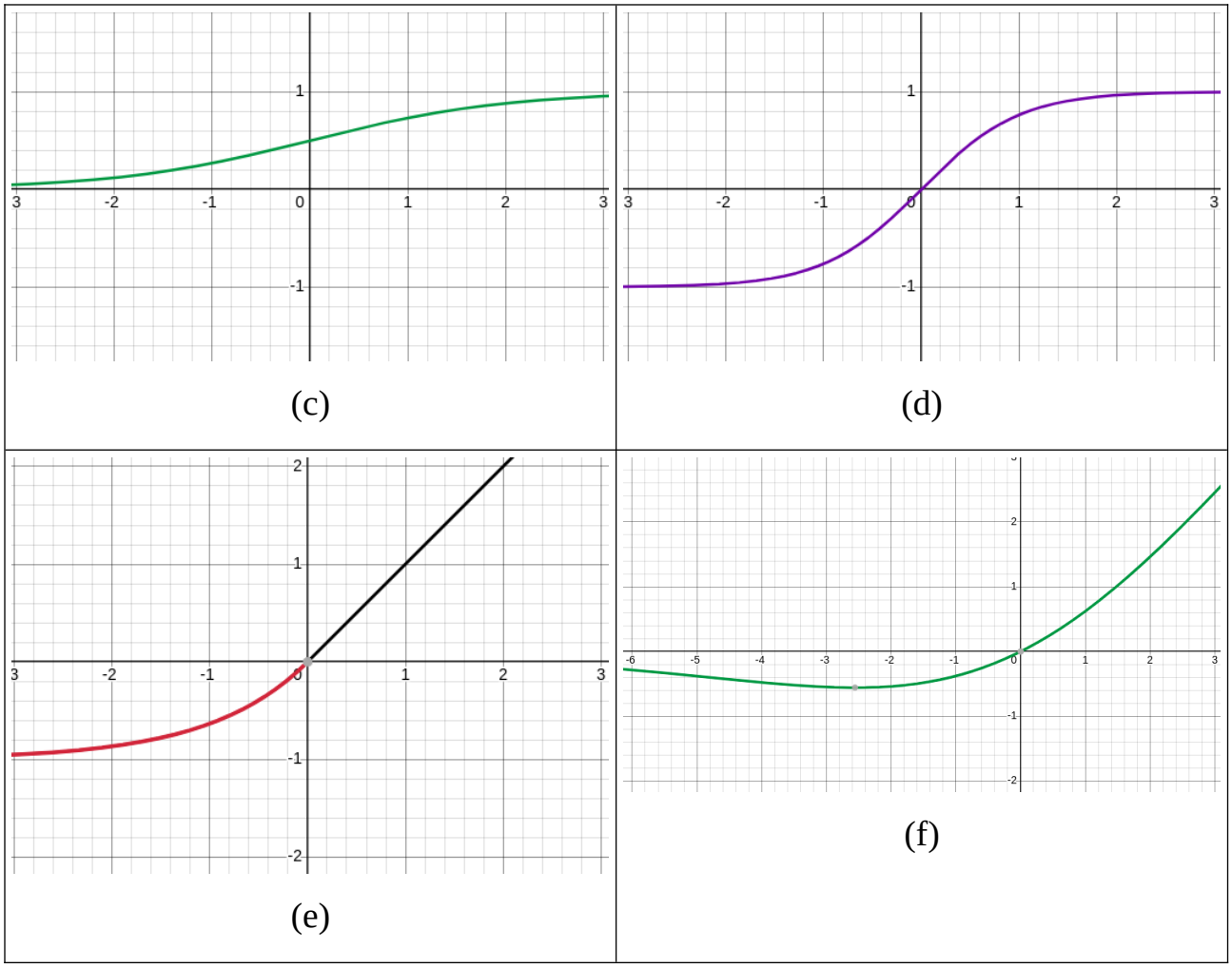



**Рисунок 2.7** – Графіки поширених функцій активації (a – ReLU, b – Leaky ReLU ($\alpha = 0.1$); c – Sigmoid; d Tanh; e – ELU ($\alpha = 1$); f – Swish ($\beta = 0.5$))

## 2.4 Операція підвибірки

Математично цей процес полягає в застосуванні операції агрегації до отриманої матриці карти активації. Іншими словами, прохід значень матриці вікном заданого розміру, що деяким чином трансформує виділену ділянку матриці в єдине значення, яке зберігається до вихідної матриці зменшеної розмірності (рис. 2.8). Ця операція, окрім іншого, за рахунок зменшення

кількості пікселів, що потребують обробки, дозволяє поступово знизити обчислювальну складність моделі, особливо на пізніх її шарах [16].

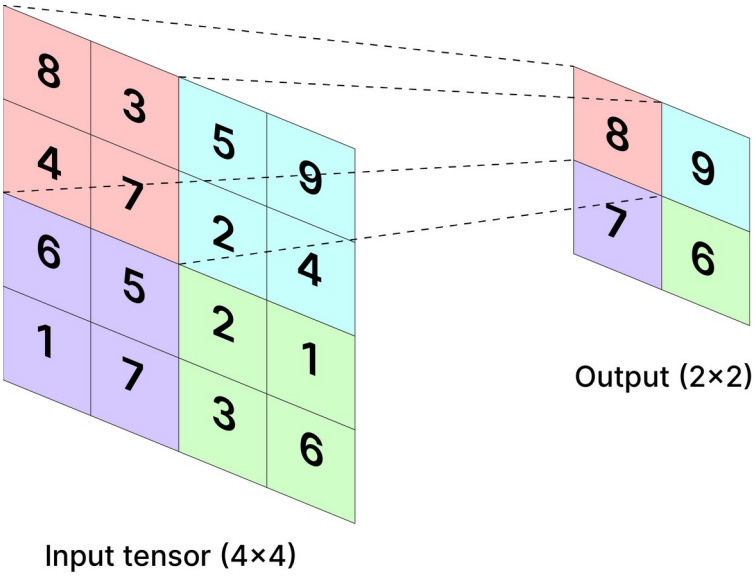



**Рисунок 2.8** – Схема застосування операції max pooling з вікном $2 \times 2$ та кроком зсуву 2 до матриці розмірністю $4 \times 4$ [7]

Найбільш поширені методи підвибірки [9]:

- max pooling – виокремлює найбільше значення серед виділених;
  - добре зберігає ключові ознаки об’єктів;
  - допомагає знизити перенавчання (overfitting) моделі – явище, за якого мережа надто добре навчається на тренувальному датасеті, але показує погані результати на тестових і реальних даних;
  - може призвести до втрати даних про просторову орієнтацію об’єктів;
- average pooling – середнє значення з виділених;
  - згладжує зображення, знижує кількість шуму та дозволяє моделі узагальнювати деталі краще;

[7] Джерело зображення: https://iq.opengenus.org/content/images/2023/06/Max-Pooling.png

- зберігає контекстну інформацію;
- підвищує вірогідність втрати дрібних деталей об’єктів;

- global max pooling – найбільше значення з карти ознак;
  - завдяки значному зменшенню розмірності вхідного масиву може бути корисним при використанні на повнозв’язних шарах на завершальному етапі обробки зображення;
  - допомагає виділити найбільш значущі ознаки карти ознак;
  - втрачає майже всю інформацію про просторову орієнтацію об’єктів;
- global average pooling – середнє значення з усієї карти ознак;
  - так само значно зменшує розмірність вхідного масиву;
  - збереження контекстної інформації завдяки врахуванню всієї карти ознак;
  - втрата інформації про деталі.

## 2.5 Регуляризація

Регуляризацією називають процес, що має на меті знизити перенавчання моделі й підвищити її здатність до узагальнення новітніх даних. Деякі поширені методи регуляризації застосовують зміни або обнулення до ваг або карт активації обраних деяким чином нейронів, інші вносять корективи до вхідних даних або процесу навчання моделі в цілому. Зазвичай при побудові мереж, особливо глибоких, використовують декілька найбільш ефективних відповідно до поставленої задачі операцій [15]. Серед поширених методів варто згадати зокрема такі:

- L1 (Lasso) та L2 (Ridge) – великі значення вагів штрафуються шляхом додавання регуляризуючого виразу до функції втрат (loss function), що обчислюється за формулами 2.1 для L1 та 2.2 для L2.
    - L1-регуляризація досягає менших значень штрафного виразу за рахунок обнулення деяких ваг, що сприяє визначенню найбільш впливових ознак та нижчій обчислювальній складності моделі;
    - L2-регуляризація для досягнення мінімуму рівномірно зменшує значення ваг, підтримуючи їх ненульовими, що корисно у випадках, коли кожна ознака об’єкта є важливою, та сприяє більшій стабільності навчання моделі порівняно з методом L1; в багатьох архітектурах L1 та L2 застосовують одночасно – такий метод регуляризації отримав назву ElasticNet та виражається формулою 2.3.

$$L1 = \lambda_1 \sum_{i=1}^{n} |w_i|, \tag{2.1}$$

$$L2 = \lambda_2 \sum_{i=1}^{n} w_i^2, \tag{2.2}$$

$$ElasticNet = \alpha \lambda_1 \sum_{i=1}^{n} |w_i| + (1-\alpha) \lambda_2 \sum_{i=1}^{n} w_i^2, \tag{2.3}$$

де $\lambda_1$ – коефіцієнт регуляризації L1;

$\lambda_2$ – коефіцієнт регуляризації L2;

$w_i$ – ваги моделі;

$\alpha$ – співвідношення між L1 та L2 регуляризаціями у методі ElasticNet;

- випадкове вилучення, викид (dropout, рис. 2.9) – відключення на кожній ітерації навчання деякої кількості обраних випадковим чином нейронів разом з їхніми вхідними та вихідними зв’язками,

обнулення їхніх активацій та градієнтів ваг. Це змушує мережу вчитись з меншими ресурсами, дозволяючи їй в результаті краще узагальнювати ознаки в нових даних, а також урізноманітнює спеціалізацію нейронів;

- середня кількість відключених нейронів залежить від встановленого значення гіперпараметра. На етапі тестування викид не застосовується, натомість всі виходи нейронів масштабуються відповідно до коефіцієнта вилучення;

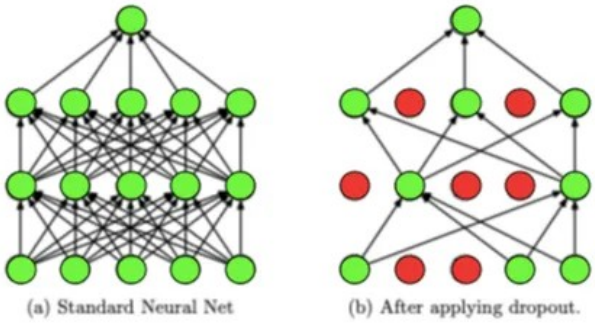



**Рисунок 2.9** – порівняння структури шарів нейромережі до (a) та після (b) застосування операції викиду (відключення) нейронів та їхніх зв’язків[8]

- випадкове відключення з’єднань (dropconnect) – те саме, що й викиди, але відключення замість нейронів застосовується до їхніх ваг (з’єднань). В цьому стані вони не оновлюються при градієнтному спуску та не передають сигнал на наступні шари;
  - під час тестування ваги масштабуються на встановлений коефіцієнт;
  - має більші обчислювальні витрати, ніж dropout, але є більш гнучким, оскільки не всі з’єднання нейрона можуть бути відключені та його сигнал може частково передаватись на наступні шари моделі;

[8] Джерело зображення: https://miro.medium.com/v2/resize:fit:640/format:webp/0*EY8R7nS10y5kQzOx

- шум ваг (weight noise) – додавання невеликого випадкового значення до ваг під час тренування;
  - зазвичай використовують шум, розподілений за нормальним розподілом з середнім значенням 0 та певною стандартною девіацією;
  - додавання шуму дозволяє зокрема зробити мережу більш стійкою до незначних змін у вхідних даних, а також стабілізувати процес навчання за рахунок подолання локальних мінімумів під час процесу оптимізації;
  - цей метод потребує тонкого налаштування стандартного відхилення для оптимального балансу між стабілізацією навчання й збереженням корисної інформації;
- рання зупинка (early stopping) – зупинка процесу тренування, коли продуктивність моделі на валідаційному датасеті починає погіршуватись або не покращується протягом кількох епох підряд;
  - збалансоване поєднання запобігання недонавчанню (underfitting) та перенавчанню, але висока залежність ефективності навчання від якості валідаційного датасету;
  - за деяких умов можлива зупинка тренування моделі, що ще не досягла свого повного потенціалу;
- штучне розширення тренувального датасету (data augmentation) – створення нових зразків вхідних даних для тренування моделі шляхом застосування до зображення різноманітних операцій, наприклад, зсуву, відображення, масштабування, розмиття, додавання шуму тощо;
  - цей метод дозволяє значно підвищити ефективність навчання моделі, особливо якщо початковий обсяг тренувальних даних невеликий або вони надто одноманітні;
  - застосовані трансформації мають бути налаштовані таким чином, аби зберігати сутність оригінальних даних і при цьому

вносити деяку різноманітність в датасет.

## 2.6 Нормалізація

Операція нормалізації застосовується беспосередньо до масиву даних і полягає в їх лінійному масштабуванні таким чином, аби вони мали деякі визначені статистичні характеристики. Ця математична операція зазвичай використовується після кожного шару згортки та дозволяє в першу чергу прискорити процес навчання моделі й запобігти проблемі затухаючих або вибухаючих градієнтів. Основні методи нормалізації в згорткових нейронних мережах:

- batch normalization – нормалізація вихідного тензору кожного шару на основі середнього значення та варіації поточного батчу (вибірки заданого об’єму з датасету);
- layer normalization – в умовах, коли розмір батчу малий, можливе застосування операції нормалізації на основі карти активації поточного шару незалежно від батчу;
- instance normalization – нормалізація кожного екземпляру окремо, тобто середнє значення й значення варіації окремо взятого вихідного зображення не впливають на відповідні значення інших;
    - часто використовується в задачах стилізації зображень, оскільки мінімально спотворює унікальні для зображення ознаки;
- group normalization – в умовах великої кількості даних доцільно розділити канали на групи й застосовувати нормалізацію всередині цих груп;
    - забезпечує стабільну нормалізацію незалежно від розміру батчу, але потребує визначення оптимального розміру груп.

Приклад застосування нормалізації до вхідного зображення показано на рис. 2.10.

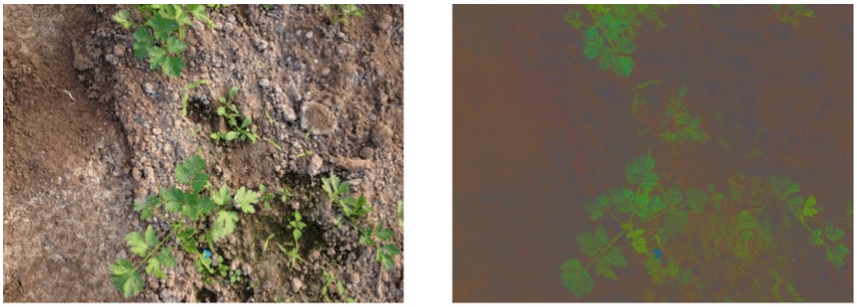


**Рисунок 2.10** – Оригінальне й нормалізоване кольорове зображення[9]

## 2.7 Ключові аспекти тренування згорткових нейронних мереж

В якості основного методу тренування моделі буде розглянуто метод зворотного поширення помилки (backpropagation) як такий, що найчастіше зустрічається саме в згорткових нейронних мережах. Він складається з 3 кроків.

- Прямий прохід (Forward Pass): вхідні дані проходять мережу шар за шаром; на кожному шарі обчислюється карта активації відповідно до ваг і функції активації.
- Обчислення загальної помилки: вихідне значення останнього шару моделі порівнюється з істинним значенням за допомогою функції втрат.
- Зворотний прохід (Backward Pass): за правилом ланцюга з кінцевого шару в напрямку початкового обчислюються градієнти функції втрат по відношенню до ваг між поточними шарами; для мінімізації функції втрат ваги мережі зміщують значення у напрямку, протилежному до відповідних отриманих градієнтів.

В загальному випадку градієнт функції втрат $C$ відносно ваги з’єднання

[9] Джерело зображення: https://www.researchgate.net/profile/Eftim-Zdravevski/publication/319570220/figure/fig3/AS:667674988253184@1536197530026/Example-RGB-image-from-the-dataset-normalized-RGB-image-ExG-result-image-ExR.ppm

між нейроном $k$ шару $L-1$ та нейроном $j$ шару $L$ $w_{jk}^{L}$ обчислюється за такою формулою:

$$\frac{\partial C}{\partial w_{jk}^{L}} = \frac{\partial z_{j}^{L}}{\partial w_{jk}^{L}} \frac{\partial a_{j}^{L}}{\partial z_{j}^{L}} \frac{\partial C}{\partial a_{j}^{L}},$$

де $z_{j}^{L} = \ldots + w_{jk}^{L} a_{k}^{L-1} + \ldots$ – сума вхідних сигналів до нейрона $j$ шару $L$;

$a_{j}^{L}$ – вихідний (активований) сигнал нейрона $j$ шару $L$;

$C$ – значення функції втрат.

Формула нових ваг буде залежати від вибору оптимізатора моделі, деякі поширені алгоритми яких описані в розділі 2.7.2.

### *2.7.1 Функція втрат (Loss Function)*

Математичний інструмент, що під час тренування моделі використовується для оцінки якості її передбачень. В загальному випадку ця функція вимірює різницю між значеннями, передбаченими мережею, та фактичними з навчальних даних. Таким чином, основна задача моделі, що навчається, – мінімізувати значення власної функції втрат за рахунок підвищення точності результатів.

Значення функції втрат через метод зворотного поширення помилки (backpropagation) застосовуються для оновлення ваг мережі в кінці кожної ітерації навчання.

Для задач багатокласової класифікації часто використовують так звану категорійну функцію крос-ентропії (Categorical Cross-Entropy), що виражається формулою:

$$CategoricalCross-Entropy=-\sum_{i-1}^{n}\sum_{j=1}^{k}y_{ij}\log\left(\widehat{y}_{ij}\right),$$

де $n$ – кількість елементів у вибірці;

$k$ – кількість класів;

$y_{ij}$ – бінарний індикатор приналежності елемента $i$ до класу $j$;

$\widehat{y}_{ij}$ – передбачена моделлю ймовірність, що елемент $i$ належить класу $j$.

Значення $y_{ij}$ приймаються у вигляді one-hot encoded вектору, тобто такого одновимірного масиву, в якому лише комірка правильного класу для елемента $i$ має значення 1, а всі решта – 0. Відповідно функція матиме тим менше значення, що більшу ймовірність модель присвоїла правильному класу.

Окрім цього, представлена функція крос-ентропії завдяки експоненційній природі має просту похідну, визначену на всьому просторі значень, та високу чутливість до невеликих змін передбачених імовірностей, що сприяє ефективному та стабільному оновленню ваг на кожній ітерації.

### *2.7.2 Оптимізатор*

Оптимізаційними алгоритмами називають методи налаштування параметрів (ваг) нейромереж, що використовуються з метою мінімізації функції втрат. Зміна ваг обчислюється після кожної ітерації навчання моделі на основі визначених під час зворотного поширення помилки градієнтів. Серед найбільш поширених методів виділяють:

- SGD (Stochastic Gradient Descent, стохастичний градієнтний спуск – оновлює ваги на основі градієнтів, обчислених для кожного окремого елементу тренувального датасету. Формула оновленого

значення ваги:

$$\acute{w} = w - \eta \nabla L(w),$$

де $\eta$ – крок градієнтного спуску, змінна, що відповідає за швидкість навчання (learning rate);
$\nabla L(w)$ – градієнт функції втрат відносно поточної ваги;

- Momentum – алгоритм на основі SGD, що додає термін, який враховує попередні градієнти для пришвидшення сходження функції втрат у напрямку мінімуму та зменшення шумів градієнтів. Оновлена вага обчислюється за формулою:

$$v_t = \gamma v_{t-1} + \eta \nabla L(w)$$
$$\acute{w} = w - v_t,$$

де $v_{t-1}$ – інерція або імпульс (значення $v_t$ попередніх градієнтів);
$\gamma$ – коефіцієнт імпульсу;

- AdaGrad (Adaptive Gradient, адаптивний градієнт) – змінює швидкість навчання для кожного параметра окремо на основі частоти його оновлень. Добре працює як для рідкісних, так і для частих ваг, але може значно сповільнити навчання при великих значеннях градієнтів. Формула оновлення ваги:

$$\acute{w} = w - \frac{\eta}{\sqrt{G + \epsilon}} \nabla L(w),$$

де $G$ – діагональна матриця, що містить суму квадратів попередніх градієнтів по кожному параметру;
$\epsilon$ – невелике значення, введене для запобігання діленню на 0;

- Adam (Adaptive Moment Estimation, адаптивна оцінка моменту) – поєднує ідеї алгоритмів Momentum та AdaGrad, використовуючи адаптивні швидкості для кожного параметра і враховуючи моменти (“інерцію”) попередніх градієнтів першого й другого порядків. Поєднання сильних сторін двох інших методів робить цей метод найбільш універсальним інструментом оптимізації з усіх зазначених, але в той же час ускладнює його налаштування через значну кількість гіперпараметрів та громіздкість обчислень. Оновлення значень ваг відбувається за формулами:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L(w)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla L(w))^2$$

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

$$\acute{w} = w - \eta \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon},$$

де $m_t$ – оцінка середнього значення градієнтів;

$v_t$ – оцінка дисперсії градієнтів;

$\beta_1$ – гіперпараметр, що визначає швидкість згладжування $m_t$;

$\beta_2$ – гіперпараметр, що визначає швидкість згладжування $v_t$.

**Висновки до розділу 2**

В розділі 2 сформовано поняття про загальний алгоритм та методи роботи згорткової нейронної мережі, розглянуто її базові структурні елементи, їхній функціональний устрій та призначення, а також декілька поширених способів реалізації зокрема ядра згортки, шару підвибірки, функції активації та інших

складових. Окрім того, досліджено принцип роботи алгоритму навчання нейромережі, розглянуто основний для згорткових моделей його метод – зворотнє поширення помилки, – а також описано декілька поширених варіантів його імплементації з тими чи іншими покращеннями.

# РОЗДІЛ 3 ОГЛЯД ВИКОРИСТАНИХ МОДЕЛЕЙ, ОПИС ТА РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

## 3.1 Аналіз та розмітка вхідного набору даних

Для тренування та валідації реалізованих нейронних було використано набір зображень, що були отриманий в процесі виконання бойових завдань на східних напрямках ведення бойових дій, зокрема в районах м. Вовчанськ та м. Кремінна. Для аерофотозйомки використовувався дрон літакового типу «Valkirye» («Валькірія»), висота польоту близько 600 м, інтервал між знімками – 3 с. В результаті отримано більше 18000 фото роздільною здатністю $4608 \times 3456$ пікселів, що покривають ділянку місцевості близько $300 \times 400$ м та містять до 10 військових об’єктів. Незважаючи на те, що абсолютна більшість (~17000) цих знімків є порожніми, було прийнято рішення використати цей датасет, попередньо аугментувавши зображення з позначеними для навчання об’єктами. Цю процедуру буде розглянуто в підрозділі 3.2.

### *3.1.1 Розмітка датасету зображень з використанням CVAT*

CVAT (Computer Vision Annotation Tool) – це безкоштовний онлайн-застосунок з відкритим кодом, що дозволяє швидко вручну відмітити потрібні для навчання об’єкти рамками з зазначеними класами. Для задач цієї роботи використовувався здебільшого інструмент «Прямокутник», інтерфейс налаштування якого наведено на рис. 3.1. Повне вікно застосунку показано на рис. 3.2. На ньому з правого боку знаходиться список позначених об’єктів, кожному відповідає деякий обраний клас. В подальшому ці позначення конвертуються у файл формату XML, що зберігає інформацію про назву зображення, клас об’єкта та його координати. В цій роботі використано формат

запису анотацій PASCAL VOC v1.1, приклад файлу якого наведено на рис. 3.3.

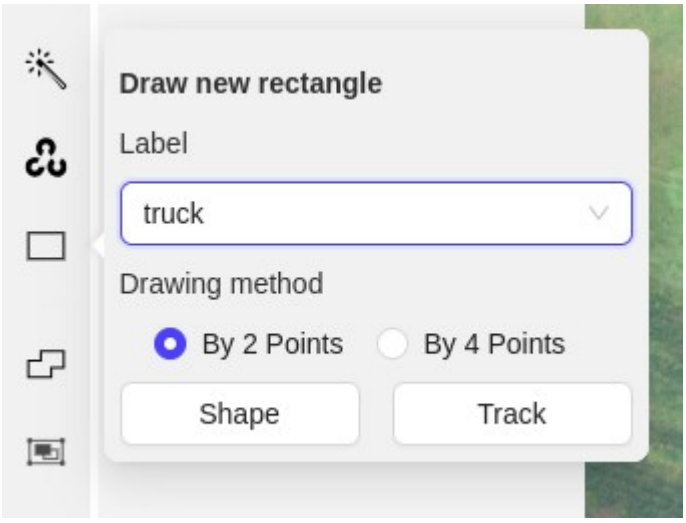



**Рисунок 3.1** – Інструмент «Прямокутник» з можливістю вибору класу об’єкта, методу позначення (за двома чи чотирма вершинами) та опціональним відслідковуванням об’єкта на наступних кадрах

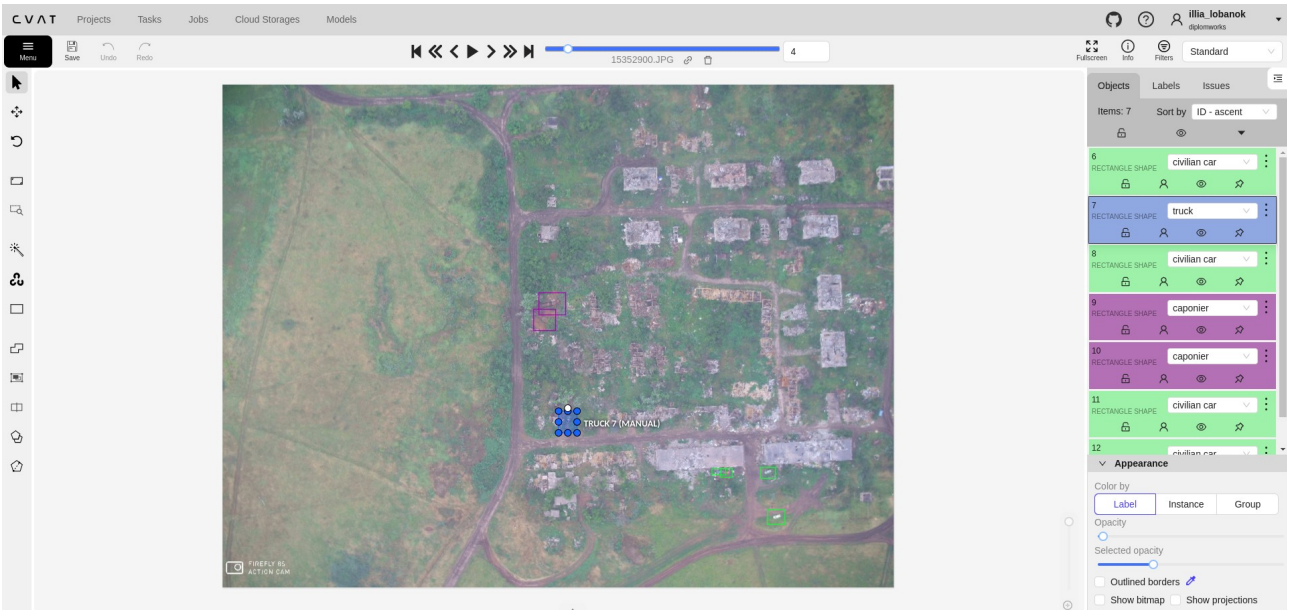



**Рисунок 3.2** – Інтерфейс онлайн-застосунку CVAT


**Рисунок 3.3** – Приклад запису даних про об’єкт класу капонір

Всього було промарковано близько 200 зображень, на яких вийшло анотувати більше 1000 об’єктів з нерівномірним розподілом за класами.

### 3.2 Аугментація датасету

Для ефективного навчання моделі рекомендується зберігати близький до рівномірного баланс між кількістю позитивних (таких, що мають шукані

об’єкти) та негативних (порожніх) зображень. Окрім того, на якість мережі в цілому позитивно впливає більша кількість якомога різноманітніших даних. З огляду на ці міркування було реалізовано алгоритм аугментації початкового датасету, завдяки якому його розмір було збільшено вдвічі. Приклад результату аугментації наведено на рис. 3.4, де (a) – ділянка оригінального зображення, (b) – трансформованого.

(a) (b)

**Рисунок 3.4** – Один і той самий окоп
(a – оригінальне зображення, b – аугментоване)

Розроблений алгоритм аугментації складається з кількох послідовних операцій, що можуть бути виконані у випадковому порядку та з різною ймовірністю. Наприклад, кожне зображення з вірогідністю 50% зазнає відзеркалення по горизонтальній або вертикальній осям, змінює контрастність в діапазоні від 75% до 150% і з вірогідністю 50% проходить через один з трьох методів розмиття. Окрім того, можливі також накладення шумів, виділення меж та поканальний викид пікселів. Сукупність цих операцій не тільки дозволяє збільшити об’єм датасету, але й привчити модель до неідеальних умов зйомки –

від легкого туману до намагань ворога замаскувати укріплення й техніку.

Для реалізації алгоритму аугментації було обрано бібліотеку imgaug. Такий вибір був спричинений, по-перше, порівняною легкістю її інтеграції в код та самодостатнім набором доступних методів, а по-друге, її здатністю обробляти зображення одночасно з анотаціями об’єктів на ньому. Таким чином, якщо в процесі аугментації зображення масштабується, відзеркалюється або повертається, аналогічна операція застосовується до відповідних рамок об’єктів. Це дозволяє не перейматись за втрату точності анотацій після аугментації датасету та значно спрощує організацію потоку даних від датасету до вхідного шару нейромережі.

### 3.3 Огляд архітектури моделі RetinaNet

RetinaNet – це алгоритм для виявлення об'єктів, вперше представлений у 2017 році в науковій статті "Focal Loss for Dense Object Detection" авторами Цун-І Лін, Пія Гоял, Росс Гіршик, Каїмін Хе та Пьотр Доллар. Ця архітектура стала важливим кроком вперед у галузі виявлення об'єктів зокрема завдяки введенню нової функції втрат – Focal Loss.

RetinaNet є одноетапним алгоритмом, тобто він безпосередньо передбачає обмежувальні рамки та ймовірності класів за один прохід зображення скрізь нейронну мережу. Цей підхід робить його швидшим і ефективнішим порівняно з двоетапними моделями, такими як Faster R-CNN.

RetinaNet складається з 3 основних компонентів (рис. 3.5):

- основа (backbone) – заснована на алгоритмі ResNet згорткова нейронна мережа, що використовується для екстракції ознак з вхідних зображень;
- шия (neck) – алгоритм FPN (Feature Pyramid Network), що завдяки обробці багаторівневих ознак дозволяє мережі виокремлювати риси

об’єктів різного масштабу одночасно. Особливістю цього алгоритму є те, що замість навчання кожного шару відображенням $H(x)$ дозволяється відображати лише різницю між $H(x)$ та вхідним зображенням. Це, в свою чергу, дозволяє до того ж вирішити проблему затухаючих градієнтів;

- голова (head) – приймає ознаки від FPN та передбачує клас об’єкта і його обмежувальну рамку.

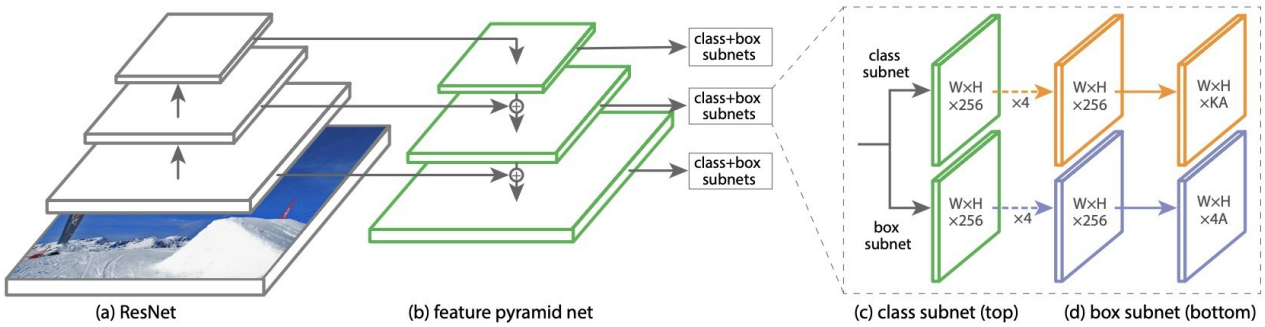



**Рисунок 3.5** – схематичне зображення архітектури моделі RetinaNet (a – основа, ResNet; b – шия, FPN; c – голова, підмережа класифікації; d – голова, підмережа визначення координат[10])

### *3.3.1 Focal Loss*

Focal Loss – це функція втрат, розроблена для вирішення проблеми дисбалансу класів у задачах виявлення об'єктів. Вона модифікує стандартну функцію втрат крос-ентропії, щоб зменшити вагу легких прикладів і збільшити вагу важких, зосереджуючи навчання на більш складних випадках, що дозволяє покращити загальну точність моделі.

Функція втрат Focal Loss визначається як:

[10]Джерело зображення: https://media.licdn.com/dms/image/D4E12AQHI1BsMx0-fTg/article-cover_image-shrink_720_1280/0/1675507996784?e=2147483647&v=beta&t=D1tJKa8_esyPNK4aE6wguv6bDbT3IHYlD6phidqnk8A

$$FL(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t),$$

де $p_t$ – ймовірність правильного класу;

$\alpha_t$ – ваговий коефіцієнт балансування класів;

$\gamma$ – параметр фокусування, що зменшує вагу легких прикладів.

Таким чином, коли приклад класифікується без ускладнень, значення виразу $(1 - p_t)^{\gamma}$ залишається невеликим, що зменшує функцію втрат для цього прикладу й не провокує сильних змін у відповідних вагах. Натомість для складних випадків значення цього виразу зростає, збільшуючи значенян функції втрат та пропагуючи більшу зміну у відповідні значення ваг під час їх коригування [13].

### *3.3.2 ResNet*

Модель ResNet (Residual Network) є важливою складовою архітектури RetinaNet і виконує роль її основної мережі (backbone) для екстракції ознак з вхідних зображень. ResNet була запропонована Хе Каїміном, Джан Сяньжуном, Реном Шаоціном та Суном Джіаном у 2015 році в статті "Deep Residual Learning for Image Recognition". Вона стала революційним кроком у глибокому навчанні завдяки використанню залишкових блоків (residual blocks), що допомагають вирішувати проблему затухання градієнта в дуже глибоких нейронних мережах. Їх ідея полягає в тому, аби дозволити моделі навчатись на різницях (residuals) між вхідним сигналом і бажаним виходом, а не фактичним виходом [7]. Кожен залишковий блок можна представити як:

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

де $x$ – вхідні дані,

$F(x,\{W_i\})$ – функція залишків, за якою модель навчається,

$y$ – вихідний сигнал.

### *3.3.3 Feature Pyramid Network*

FPN використовує ієрархічну структуру характеристик, що дозволяє моделі отримувати контекстну інформацію на різних рівнях абстракції. Цей процес відбувається в 3 етапи [12].

1. Низхідний потік – вхідне зображення проходить через кілька рівнів ResNet (C1, C2, C3, C4, C5), де витягуються характеристики з різними ступенями абстракції.
2. Висхідний потік – високорівневі характеристики з C5 передаються до нижчих рівнів через шари перевірки та злиття. Наприклад, характеристики з C5 об'єднуються з характеристиками з C4, створюючи P4, і так далі.
3. Злиття – кожен рівень висхідного потоку об'єднується з відповідним рівнем низхідного потоку, створюючи багатошарові представлення характеристик (P2, P3, P4, P5, P6).

Поєднання цих елементів забезпечує високу точність моделі RetinaNet навіть на незбалансованих датасетах, що в поєднанні зі значною швидкістю обробки зображень та універсальністю у виявленні ознак будь-якого масштабу робить цю архітектуру однією з найкращих одноетапних згорткових нейромереж. Слабкими сторонами цієї моделі є хіба що вимогливість до обчислювальних ресурсів на етапі тренування та велика кількість гіперпараметрів, що ускладнює її налаштування.

### 3.4 Огляд архітектури моделі YOLOv8

Алгоритм для виявлення об'єктів, вперше представлений у 2015 році в науковій статті Джозефом Редмоном, Сантошем Дівала, Россом Гіршиком і Алі Фархаді. Архітектура YOLO стала значною революцією в області виявлення об'єктів у реальному часі, перевершивши свого попередника – Region-based Convolutional Neural Network (R-CNN).

YOLO – це одноетапний алгоритм, який безпосередньо класифікує об'єкт за один прохід, використовуючи лише одну нейронну мережу для передбачення обмежувальних рамок і ймовірностей класів на повному зображенні. Сімейство моделей YOLO постійно розвивається та доповнюється як спеціалізованими, так і загально направленими моделями. Останньою наявною ланкою їх розвитку є YOLOv8.

Загалом ця модель має на меті максимізацію середньої точності (mAP – mean average precision), що дорівнює середньому відношенню кількості передбачених позитивних результатів до загальної кількості істинних позитивних результатів.

Як і RetinaNet, YOLOv8 складається з основних 3 структурних елементів [18]:

- основа (backbone) – згорткова глибока нейронна мережа, що виявляє візуальні ознаки різних форм і масштабу. В якості екстракторів ознак можуть використовуватись різні класифікаційні моделі, зокрема вже описаний ResNet, VGG або EfficientNet, але стандартним варіантом є CSPDarknet53;
- шия (neck) – набір шарів, які інтегрують і змішують характеристики перед передачею їх у шар прогнозування. В цій частині може використовуватись вже описана FPN – мережева піраміда ознак, – або інші подібні до неї алгоритми, наприклад, мережева агрегація шляхів (PAN) та Bi-FPN;

- голова (head) – приймає ознаки з шиї разом із прогнозами обмежувальних рамок. Виконує класифікацію та регресію цих рамок для завершення процесу виявлення. Зазвичай вихід останнього шару трансформується у ветор вигляду $(x, y, width, height)$, що містить координати центру обмежувальної рамки об’єкта та її розміри.

### *3.4.1 CSPDarknet53*

Ця базова модель вигідно вирізняється серед інших варіантів тим, що інтегрує в уже надійну та швидку Darknet концепцію CSPNet – Cross Stage Partial Network. Особливість CSPNet полягає у використанні абстрактної структури CSPBlock (рис. 3.6), що випадковим чином розділяє потік ознак на дві частини, одна з яких замість звичної обробки на серії згорткових шарів передається повз неї без змін. Згодом ці два набори ознак об’єднуються в один, і процес повторюється.

Ця операція дозволяє підвищити стабільність мережі та знизити її схильність до перенавчання, а також знизити обчислювальну складність моделі, при цьому не знижуючи її точність завдяки великій кількості згорткових шарів (53 в стандартному варіанті реалізації) [17].

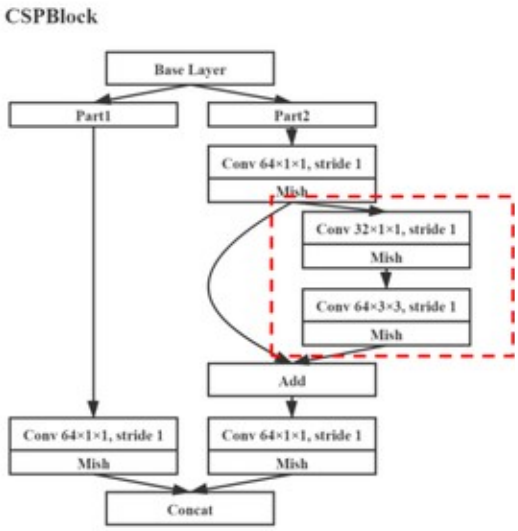



**Рисунок 3.6** – Схематичне зображення алгоритму роботи CSPBlock[11]

### *3.4.2 Anchor-free detection*

Одним з основних нововведень моделі версії 8 стало застосування без’якірної детекції (anchor-free detection). Цей алгоритм дозволяє визначати координати й обмежувальні рамки об’єктів без використання набору попередньо заданих якорів (anchor boxes) [14]. На рис. 3.7 показано 1% набору якорів, що використовується в моделі RetinaNet.

Архітектура YOLOv8 використовує декілька без’якірних методів визначення розташування об’єктів, зокрема такі:

- теплові карти (Heatmaps) – генерація теплових карт, що відображають ймовірність наявності об’єктів у кожному пікселі, та їх подальше виділення з використанням методів порогового значення;
- пряма регресія (Direct Regression) – безпосереднє передбачення

[11]Джерело зображення: https://www.researchgate.net/publication/351731859/figure/fig3/AS:11431281212301365@1702581198414/The-structure-of-CSPDarknet53-a-and-CSPDarknet53-tiny-b.tif

кутів обмежуючої рамки об’єкту на основі його ознак;

- ключові точки (Keypoints) – використання ключових точок, таких як центр об’єкту, для передбачення його розташування, форми й орієнтації.

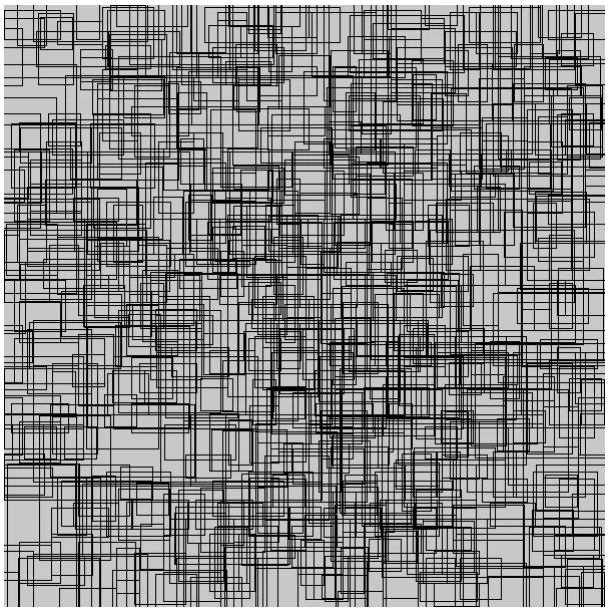


**Рисунок 3.7** – Візуалізація 1% якорів з набору, що часто використовується в нейромережах[12]

Завдяки поєднанню глибокої архітектури, блоку CSP в згорткових шарах та методів без’якірної детекції модель YOLOv8 є однією з найшвидших мереж розпізнавання й класифікації об’єктів на зображеннях, що в поєднанні з високою точністю та порівняно простим налаштуванням під різні потреби робить її ідеальним вибором зокрема й для застосуванні у задачах військового спрямування. Тим не менш, для ефективного тренування мережа потребує значних обчислювальних потужностей та збалансованого, якісно розміченого й великого датасету.

[12]Джерело зображення:
https://miro.medium.com/v2/resize:fit:640/format:webp/1*q7bYvvpZWeZ9dBVB4yI8EA.png

### 3.5 Опис програмного продукту

Для реалізації програмного продукту було використано такі інструменти й програмні компоненти:

- інструмент розмітки вхідних зображень – CVAT;
- мова програмування Python v3.12;
- середовище розробки програмного коду – Jupyter Notebook в поєднанні з IDE JetBrains PyCharm Professional;
- віртуальне середовище виконання програмного коду на основі інтерпретатора miniconda3;
- інструмент відслідковування прогресу навчання моделі ClearML;
- бібліотека imgaug для реалізації методів аугментації вхідних зображень;
- бібліотеки pandas, numpy для зберігання й трансформації формату інформації про обмежуючі рамки об’єктів;
- бібліотеки matplotlib, PIL для візуалізації графіків та зображень датасету;
- бібліотеки keras_cv, keras_retinanet для реалізації моделі RetinaNet_ResNet50V2;
- бібліотека ultralytics для реалізації мережі YOLOv8_small;
- бібліотеки tensorflow, sklearn для використання розширених методів навчання моделей.

Виконання програми функціонально та структурно поділено на такі етапи.

1. Обробка та аугментація вхідних даних: експорт анотацій з CVAT у форматі PASCAL VOC v1.1; парсинг необхідної інформації про об’єкти з отриманих файлів анотацій до змінної всередині програми; аугментація та запис нових версій зображень, що містять хоча б один об’єкт, до відповідної директорії; випадковий вибір 500

з 17000 зображень, що не містять об’єктів; оновлення змінної з інформацією про анотації файлів, запис повної інформації до файлу формату CSV; консолідація всіх зображень та анотацій датасету у спільну директорію; застосування методу K-Fold для реалізації крос-валідації під час навчання моделі, кількість ітерацій – 5.

2. Навчання моделі RetinaNet: імпорт переднавченої на датасеті Imagenet моделі ResNet50V2; компіляція моделі з застосуванням оптимізатора Adam та функції втрат класифікатора Focal Loss; створення генераторів для тренування та валідації з розміром батчу 16; тренування моделі із заданими параметрами протягом 50 епох; збереження вагів моделі до відповідного файлу.
3. Навчання моделі YOLOv8: імпорт переднавченої на датасеті COCO моделі YOLOv8s; налаштування конфігураційного файлу мережі; тренування моделі протягом 50 епох; збереження вагів моделі до відповідного файлу. порівняння результатів навчання отриманих моделей.

Приклад комірки Jupyter Notebook з кодом наведено на рис. 3.8.

```
def rename_batch(img_path: str, rn_img_path: str, prefix=str):
    for img_file in os.listdir(img_path):
        if img_file.endswith(".jpg") or img_file.endswith(".JPG"):
            shutil.copy(os.path.join(img_path, img_file), os.path.join(rn_img_path, prefix+img_file))
        elif img_file.endswith(".png"):
            im = Image.open(os.path.join(img_path, img_file))
            rgb_im = im.convert('RGB')
            rgb_im.save(os.path.join(rn_img_path, prefix+img_file+'.jpg'))
    return True

def rename_edit_xml(xml_path: str, rn_xml_path, prefix: str):
    for xml_file in os.listdir(xml_path):
        tree = ET.parse(os.path.join(xml_path, xml_file))
        root = tree.getroot()
        for child in root.iter('filename'):
            if child.text.endswith(".png"):
                child.text = prefix + child.text + ".jpg"
            else:
                child.text = prefix + child.text
        tree.write(os.path.join(rn_xml_path, prefix + xml_file))
    return True
Executed at 2024.06.15 01:00:22 in 13ms
```

**Рисунок 3.8** – Програмна комірка, що відповідає за редагування імен файлів та

відповідну зміну інформації у файлах анотації

### 3.6 Результати роботи програми

В процесі навчання обидві мережі показали приблизно однакові результати досягненої точності. З кожною наступною епохою кількість помилок на валідаційному наборі стабільно знижувалась, але не досягла нульового значення. Це може бути пов’язано передусім з невеликою кількістю зображень у батчі, що зумволено їхнім розміром, та нерівномірним розподілом об’єктів за класами. Таким чином, наприклад, об’єкти класу «окоп» розпізнаються моделями набагато краще, аніж вантажівки чи танки. Це також може бути пов’язано з тим, що броньована техніка зазвичай маскується деревами або снігом, що подекуди значно ускладнює її виявлення, в той час як окопи або капоніри  мають виразну текстуру, форму й межі.

Приклад визначення об’єктів нейронними мережами наведено на рис. 3.9.


**Рисунок 3.9** – Результат обробки зображення навченою моделлю (виявлені об’єкти позначено зеленими рамками з назвою класу)

## Висновки до розділу 3

Увагу в цьому розділі було зосереджено в основному на аналізі та первинній обробці вхідних даних, а також тому, як операції їх трансформації впливають на якість тренування моделі. Окрім того, розглянуто використані інструменти для розмітки датасету, алгоритм та в цілому можливості аугментації, досліджено ключові властивості архітектур моделей RetinaNet_ResNet50V2 та YOLOv8s, а також описано використані в процесі розробки програми компоненти та результати її роботи.

# РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У цьому розділі буде проведено оцінку основних характеристик майбутнього програмного продукту для пошуку військових об’єктів на аерофотознімках використовуючи методи штучного інтелекту.

Ця реалізація сприятиме проведенню всіх необхідних досліджень, що дозволить якісно вивчити питання не тільки в Україні, але й у всьому світі.

У даному дослідженні також було продемонстровано різноманітні варіанти реалізації, які сприяють найбільш коректній та оптимальній стратегії вибору. Це має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для досягнення цих цілей було використано метод функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це методологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Його головна мета – виявлення можливостей зниження витрат шляхом впровадження більш ефективних варіантів виробництва та досягнення кращого співвідношення між споживчою вартістю продукту та витратами на його виготовлення. Для проведення ФВА використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу передбачає визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих годин, визначення джерел витрат та остаточний розрахунок вартості програмного продукту.

## 4.1 Постановка задачі проектування

У роботі використовується метод ФВА для проведення техніко-економічного аналізу розробки системи пошуку військових об’єктів на аерофотознімках використовуючи методи штучного інтелекту. Оскільки рішення щодо проектування та реалізації розроблюваних компонентів впливають на всю систему, кожна окрема підсистема повинна їх задовольняти. Тому фактичний аналіз полягає у вивченні функцій програмного продукту, призначеного для збору, обробки та аналізу даних про компанію.

Технічні вимоги до програмного продукту є наступними:

- функціонування на персональних комп’ютерах зі стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

## 4.2 Обґрунтування функцій програмного продукту

Головна функція$F_0$– розробка можливого програмного продукту, яка дозволяє шукати військові об’єкти, визначати їх положення та відмічати на початковому знімку. Виділяються такі: $F_1$ – вибір мови програмування, $F_2$ – вибір основного методу пошуку об’єктів, $F_3$ – вибір середовища розробки.

Кожна з цих функцій має декілька варіантів реалізації:

- функція $F_1$:
  a) Python;

b) C++;

- функція $F_2$:
  a) RetinaNet;
  b) YOLOv8;
- функція $F_3$:
  a) Visual Studio Code;
  b) Jupyter Notebook.

Варіанти реалізації основних функцій наведено у морфологічній карті систем на рис. 4.1.

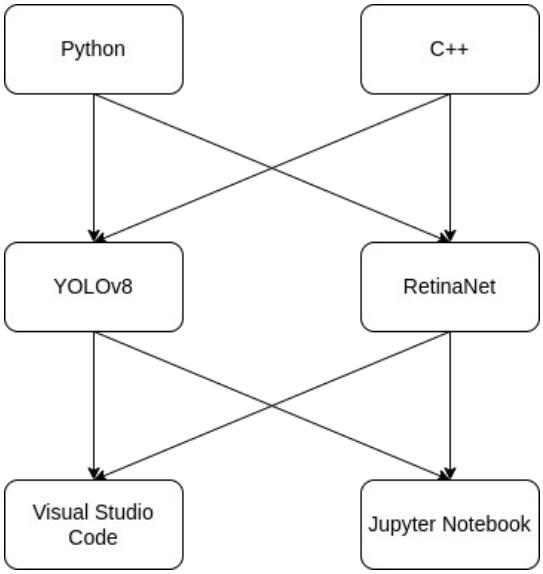



**Рисунок 4.1** – Морфологічна карта

Морфологічна карта показує множину основних функцій. Таблиця 4.1 відображає позитивно-негативну матрицю.

**Таблиця 4.1** – Позитивно-негативна матриця

| Функції | Варіанти реалізації | Переваги | Недоліки |
|---|---|---|---|
| $F_1$ | Python | Велика кількість та доступність бібліотек для будь-якої задачі, загальна простота написання програми. | Порівняно більший час виконання програми. |
| | C++ | Висока продуктивність написаної програми | Мала кількість бібліотек для взаємодії з зображеннями та нейронними мережами, складний синтаксис і збільшений час розробки. |
| $F_2$ | RetinaNet | Висока точність, здатність ефективно працювати з дрібними деталями та тренуватись на датасетах нижчої якості. | Складна архітектура, довший час навчання й роботи мережі. |
| | YOLOv8 | Ефективний баланс між високою точністю і швидкістю. | Проблеми з виявленням об’єктів різного масштабу, а також невисока пристосованість до дрібних деталей. |
| $F_3$ | Visual Studio Code | Широкий вибір доповнень, гнучкість у налаштуваннях, автоматичне заповнення та підтримка різних мов програмування. | Застарілість деяких систем підтримки розробки. |
| | Jupyter Notebook | Значно вища швидкість написання коду за рахунок можливості модульного (поблокового) виконання програми. | Мала кількість інструментів для полегшення розробки |

Згідно з аналізом позитивно-негативної матриці приходимо до наступного висновку. Деякі варіанти реалізації функцій слід відкинути, оскільки вони не відповідають поставленим перед програмним продуктом завданням. Ці варіанти відзначені у морфологічній карті.

Функція $F_1$.

Ми віддаємо перевагу швидкості розробки та простоті. Python має величезну кількість бібліотек і інструментів, які полегшують розробку програм та роблять її більш комфортною.

Функція $F_2$.

Віддаємо перевагу YOLO, через значно більшу легкість у розробці та інтеграції такої моделі. На даному етапі розробки програмного продукту недоцільно використовувати настільки потужний та складний інструмент як RetinaNet.

Функція $F_3$.

Перевагу надаємо більшій кількості додатків та розширень Visual Studio Code, які можуть значно прискорити та спростити розробку програмного застосунку.

Найбільше значення має вибір моделі пошуку об’єктів. Отже, будемо розглядати такі варіанти реалізації ПП:

$$F_1\text{а} - F_2\text{а} - F_3\text{а}$$

$$F_1\text{а} - F_2\text{б} - F_3\text{а}$$

Для оцінювання якості розглянутої функцій обрана система параметрів, описана нижче.

## 4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих раніше, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- X1 – швидкодія мови програмування;
- X2 – об’єм пам’яті для обчислень та збереження даних;

- X3 – час визначення об’єктів;
- X4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

**Таблиця 4.2** – Основні параметри програмного продукту

| Параметр | Умовні позначення | Одиниці виміру | Значення параметра | | |
|---|---|---|---|---|---|
| | | | гірші | середні | кращі |
| Швидкодія мови | X1 | оп/мс | 9000 | 15000 | 18000 |
| Об’єм пам’яті | X2 | Мб | 600 | 400 | 200 |
| Час визначення об’єкту | X3 | с | 7 | 4 | 2 |
| Потенційний об’єм програмного коду | X4 | кількість рядків коду | 1800 | 1400 | 1000 |

За даними табл. 4.2 побудовано графічні характеристики параметрів (рис. 4.2 – рис. 4.5).

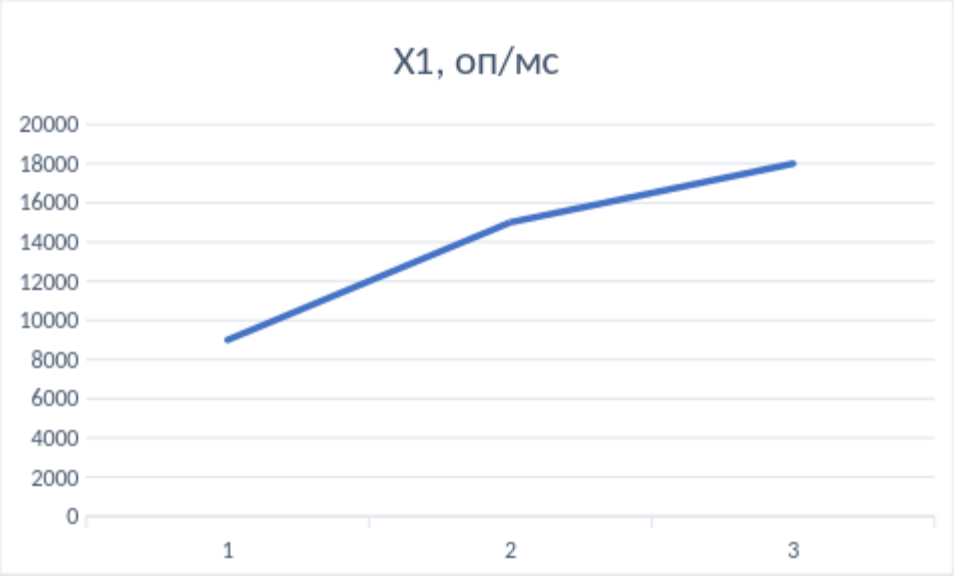



**Рисунок 4.2** – X1, швидкодія мови програмування

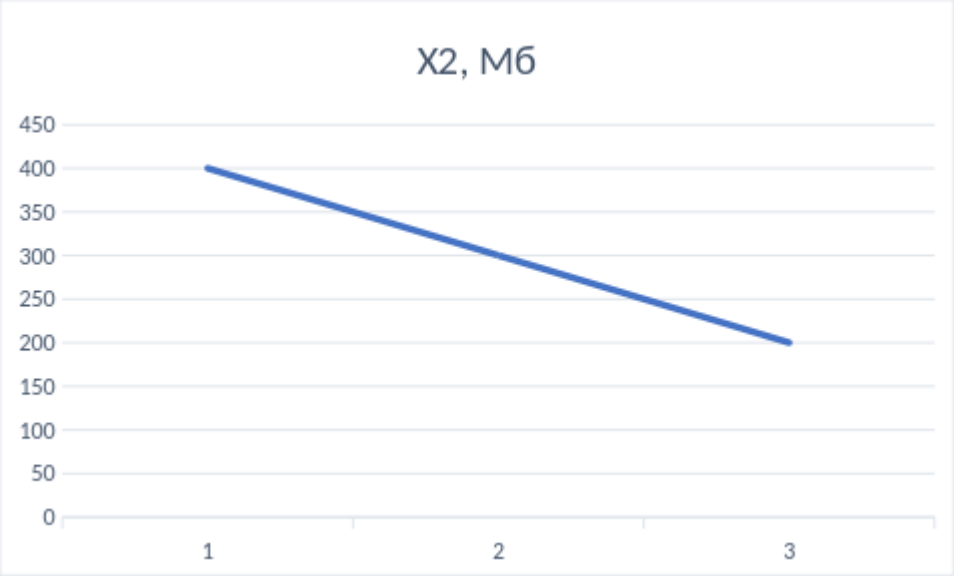



**Рисунок 4.3** – X2, об’єм пам’яті

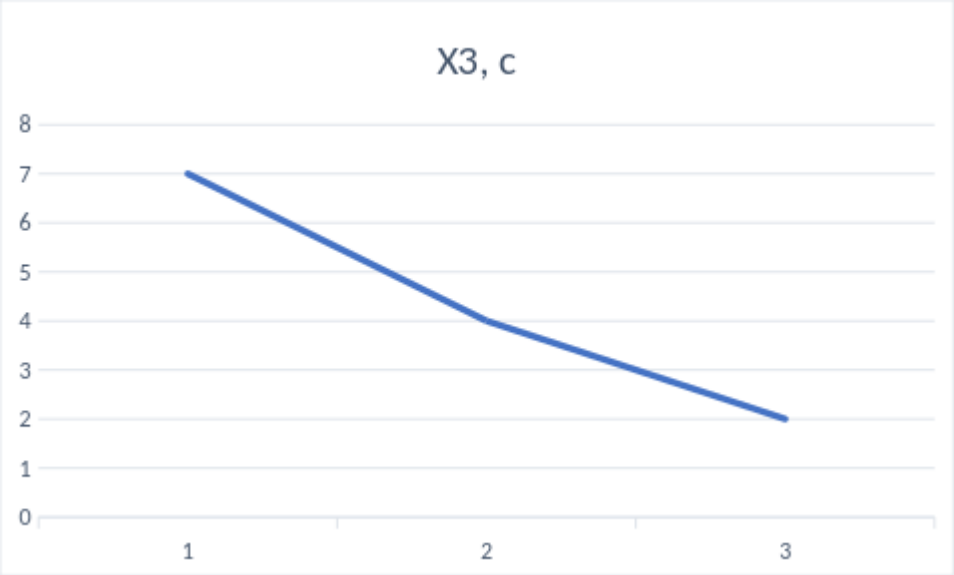



**Рисунок 4.4** – X3, час визначення об’єкта

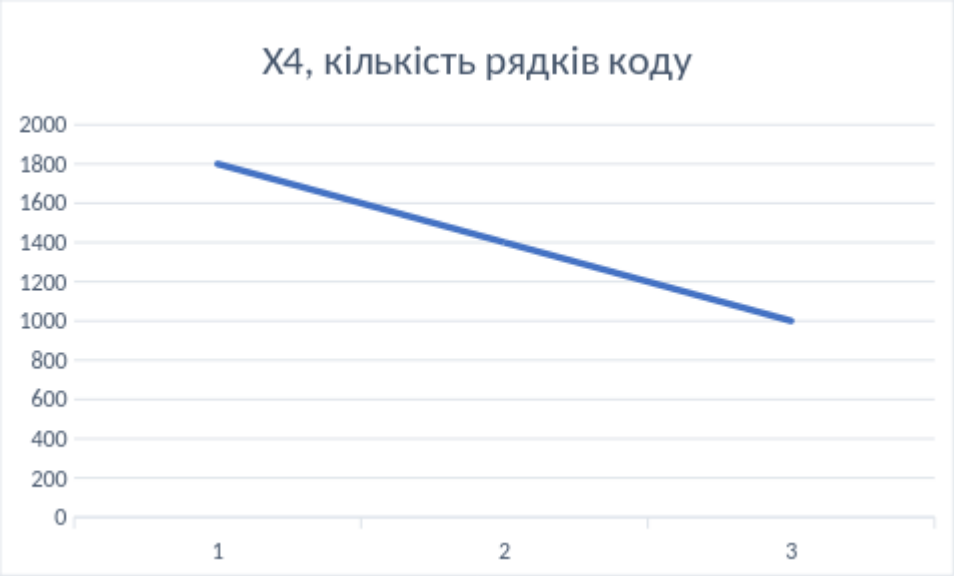



**Рисунок 4.5** – X4, потенційний об’єм програмного коду

### 4.4 Аналіз експертного оцінювання параметрів

Після ретельного обговорення та аналізу кожен експерт оцінює рівень важливості кожного параметру для конкретно поставленої мети – розробка програмного продукту, який забезпечує найточніші результати при визначенні параметрів моделей адаптивного прогнозування та обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

**Таблиця 4.3** – Результати ранжування параметрів

| Позначення параметра | Назва параметра | Одиниці виміру | Ранг параметра за оцінкою експерта | | | | | | | Сума рангів $R_1$ | Відхилення $\Delta_i$ | $\Delta_i^2$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | | |
| X1 | Швидкодія мови програмування | Оп/мс | 1 | 2 | 2 | 1 | 2 | 1 | 2 | 11 | -6.5 | 42.25 |
| X2 | Об’єм пам’яті | МВ | 3 | 4 | 3 | 3 | 4 | 3 | 4 | 24 | 6.5 | 42.25 |
| X3 | Час визначення об’єкту | с | 2 | 1 | 1 | 2 | 1 | 2 | 1 | 10 | -7.5 | 56.25 |
| X4 | Потенційний об’єм програмного коду | кількість рядків коду | 4 | 3 | 4 | 4 | 3 | 4 | 3 | 25 | 7.5 | 56.25 |
| | Разом | | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 70 | 0 | 197 |

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

- сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70 ,$$

де $N$ – число експертів,

$n$ – кількість параметрів;

- середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5$$

- відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T,$$

сума відхилень по всім параметрам має дорівнювати 0;

- загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} {}^{2}_{i} = 197.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 \quad n)} = \frac{12 \quad 197}{7^2(4^3 \quad 4)} = 0{,}804 > W_k = 0{,}67.$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, який дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

**Таблиця 4.4** – Попарне порівняння параметрів.

| Параметри | Експерти | | | | | | | Кінцева оцінка | Числове значення |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | |
| X1 і X2 | < | < | < | < | < | < | < | < | 0.5 |
| X1 і X3 | < | > | > | < | > | < | > | > | 1.5 |
| X1 і X4 | < | < | < | < | < | < | < | < | 0.5 |
| X2 і X3 | > | > | > | > | > | > | > | > | 1.5 |
| X2 і X4 | < | > | < | < | > | < | > | < | 0.5 |
| X3 і X4 | < | < | < | < | < | < | < | < | 0.5 |

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, $a_{ij}$ визначається по формулі:

$$a_{ij}=\begin{cases}1.5\,при\,X_i>X_j\\1.0\,при\,X_i=X_j\\0.5\,при\,X_i<X_j\end{cases}.$$

З отриманих числових оцінок переваги складемо матрицю $A=\|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{вi}=\frac{b_i}{\sum_{i=1}^{n} b_i},$$

$$b_i=\sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{вi}=\frac{b_i^{'}}{\sum_{i=1}^{n} b_i^{'}},$$

$$b_i^{'}=\sum_{i=1}^{N} a_{ij}b_j.$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

**Таблиця 4.5** – Розрахунок вагомості параметрів.

| Параметри $x_i$ | Параметри $x_j$ | | | | Перша ітер. | | Друга ітер. | | Третя ітер. | |
|---|---|---|---|---|---|---|---|---|---|---|
| | X1 | X2 | X3 | X4 | $b_i$ | $K_{вi}$ | $b_i^1$ | $K_{вi}^1$ | $b_i^2$ | $K_{вi}^2$ |
| X1 | 1 | 0,5 | 1,5 | 0,5 | 3,5 | 0,2188 | 12,25 | 0,2076 | 44,875 | 0,2078 |
| X2 | 1,5 | 1 | 1,5 | 0,5 | 4,5 | 0,2813 | 16,25 | 0,2754 | 59,125 | 0,2737 |
| X3 | 0,5 | 0,5 | 1 | 0,5 | 2,5 | 0,1563 | 9,25 | 0,1568 | 34,125 | 0,1580 |
| X4 | 1,5 | 1,5 | 1,5 | 1 | 5,5 | 0,3438 | 21,25 | 0,3602 | 77,875 | 0,3605 |
| Всього: | | | | | 16 | 1 | 59 | 1 | 216 | 1 |

## 4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об’єм пам’яті), *X3* (час визначення об’єкту) та *X4* (потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j},$$

де $n$ – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості $i$–го параметра;

$B_i$ – оцінка $i$–го параметра в балах.

**Таблиця 4.6** – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

| Основні функції | Варіант реалізації функції | Параметри | Абсолютне значення параметра | Бальна оцінка пара-метра | Коефіцієнт вагомості параметра | Коефіцієнт рівня якості |
|---|---|---|---|---|---|---|
| F1 | А | X1 | 16000 | 7 | 0,21 | 1,47 |
| F2 | А | X2 | 530 | 3 | 0,27 | 0,81 |
| | Б | X3 | 3 | 8 | 0,16 | 1,28 |
| F3 | А | X4 | 1100 | 9 | 0,36 | 3,24 |

За даними з таблиці 4.6 за формулою:

$$K_K = K_{TY}\left[F_{1k}\right] + K_{TY}\left[F_{2k}\right] + \ldots + K_{TY}\left[F_{zk}\right].$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1{,}47 + 0{,}81 + 3{,}24 = 5{,}52,$$

$$K_{K2} = 1{,}47 + 1{,}28 + 3{,}24 = 5{,}99.$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

## 4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання.

1. Розробка проекту програмного продукту.
2. Розробка програмної оболонки.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_O = T_P \quad K_П \quad K_{СК} \quad K_M \quad K_{СТ} \quad K_{СТ.М},$$

де $T_P$ – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_M$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_P$ = 60 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.6. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1.

Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.7. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 60 \cdot 1.6 \cdot 0.7 = 67,2 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =40 людино-днів, $K_П$ =0.9, $K_{СК}$ = 1, $K_{СТ}$ =0.75:

$$T_2 = 40 \cdot 0.9 \cdot 0.75 = 27 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (67,2 + 27 + 4.8 + 27) \cdot 8 = 1008 \text{ людино-годин.}$$

$$T_{II} = (67,2 + 27 + 6.91 + 27) \cdot 8 = 1024,88 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці бере участь один програміст-аналітик з окладом 26800 грн. Визначимо середню зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \quad t} грн.,$$

де М – місячний оклад працівників;

$T_m$ – кількість робочих днів тиждень;

$t$ – кількість робочих годин в день.

$$СЧ = \frac{26800}{21 \quad 8} = 159,52 грн.$$

Тоді, розрахуємо заробітну плату за формулою:

$$\text{СЗП} = C_{ч} \quad T_i \quad \text{КД},$$

де $C_ч$– величина погодинної оплати праці програміста;

$T_i$ – трудомісткість відповідного завдання;

$K_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I.}C_{ЗП} = 159{,}52 \cdot 1008 \cdot 1.2 = 192955{,}39 \text{ грн.}$$

$$\text{II.}C_{ЗП} = 159{,}52 \cdot 1024{,}88 \cdot 1.2 = 196186{,}63 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I.}C_{ВІД} = C_{ЗП} \cdot 0.22 = 192955{,}39 \cdot 0.22 = 42450{,}18 \text{ грн.}$$

$$\text{II.}C_{ВІД} = C_{ЗП} \cdot 0.22 = 196186{,}63 \cdot 0.22 = 43161{,}05 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. ($C_М$) Так як одна ЕОМ обслуговує одного програміста з окладом 26800 грн., з коефіцієнтом зайнятості 0,2, то для однієї машини отримаємо:

$$C_Г = 12 \cdot М \cdot K_З = 12 \cdot 26800 \cdot 0{,}2 = 64320 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{ЗП} = C_Г \cdot (1 + K_З) = 64320 \cdot (1 + 0.2) = 77184 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД}= С_{ЗП} \cdot 0.22 = 77184 \cdot 0,22 = 16980,48 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 35500 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.2 \cdot 0.25 \cdot 35500 = 10650 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.2 \cdot 35500 \cdot 0.07 = 2982 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.88 =$$

$$= 1640,32 \text{ години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1640,32 \cdot 0,35 \cdot 0,88 \cdot 5,32 = 2187 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;
$К_З$– коефіцієнтом зайнятості приладу;
$Ц_{ЕН}$ – тариф за 1 кВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 35500 \cdot 0,67 = 23785 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{ЕКС} = C_{ЗП} + C_{ВІД} + C_A + C_P + C_{ЕЛ} + C_H ,$$

$С_{ЕКС} = 77184 + 16980,48 + 10650 + 2982 + 2187 + 23785 = 133768,48$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 133768,48 / \ 1640,32 = 81,55 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_M = C_{M\ Г} \quad T,$$

I. $C_М = 81,55 \cdot 1008 = 82202,4$ грн.

II. $C_М = 81,55 \cdot 1024,88 = 83578,96$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \quad 0,67,$$

I. $C_Н = 192955,39 \cdot 0,67 = 129280,11$ грн.

II. $C_Н = 196186,63 \cdot 0,67 = 131445,04$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н ,$$

I. $C_{ПП} = 192955,39 + 42450,18 + 82202,4 + 129280,1 = 446888,07$ грн.

II. $C_{ПП} = 196186,63 + 43161,05 + 83578,96 + 131445,04 = 454371,68$ грн.

## 4.7 Вибір найкращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{ТЕР\,j} = K_{Кj} / C_{Фj},$$

$K_{ТЕР1} = 5,52 \quad / 446888,07 = 12,352 \cdot 10^{-6}$,

$K_{ТЕР2} = 5,99 \quad / \quad 454371,68 = 13,183 \cdot 10^{-6}$.

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{ТЕР2}$= 13,183·$10^{-6}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $K_{ТЕР}$ = 13,183·$10^{-6}$.

Цей варіант реалізації програмного продукту має такі параметри:

- вибір мови програмування – Python;
- модель інтерполяції відео – YOLO;
- середовище розробки – Visual Studio Code.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

## Висновки до розділу 4

У цьому розділі було проведено повний аналіз функціонально-вартісного характеру програмного продукту. Також було встановлено оцінку основних функцій даного програмного продукту.

В результаті виконання аналізу функціонально-вартісного характеру розроблюваного програмного комплексу, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують. На підставі проведеного аналізу було обрано варіант реалізації даного програмного продукту.

# ВИСНОВКИ

У цій дипломній роботі було виконано дослідження використання методів нейронних мереж для автоматизації розпізнавання та класифікації військових об’єктів на аерофотознімках.

Перший розділ було присвячено опису предметної області та постановці задачі на дипломну роботу. В ньому було обґрунтовано актуальність цієї роботи, проведено аналіз категорій військових об’єктів та огляд існуючих застосувань технологій штучного інтелекту й машинного навчання для застосування у воєнних цілях. Окрім цього, було пояснено причини вибору згорткових нейронних мереж як фундаментального способу вирішення поставленої задачі.

У другому розділі було досліджено й детально описано основні структурні елементи згорткових нейронних мереж, такі як шари згортки та підвибірки та функції активації. Для більшості елементів пояснено принцип їх роботи та надано приклади поширених методів їхньої реалізації, між якими проведено порівняння. Окрім того, розглянуто 2 основні способи підвищення ефективності нейронних мереж – нормалізація й регуляризація, – а також роз’яснено принцип навчання моделей та його ключові аспекти.

У третьому розділі проаналізовано вхідний датасет, показано процес його розмітки й підготовки до обробки нейронною мережею. Досліджено переваги й методи такої процедури як аугментація, пояснено процес її реалізації в програмному коді та надано приклади її роботи. Окрім того, детально розглянуто використані при написанні програмного застосунку моделі нейронних мереж RetinaNet та YOLOv8, описано їхні переваги й недоліки, а також особливі структурні й функціональні елементи, принцип їх роботи та мету застосування. Наприкінці розділу описано алгоритм реалізованого програмного продукту, використані при його написанні програмні компоненти, та показано приклад його застосування.

У четвертому розділі було проведено функціонально-вартісний аналіз програмного забезпечення, за результатами якого обрано оптимальний варіант реалізації програмного продукту. Зроблено висновок про ефективність використання нейронної мережі архітектури YOLOv8 для розпізнавання й класифікації військових об’єктів на аерофотознімках, використання мови програмування Python та середовища розробки Visual Studio Code.

В результаті роботи було створено дві системи, що показали себе однаково пристосованими до автоматизації розпізнавання й класифікації військових об’єктів на аерофотознімках, та можуть використовуватись для полегшення задачі розвідки й прискорення передачі інформації від підрозділу до командування. Подальша задача полягає у вдосконаленні цих систем для підвищення їх точності, розширенні й уточненні класів об’єктів для більшої універсальності систем, а також спрощення процесу імпорту зображень, призначених для обробки.

Для подальшого вдосконалення систем рекомендується оптимізувати параметри моделей для зменшення часу навчання й обробки зображень, провести більш ґрунтовне їх порівняння між собою з урахуванням обмеженості ресурсів на ЛБЗ з метою визначення найбільш ефективної для роботи в польових умовах, а також постійно підвищувати їхню здатність до розпізнавання замаскованих різними способами об’єктів шляхом регулярного доповнення тренувального датасету. Ці та інші потенційні вдосконалення безсумнівно сприятимуть підвищенню бойової ефективності як підрозділів розвідки, так і ЗСУ в цілому, що в умовах війни є критично важливою задачею.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Saif Ali, «LeNet-5 – A complete guide,» Apr 29, 2024. https://www.educative.io/blog/lenet-5
2. EITCA, «Які основні компоненти згорткової нейронної мережі (CNN) і як вони сприяють розпізнаванню зображень?», Aug 08, 2023. https://bit.ly/3VKrKMl
3. Віталій Кравченко, «Що таке нейронні мережі та як вони працюють? Класифікація штучних нейромереж», Jan 24, 2022. https://livingfo.com/shcho-take-nejronni-merezhi-ta-iak-vony-pratsiuiut/
4. Sonoo Jaiswal, «Activation Functions in Neural Networks». https://www.javatpoint.com/activation-functions-in-neural-networks
5. Pragati Baheti, «Activation Functions in Neural Networks [12 Types & Use Cases]», May 27, 2021. https://www.v7labs.com/blog/neural-networks-activation-functions
6. A. Krizhecsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", [Electronic resource]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
7. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition", [Electronic resource]. Available: https://arxiv.org/pdf/1512.03385.pdf
8. Samaya Madhavan, «Introduction to convolutional neural networks», Jul 12, 2021. https://developer.ibm.com/articles/introduction-to-convolutional-neural-networks/
9. Nafiz Shahriar, "What is Convolutional Neural Network – CNN (Deep Learning)", [Electronic resource]. Available: https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5

10. K. Simonyan, A. Zisserman, "Very deep convolutional networks for large-scale image recognition", [Electronic resource]. Available: https://arxiv.org/pdf/1409.1556.pdf
11. Christopher M. Bishop, «Pattern Recognition and Machine Learning», ISBN; 978-0-387-31073-2, Published: 17 August 2006
12. Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie, «Feature Pyramid Networks for Object Detection», Apr19, 2017. https://arxiv.org/abs/1612.03144
13. Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár, «Focal Loss for Dense Object Detection», Feb 7, 2018. https://arxiv.org/abs/1708.02002
14. Ze Yang, Shaohui Liu, Han Hu, Liwei Wang, Stephen Lin, «RepPoints: Point Set Representation for Object Detection», Aug 19, 2019. https://arxiv.org/abs/1904.11490
15. Ian Goodfellow and Yoshua Bengio and Aaron Courville, «Deep Learning. An MIT Press book», MIT Press, 2016. https://www.deeplearningbook.org/contents/regularization.html
16. Ian Goodfellow and Yoshua Bengio and Aaron Courville, «Deep Learning. An MIT Press book», MIT Press, 2016. https://www.deeplearningbook.org/contents/convnets.html
17. Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao, «YOLOv4: Optimal Speed and Accuracy of Object Detection», Apr 23, 2020. https://paperswithcode.com/paper/yolov4-optimal-speed-and-accuracy-of-object
*18.* Akshit Mehra, «Understanding YOLOv8 Architecture, Applications & Features», Apr 16, 2023. https://www.labellerr.com/blog/understanding-yolov8-architecture-applications-features/

# ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import clearml
from clearml import Task

augmentation_task = Task.init(project_name="ClearML-Augmentation",
task_name="Augmentation Test")

%%capture
import os, shutil
import xml.etree.ElementTree as ET
import numpy as np
import pandas as pd
import imgaug as ia
import imageio
import re
import glob
import PIL

from imgaug import augmenters as iaa
from imgaug.augmentables.bbs import BoundingBox, BoundingBoxesOnImage
from typing import Optional, Union, List, Any
from PIL import Image

import matplotlib.pyplot as plt

def rename_batch(img_path: str, rn_img_path: str, prefix=str):
    for img_file in os.listdir(img_path):
        if img_file.endswith(".jpg") or img_file.endswith(".JPG"):
```

```
            shutil.copy(os.path.join(img_path, img_file), os.path.join(rn_img_path,
prefix+img_file))
        elif img_file.endswith(".png"):
            im = Image.open(os.path.join(img_path, img_file))
            rgb_im = im.convert('RGB')
            rgb_im.save(os.path.join(rn_img_path, prefix+img_file+'.jpg'))
    return True

def rename_edit_xml(xml_path: str, rn_xml_path, prefix: str):
    for xml_file in os.listdir(xml_path):
        tree = ET.parse(os.path.join(xml_path, xml_file))
        root = tree.getroot()
        for child in root.iter('filename'):
            if child.text.endswith(".png"):
                child.text = prefix + child.text + ".jpg"
            else:
                child.text = prefix + child.text
        tree.write(os.path.join(rn_xml_path, prefix + xml_file))
    return True

# rename_batch(RAW_IMG_PATH + "task4", RNMD_IMG_PATH, "t4_")
# rename_edit_xml(ANNTS_PASCAL_PATH, RNMD_AS_PATH, "t4_"))

def xml_to_csv(path):                          # convert xmls in PASCAL format to
pandas DF
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.iter('object'):
```

```
            try:
                value = (root.find('filename').text,
                         int(root.find('size')[0].text),
                         int(root.find('size')[1].text),
                         member[0].text,
                         round(float(member[4][0].text)),
                         round(float(member[4][1].text)),
                         round(float(member[4][2].text)),
                         round(float(member[4][3].text))
                         )
                xml_list.append(value)
            except ValueError as val_e:
                print(val_e)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df

def bbs_obj_to_df(bbs_object):
    bbs_array = bbs_object.to_xyxy_array()
    df_bbs = pd.DataFrame(bbs_array, columns=['xmin', 'ymin', 'xmax', 'ymax'])
    return df_bbs

def augment_img(img_path = str, aug_img_path=str, batch_prefix=str, bbs_df =
pd.DataFrame):
        #   create sequential augmentor
    seq = iaa.Sequential([                  # create sequential augmentor
        iaa.Fliplr(0.5),                    # flip horizontally 50%
        iaa.Flipud(0.5),                     # flip vertically 50%
        iaa.LinearContrast((0.75, 1.5)),        # contrast - or +
        iaa.Sometimes(p=0.5, then_list=[        # 50% of photos will be blurred by one of
```

```
the methods
            iaa.OneOf([
                iaa.GaussianBlur((0, 3.0)),
                iaa.AverageBlur(k=(2, 3)),
                iaa.MedianBlur(k=(3, 5))
            ])
        ]),
        iaa.Sometimes(p=0.5, then_list=[        # 50% of photos will be affected by 1-3 of
following methods
            iaa.SomeOf((1,3), [
                iaa.Sharpen(alpha=(0, 1.0), lightness=(0.75, 1.5)),
                iaa.Emboss(alpha=(0, 1.0), strength=(0, 1.0)),
                iaa.Dropout((0.03, 0.1), per_channel=0.5),
                iaa.CoarseDropout((0.03, 0.1), size_percent=(0.3, 0.07), per_channel=0.2)
            ])
        ])
    ], random_order=True)
        #   create empty DF for FULL info on ALL bbs & group input bbs DF by
filename
    aug_bbs_all_full = pd.DataFrame(columns=['filename','width','height','class',
'xmin', 'ymin', 'xmax', 'ymax'])
    grouped = bbs_df.groupby('filename')

    for filename in bbs_df['filename'].unique():
        #   group_df is grouped by filename
        current_group_df = grouped.get_group(filename)
        current_group_df = current_group_df.reset_index()
        current_group_df = current_group_df.drop(['index'], axis=1)
        #   read images, get array of bbs & convert to BoundingBoxesOnImage
        image = imageio.imread(img_path + filename)
```

```
        bbs_array = current_group_df.drop(['filename', 'width', 'height', 'class'],
axis=1).values
        bbs = BoundingBoxesOnImage.from_xyxy_array(bbs_array,
shape=image.shape)
        #   apply sequential augmentor
        aug_img, aug_bbs = seq(image=image, bounding_boxes=bbs)
        #   fix bbs
        aug_bbs = aug_bbs.remove_out_of_image()
        aug_bbs = aug_bbs.clip_out_of_image()
        #   if there are no bbs left on image, pass
        if re.findall('Image...', str(aug_bbs)) == ['Image([]']:
            pass
        #   otherwise write file down with prefix
        else:
            imageio.imwrite(aug_img_path + batch_prefix + filename, aug_img)
        #   aug_img_params_df for augmented width & height
            aug_img_params_df = current_group_df.drop(['xmin', 'ymin', 'xmax', 'ymax'],
axis=1)
            for index, _ in aug_img_params_df.iterrows():
                aug_img_params_df.at[index, 'width'] = aug_img.shape[1]
                aug_img_params_df.at[index, 'height'] = aug_img.shape[0]
            aug_img_params_df['filename'] =
aug_img_params_df['filename'].apply(lambda x: batch_prefix+x)
            bbs_df = bbs_obj_to_df(aug_bbs)
            aug_full_df = pd.concat([aug_img_params_df, bbs_df], axis=1)
            aug_bbs_all_full = pd.concat([aug_bbs_all_full, aug_full_df])

    aug_bbs_all_full = aug_bbs_all_full.reset_index()
    aug_bbs_all_full = aug_bbs_all_full.drop(['index'], axis=1)
    return aug_bbs_all_full
```

```
import pandas as pd

df = pd.read_csv("./data/train/bbs_augmented.csv")
grouped = df.groupby('filename')
no_name = grouped.get_group('aug_t1_no-name.png.jpg')
no_name.head(15)

no_name.iloc[0]['xmin']

import imgaug as ia
import imageio
from imgaug.augmentables.bbs import BoundingBox, BoundingBoxesOnImage

# Load your image
image = imageio.imread("./data/train/pos_img/aug_t1_no-name.png.jpg")

list = []
for index, bs in no_name.iterrows():
    nbs = BoundingBox(x1=bs['xmin'], y1= bs['ymin'], x2= bs['xmax'], y2= bs['ymax'],
label=bs['class'])
    list.append(nbs)

# Define your bounding boxes
bbs = BoundingBoxesOnImage(list, shape=image.shape)

# Draw bounding boxes on the image
image_with_bbs = bbs.draw_on_image(image, size=2)

# Display the image with bounding boxes
```

```
ia.imshow(image_with_bbs)
augmentation_task.close()

import clearml
from clearml import Task

training_task = Task.init(project_name="RetinaNet training",
task_name="Test1_RetinaNet")

%%capture
import os
import pandas as pd
import numpy as np
import tensorflow as tf
from keras_retinanet.preprocessing.csv_generator import CSVGenerator

from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score

pos_img = os.listdir(POS_IMG_PATH)
neg_img = os.listdir(NEG_IMG_PATH)

orig_bbs_df = pd.read_csv(ORIG_BBS_CSV_PATH)
aug_bbs_df = pd.read_csv(AUG_BBS_CSV_PATH)
pos_df = pd.concat([orig_bbs_df, aug_bbs_df], ignore_index=True)
pos_df['filename'] = POS_IMG_PATH + '/' + pos_df['filename']
pos_df = pos_df.drop(columns=['width', 'height'])
pos_df[['xmin', 'ymin', 'xmax', 'ymax']] = pos_df[['xmin', 'ymin', 'xmax',
'ymax']].astype(int)
```

```
neg_img_sample = np.random.choice(neg_img, 500, replace=False)
neg_df = pd.DataFrame(neg_img_sample, columns = ['filename'])
neg_df['filename'] = NEG_IMG_PATH + '/' + neg_df['filename']
neg_df['xmin'] = None
neg_df['ymin'] = None
neg_df['xmax'] = None
neg_df['ymax'] = None
neg_df['class'] = None



full_df = pd.concat([pos_df, neg_df], ignore_index=True)
full_df = full_df[['filename', 'xmin', 'ymin', 'xmax', 'ymax', 'class']]
full_df.to_csv("./data/train/bbs_pos_neg.csv", index=False, header = False)

classes = full_df['class'].dropna().unique()
class_mapping = {class_name: idx for idx, class_name in enumerate(classes)}
classes_df = pd.DataFrame(list(class_mapping.items()), columns=['class', 'id'])
classes_df.to_csv('./data/train/class_mapping.csv', index=False, header=False)
import keras_cv
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau,
EarlyStopping
from keras_retinanet.preprocessing.csv_generator import CSVGenerator

def generator_wrapper(generator):
    for inputs, targets in generator:
        boxes = tf.convert_to_tensor(targets['boxes'], dtype=tf.float32)
        classes = tf.convert_to_tensor(targets['labels'], dtype=tf.float32)
        yield inputs, {'boxes': boxes, 'classes': classes}
```

```
def train_retinanet(train_csv, val_csv, pretrained_model_path):
    # Load pretrained RetinaNet model
    model = keras_cv.models.RetinaNet(
        num_classes=5,
        bounding_box_format="xyxy",

backbone=keras_cv.models.ResNet50Backbone.from_preset("resnet50_imagenet")
    )
    # Train model
    model.compile(
        classification_loss='focal',
        box_loss='smoothl1',
        optimizer=keras.optimizers.Adam(learning_rate=1e-5),
        jit_compile=False,
    )
        # Create data generators
    train_generator = CSVGenerator(
        csv_data_file=train_csv,
        csv_class_file=CLASS_MAPPING_PATH,
        base_dir='',
        batch_size=16,
        image_min_side=3456,
        image_max_side=4608
    )

    val_generator = CSVGenerator(
        csv_data_file=val_csv,
        csv_class_file=CLASS_MAPPING_PATH,
```

```
        base_dir='',
        batch_size=16,
        image_min_side=3456,
        image_max_side=4608
    )

     # Wrap generators
    train_gen = generator_wrapper(train_generator)
    val_gen = generator_wrapper(val_generator)

    # Define callbacks (optional but recommended)
    checkpoint = ModelCheckpoint(RETINANET_TRAIN_MODEL_PATH,
monitor='val_loss', save_best_only=True, save_weights_only=True, verbose=1)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5,
verbose=1)
    early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

    # Train model
    model.fit(
        train_gen,
        epochs=1,
        steps_per_epoch=len(train_generator),
        validation_data=val_gen,
        validation_steps=len(val_generator),
        callbacks=[checkpoint, reduce_lr, early_stopping],
        verbose=1
    )

    return model
```

```
from keras_retinanet.models import load_model as load_retinanet_model
from keras_retinanet.utils.eval import evaluate

def evaluate_model(val_csv, model_path):

    # Create data generator for testing
    test_generator = CSVGenerator(
        csv_data_file=val_csv,
        csv_class_file=CLASS_MAPPING_PATH,
        image_min_side=3456,
        image_max_side=4608,
        batch_size=1,
        transform_generator=None,
        shuffle_classes=False,
        shuffle=False,
        convert_classes_to_ids=False
    )

    # Load trained RetinaNet model
    model = load_retinanet_model(model_path, backbone_name='resnet50')

    average_precisions = evaluate(test_generator, model)

    for label, average_precision in average_precisions.items():
        print(f'Class: {label} - Average Precision: {average_precision:.4f}')

    mean_ap = np.mean(list(average_precisions.values()))
    print(f'Overall Mean Average Precision: {mean_ap:.4f}')
```

```
    return average_precisions, mean_ap
kf = KFold(n_splits=N_SPLITS, shuffle=True, random_state=42)
accuracies = []
grouped_df = full_df.groupby('filename')
groups = [group for group, _ in grouped_df]

for fold, (train_index, val_index) in enumerate(kf.split(groups)):
    print(f"Fold {fold+1}/{N_SPLITS}")

    train_groups = [groups[i] for i in train_index]
    val_groups = [groups[i] for i in val_index]

    train_data = full_df[full_df['filename'].isin(train_groups)]
    val_data = full_df[full_df['filename'].isin(val_groups)]

    train_data.to_csv(TRAIN_CSV_PATH, index=False, header=False)
    val_data.to_csv(VAL_CSV_PATH, index=False, header=False)

    train_retinanet(TRAIN_CSV_PATH, VAL_CSV_PATH,
RETINANET_INITIAL_PATH)

    accuracy = evaluate_model(VAL_CSV_PATH,
RETINANET_TRAIN_MODEL_PATH)
    accuracies.append(accuracy)

# Виведення середньої точності по всім фолдам
print(f"Mean accuracy across {5} folds: {np.mean(accuracies)}")

training_task.close()
```

```
import ultralytics
from ultralytics import YOLO

# Завантаження попередньо натренованої моделі YOLOv8
model = YOLO('yolov8n.pt')

# Тренування моделі на власному датасеті
results = model.train(data='data.yaml', epochs=100, imgsz=640, batch=16)

# Валідація моделі
results = model.val()

# Використання натренованої моделі для передбачення
predictions = model('path/to/your/image.jpg')

import ultralytics
from ultralytics import YOLO
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import cv2

# Завантаження попередньо натренованої моделі YOLOv8
model = YOLO('yolov8n.pt')

# Використання натренованої моделі для передбачення
image_path = 'path/to/your/image.jpg'
results = model(image_path)

# Отримання зображення
image = cv2.imread(image_path)
```

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Відображення зображення з bounding boxes
fig, ax = plt.subplots(1, figsize=(12, 9))

# Вивід зображення
ax.imshow(image)

# Перебір передбачень та накладення bounding boxes
for result in results.xyxy[0]:  # xyxy формат
    x_min, y_min, x_max, y_max, confidence, class_id = result
    rect = patches.Rectangle((x_min, y_min), x_max - x_min, y_max - y_min,
linewidth=2, edgecolor='r', facecolor='none')
    ax.add_patch(rect)
    # Додавання підпису з класом та рівнем довіри
    plt.text(x_min, y_min, f'{model.names[int(class_id)]}: {confidence:.2f}',
bbox=dict(facecolor='yellow', alpha=0.5))

plt.axis('off')
plt.show()
```