

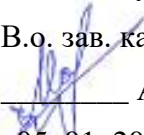
**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»**

**Радіотехнічний факультет  
Кафедра прикладної радіоелектроніки**

«На правах рукопису»  
УДК 004.55

До захисту допущено:

В.о. зав. кафедри

 Андрій МОВЧАНЮК

«05»01. 2024 р.

**Магістерська дисертація**

**на здобуття ступеня магістра**

**за освітньо-професійною програмою «Інтелектуальні технології радіоелектронної техніки»**

**за спеціальністю 172 «Електронні комунікації та радіотехніка»**

**на тему: « Розпізнавання об'єктів на зашумлених зображеннях»**

Виконав:


студент 2 курсу, групи РЕ-21мп  
Левченко Ігор Сергійович

Керівник:

Лащевська Наталія Олександрівна

Рецензент:

Сушко Олександр Юрійович



підпис

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент 

Київ – 2024 року

**Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»**

**Радіотехнічний факультет**

**Кафедра прикладної радіоелектроніки**

Рівень вищої освіти – другий (магістерський)

Спеціальність – 172 «Електронні комунікації та радіотехніка»

Освітньо-професійна програма «Інтелектуальні технології радіоелектронної техніки»

**ЗАТВЕРДЖУЮ**

В.о. зав. кафедри

**Андрій МОВЧАНЮК**

« 09 » 11 2023 р.

**ЗАВДАННЯ**

**на магістерську дисертацію студента  
Левченка Ігоря Сергійовича**

1. Тема дисертації «Розпізнавання об'єктів на зашумлених зображеннях»  
науковий керівник дисертації Лащевська Наталія Олександрівна  
затверджені наказом по університету від «09» листопада 2023 р. №  
5206-с

2. Термін подання студентом дисертації 11 січня 2024 року

3. Об'єкт дослідження: методи, техніки та архітектури нейронних мереж, які дозволяють виконувати розпізнавання об'єктів на зашумлених знімках

4. Вихідні дані: супутникові знімки з розпізнаними об'єктами

5. Перелік завдань, які потрібно розробити: 1. Огляд матеріалів та літератури дотичної до теми завдання; 2. Дослідження актуальності теми 3. Ознайомлення з основами штучного інтелекту та нейронних мереж; 4. Аналіз існуючих методів та архітектур зменшення шуму на зображенні; 5. Аналіз існуючих методів та архітектур нейронних мереж для розпізнавання об'єктів; 6. Вибір та обґрунтування методів та архітектур для зменшення шуму та для розпізнавання об'єктів; 7. Реалізація такої архітектури, що дозволить розпізнавати

об'єкти на зашумлених супутникових знімках; 8. Перевірка роботи такої реалізації та аналіз отриманих результатів.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: Презентація, лістинг програми, результати роботи нейронної мережі з зашумленими зображеннями.

7. Орієнтовний перелік публікацій: «Розпізнавання зображень отриманих із супутникових систем з використанням нейронних мереж», V Всеукраїнська науково-технічна конференція студентів та аспірантів «Радіoeлектроніка в XXI столітті»

8. Дата видачі завдання 01 вересня 2023 року

#### Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Отримання теми магістерської дисертації	09.11.2023р.	виконано
2	Розробка плану магістерської дисертації	10.11.2023р. – 12.11.2023р	виконано
3	Збір інформації для дослідження	13.11.2023р. – 19.11.2023р.	виконано
4	Підготовка матеріалів до першого розділу	20.11.2023р. – 24.11.2023р.	виконано
5	Підготовка матеріалів до другого розділу	25.11.2023р. – 29.11.2023р.	виконано
6	Вибір та формування датасету	30.11.2023р. – 04.12.2023р.	виконано
7	Написання, тренування та тестування нейронної мережі	05.12.2023р. – 21.12.2023р.	виконано
8	Підготовка матеріалів до третього розділу	22.12.2023р. – 26.12.2023р.	виконано
9	Підготовка матеріалів до четвертого розділу	27.12.2023р. – 30.12.2023р.	виконано
10	Написання висновків	02.01.2024р. – 04.01.2024р.	виконано
11	Оформлення магістерської дисертації	05.01.2024р. – 11.01.2024	виконано

Студент



ЛЕВЧЕНКО Ігор

Науковий керівник



ЛАЩЕВСЬКА Наталія

## РЕФЕРАТ

Текстова частина магістерської дисертації містить: 99 с., 65 рис., 2 табл., 2 додатка, 40 використаних джерел.

Мета роботи: Реалізувати ефективний метод машинного навчання, за допомогою вибору архітектури нейронної мережі для розпізнавання об'єктів різних типів на зашумлених супутникових знімках.

Для досягнення поставленої мети в магістерській дисертації були вирішені наступні завдання: збір та підготовка даних; попередня обробка даних, а саме використання та навчання автокодера для зменшення рівня шумів для покращення точності розпізнавання; використання та навчання нейронної мережі YOLO останньої версії для розпізнавання об'єктів на попередньо оброблених знімках; тестування та оцінка результатів.

В результаті було отримано модель нейронної мережі, яка продемонструвала гарну точність у розпізнаванні об'єктів на зашумлених супутникових знімках. Вона може бути реалізована для застосування у сферах моніторингу довкілля, геологічних досліджень та низці інших областей, де важлива автоматизована обробка супутникових зображень.

Можливе подальше удосконалення моделі, тобто продовження дослідження для покращення точності та надійності розпізнавання об'єктів, шляхом покращення ланки автокодера або застосування інших комбінацій підходів машинного навчання. Також можливе покращення моделі шляхом реалізації розпізнавання об'єктів в режимі реального часу.

ДИСТАНЦІЙНЕ ЗОНДУВАННЯ, СУПУТНИКОВІ ЗНІМКИ, ШУМИ, ЗМЕНШЕННЯ ШУМІВ, РОЗПІЗНАВАННЯ ОБ'ЄКТІВ, НЕЙРОННА МЕРЕЖА, АВТОКОДЕР, YOLO.

## ABSTRACT

The text part of the master's thesis contains: 99 p., 65 fig., 2 tables, 2 appendix, 40 references.

The purpose of the work: To implement an effective machine learning method by selecting a neural network architecture for recognition various types of objects at noisy satellite images.

To achieve this goal, the following tasks were solved in the master's thesis: data collection and preparation; data preprocessing, namely using and training an auto-encoder to reduce noise and improve recognition accuracy; using and training the latest version of the YOLO neural network to recognize objects on pre-processed images; testing and evaluation of results.

As a result a neural network model demonstrated good accuracy in recognizing objects on noisy satellite images was obtained. It could be implemented for use in environmental monitoring, geological research, and a number of other areas where automated processing of satellite images is important.

Further improvement of the model is possible, i.e., continuing research to improve the accuracy and reliability of object recognition by improving the auto-encoder part or applying other combinations of machine learning approaches. It's also possible to improve the model by implementing real-time object recognition.

REMOTE SENSING, SATELLITE IMAGERY, NOISE, NOISE REDUCTION, OBJECT RECOGNITION, NEURAL NETWORK, AUTOENCODER, YOLO.

**ПОЯСНЮВАЛЬНА ЗАПИСКА**  
**до магістерської дисертації**

на тему: "Розпізнавання об'єктів на зашумлених зображеннях"

Київ — 2024 року

## ЗМІСТ

Перелік скорочень.....	10
Вступ.....	11
1 Огляд предметної області.....	13
1.1 Засоби і методи зондування земної поверхні.....	13
1.1.1 Дистанційне зондування .....	13
1.1.2 Історія розвитку засобів дистанційного зондування.....	13
1.1.3 Датчики дистанційного зондування.....	15
1.1.4 Методи дистанційного зондування.....	16
1.2 Основи штучних нейронних мереж .....	16
1.3 Навчання штучної нейронної мережі.....	19
1.4 Алгоритм зворотного поширення помилки .....	21
1.5 Архітектури штучних нейронних мереж.....	23
1.5.1 Штучна нейронна мережа прямого поширення .....	23
1.5.2 Рекурентна нейронна мережа .....	24
1.5.3 Згортова нейронна мережа.....	25
1.6 Висновки до першого розділу .....	27
2 Ахітектури НМ Deep Learning .....	28
2.1 Мережі LeNet-5 та AlexNet .....	28
2.2 Мережі VGG-16 VGG-19.....	30
2.3 Мережа ResNet .....	33
2.4 Мережі R-CNN, Fast R-CNN та Faster R-CNN .....	35
2.4.1 R-CNN .....	35
2.4.2 Мережа Fast R-CNN.....	37

2.4.3	Мережа Faster R-CNN.....	38
2.5	You Only Look Once — YOLO.....	39
2.6	Single Shot Detector .....	43
2.7	Порівняння ефективності архітектур для виявлення об'єктів .....	44
2.8	Висновки до другого розділу.....	48
3	Підходи до обробки знімків та виявлення об'єктів.....	49
3.1	Шум на супутникових знімках та методи його зменшення .....	49
3.1.1	Типові шуми .....	49
3.1.2	Підходи до зменшення шумів.....	50
3.1.3	Автокодери .....	51
3.2	Архітектура обраного методу зменшення шуму .....	53
3.2.1	Архітектура автокодера.....	54
3.3	Задача виявлення об'єктів .....	55
3.4	Метрики оцінки ефективності моделей виявлення об'єктів.....	57
3.4.1	Intersection over Union.....	57
3.4.2	Average Precision .....	58
3.5	You Only Look Once v8.....	60
3.5.1	Архітектура YOLOv8.....	61
3.6	Висновки до третього розділу .....	67
4	Реалізація Нейронної мережі .....	68
4.1	Підготовка наборів даних.....	68
4.1.1	Доповнення набору даних шляхом видалення фрагментів із зображення.....	70
4.1.2	Доповнення набору даних шляхом обертання оригінального знімку.....	72



4.1.3 Доповнення набору даних шляхом перекошу зображення.....	73
4.2 Навчання автокодера .....	74
4.3 Навчання мережі YOLOv8.....	77
4.4 Отримані результати.....	78
4.5 Висновки до четвертого розділу .....	86
Висновки .....	87
Перелік джерел посилань .....	89
Додаток А.....	93
Додаток Б .....	94

**ПЕРЕЛІК СКОРОЧЕНЬ**

БПЛА — Безпілотний літальний апарат  
НМ — Нейронна мережа  
РНМ — Рекурентна нейронна мережа  
ШНМ — Штучна нейронна мережа  
AUC — Area Under the Curve  
AP — Average Precision  
BAM — Bidirectional associative memory  
CNN — Convolutional Neural Network  
FC — Fully Connected  
FP — False Positives  
IndRNN — Independently Recurrent Neural Network  
IoU — Intersection over Union  
LiDAR — Light Identification, Detection and Ranging  
LSTM — Long short-term memory  
mAP — Mean Average Precision  
ReLU — Rectified Linear Unit  
R-CNN — Region-Based Convolutional Neural Network  
RNN — Recurrent Neural Network  
SiLU — Sigmoid Linear Unit  
SPP — Spatial Pyramid Pooling  
SPPF — Spatial Pyramid Pooling Fast  
SSD — Single Shot Detector  
TP — True Positives  
VGG — Visual Geometry Group  
YOLO — You Only Look Once

## ВСТУП

На сьогоднішній день на Земній орбіті перебуває близько трьох тисяч штучних супутникових систем, при цьому їх кількість збільшується з кожним днем і, одночасно з тим, як наслідок збільшилися обсяги даних, які можна отримати з цих супутників. Сучасні супутникові системи надають значну кількість зображень, які можуть бути використані для відслідковування змін на Землі, здійснення геологічних досліджень, а також для збору інформації про стан атмосфери і клімату. Однак, збільшення обсягу даних такого роду ставить нові завдання з їхньої обробки, щоб забезпечити ефективне використання таких інформаційних ресурсів.

Використання нейронних мереж є одним зі способів розв'язання цієї проблеми. Нейронні мережі дозволяють автоматизувати процес обробки та аналізу супутникових зображень. Вони є потужним інструментом для обробки великої кількості зображень і можуть бути використані для автоматичного розпізнавання об'єктів на зображеннях, а також, наприклад, для відстеження змін у часі. Проте, однією з головних проблем у використанні супутникових знімків є складність розпізнавання об'єктів через наявний шум.

У цьому контексті актуальним є питання нових підходів до методів обробки супутникових зображень для ефективного розпізнавання об'єктів на них. Застосування нейронних мереж дає можливість покращити точність та швидкість аналізу, зменшити вплив шуму та підвищити рівень автоматизації в роботі з супутниковими зображеннями.

На сьогоднішній день існують певні підходи до розпізнавання об'єктів на супутникових знімках за допомогою нейронних мереж. Однак, є необхідним подальше вдосконалення цих методів та їх адаптація до різноманітних умов отримання зображень з метою отримання більш точних та стабільних результатів. Крім цього дана робота має потенціал для подальшого розвитку наукових та практичних досліджень у цій області.

Результати дослідження можуть знайти широке застосування в галузях картографування, розвідки, відслідковування змін на Землі, плануванні містобудівних та інфраструктурних проєктів, що сприятиме покращенню якості прийняття рішень у цих сферах.

## 1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

### 1.1 Засоби і методи зондування земної поверхні

#### 1.1.1 Дистанційне зондування

Термін "дистанційне зондування" є відносно новим і вперше був використаний для опису цієї галузі в 1960-х роках. Хоча сам термін «дистанційне зондування» з'явився лише в середині двадцятого століття, дистанційне зондування вперше почалося майже 150 років тому. Аерофотозйомка є найпершим прикладом дистанційного зондування [1].

Дистанційне зондування — це наука про отримання фізичних властивостей місцевості без необхідності перебувати там. Воно дозволяє користувачам фіксувати, візуалізувати та аналізувати об'єкти та особливості земної поверхні. Збираючи знімки, ми можемо класифікувати їх для наприклад аналізу ґрунтового покриву, моніторингу водних ресурсів, визначення масштабів природних катастроф та оцінки їх наслідків, аналізу клімату, та інших видів аналізу.

#### 1.1.2 Історія розвитку засобів дистанційного зондування

Історія розвитку засобів дистанційного зондування починається ще в середині XIX століття. У 1855 році французький фотограф Гаспар-Фелікс Турнашон, більш відомий під псевдонімом Надар, вперше запропонував ідею фотографування землі з повітряних куль для картографування та спостереження за територією. Він реалізував цю ідею 1858 року, зробивши фотографії селища поблизу Парижа з висоти 80 метрів [1].

На початку 20-го століття зображення дистанційного зондування робили за допомогою повітряних зміїв і навіть за допомогою камер, встановлених на голубах. Колаж аерофотознімків зробленими за допомогою голубів з камерами показаний на Рисунок 1.1 [2].



Рисунок 1.1 — Колаж аерофотознімків (зліва) зробленими за допомогою голубів з камерами (справа)

У 1906 році професійний фотограф Джордж Лоуренс використав трос під'єднаний до повітряних зміїв, щоб підняти 49-фунтову камеру на висоту 1000 футів та зафіксувати руйнування від землетрусу в Сан-Франциско.

Та сама фотографія «Сан-Франциско в руїнах» зображена на Рисунок 1.2.



Рисунок 1.2 — «Сан-Франциско в руїнах» Джордж Лоуренс 1906 р.

Перші аерофотознімки, зроблені з літака, були отримані у 1909 році Вілбером Райтом. До початку Першої світової війни камери, встановлені на літаках, дозволяли отримувати повітряні знімки великих територій, що виявилось безцінним для військової розвідки. До Другої світової війни літаки були обладнані камерами, а союзні війська найняли команду експертів для перегляду мільйонів стереоскопічних аерофотознімків для виявлення прихованих нацистських ракетних баз.

Розвиток супутникового дистанційного зондування почався з "космічних перегонів" між СРСР та США в 50 – 60-х роках ХХ століття. Наступні десяти-

ліття принесли швидкий розвиток супутників і технологій отримання зображень. У 1972 році США запустили перший супутник для отримання інформації — Landsat 1. Початковою метою програми Landsat був збір даних про Землю за допомогою методів дистанційного зондування. Програма Landsat триває вже 51 рік, і в 2021 році було запущено супутник Landsat 9. З моменту запуску "Супутника" СРСР в 1957 році було запущено тисячі супутників. Сьогодні в експлуатації перебуває величезна кількість комерційних і державних супутників, багато з яких використовуються для дистанційного зондування.

Супутникова зйомка зробила революцію в наших знаннях про Землю, завдяки детальним зображенням майже кожного куточка вулиці, які можна легко знайти в Інтернеті. Останніми роками значно зросла кількість комерційних супутників для зйомки.

Ще одним засобом дистанційного зондування, який на сьогоднішній дуже стрімко розвивається, є безпілотні літальні апарати (БПЛА). Безпілотні авіаційні комплекси, більш відомі як дрони, — це будь-який тип літальних апаратів, які не мають на борту людини-пілота. БПЛА можуть управлятися за допомогою пульта дистанційного керування з пілотом на землі або автономно за допомогою бортових комп'ютерів. Ці невеликі і відносно доступні платформи ідеально підходять для моніторингу пожеж і стихійних лих, спостереження за дикою природою і рослинністю.

### ***1.1.3 Датчики дистанційного зондування***

Для дистанційного зондування використовується датчик для отримання зображення. Наприклад, розглянуті вище системи, такі як літаки, супутники та БПЛА мають спеціалізовані платформи з датчиками.

Кожен тип датчика має свої переваги та недоліки. Коли метою є отримати зображення, повинно враховуватись такі фактори, як обмеження польоту, роздільна здатність зображення та покриття. Наприклад, супутники збирають дані в глобальному масштабі [3]. При цьому дрони краще підходять для польотів

на невеликих територіях. У свою чергу, літаки та гелікоптери займають середню позицію.

Існує два типи датчиків дистанційного зондування:

- Пасивні датчики;
- Активні датчики.

#### ***1.1.4 Методи дистанційного зондування***

- Фотозйомка;
- Сканерна зйомка;
- Радарна зйомка;
- Теплова зйомка;
- Спектрометрична зйомка;
- Лідарна зйомка [4].

### **1.2 Основи штучних нейронних мереж**

Штучні нейронні мережі (ШНМ) — це математичні моделі, які імітують роботу біологічних нейронних мереж у мозку людини. ШНМ базується на сукупності з'єднаних одиниць або вузлів, які називаються штучними нейронами, вони власне і моделюють нейрони біологічного мозку. Кожне з'єднання, як синапси в біологічному мозку, може передавати сигнал до інших нейронів. Штучний нейрон отримує сигнали, потім обробляє їх і може передавати їх тим нейронам, які з ним з'єднані. Його зазвичай представляють як деякий пороговий елемент, реакція якого визначається шляхом порівняння величини постсинаптичного потенціалу з деякою величиною порогу [5].

Покажемо структуру штучного нейрону на Рисунок 1.3.



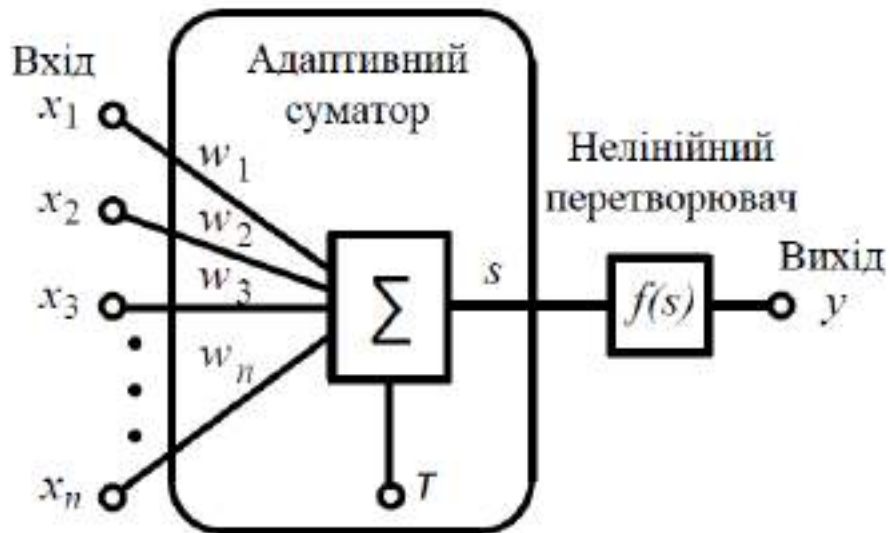


Рисунок 1.3 — Структура штучного нейрону

Штучні нейрони, незалежно від їх розташування та призначення, мають спільні складові. Розглянемо їх [5].

Перша складова — вагові коефіцієнти. Ваги визначають силу впливу кожного вхідного сигналу і можуть змінюватись під час навчання нейрона. Система ваг дозволяє нейрону враховувати важливість різних входів у своїй роботі і адаптуватися до змін у процесі навчання та роботи в мережі.

Друга складова — функція суматора. На початковому етапі роботи, нейрон проводить обчислення зваженої суми всіх вхідних сигналів. Результат множення цих векторів утворює загальний вхідний сигнал. Зазвичай функція суматора складніша і виконує різні більш комплексні операції, такі як вибір мінімуму, максимуму, середнього арифметичного, добутку або будь який інший нормалізуючий алгоритм. Також важливо буде зазначити, що функції суматора може виконувати додаткову обробку, що притаманне деяким нейронним мережам, це зветься функцією активації.

Третя складова — функція активації. У блоці функції активації загальна сума порівнюється з порогом і таким чином буде отримано значення виходу нейрону. Зазвичай в якості передатної функції використовують нелінійні функції, оскільки лінійні є обмеженими і вихід буде просто пропорційний до входу.

Четверта складова — масштабування. Результат, отриманий після функції активації, множиться на певний коефіцієнт масштабування і також має місце додавання зміщення (bias).

П'ята складова — вихідна функція. Аналогічно до біологічного нейрону, кожен штучний нейрон генерує один вихідний сигнал, який в свою чергу, передається до багатьох інших нейронів. Зазвичай, вихідний сигнал прямо пропорційний результату передатної функції. Проте в деяких структурах неймереж результати передатної функції можуть бути змінені для створення «конкуренції» між сусідніми нейронами.

Шоста складова — функція похибки. У більшості мереж, які використовують контрольоване навчання, обчислюється різниця між спродукованим, тобто виходом мережі і бажаним значенням. Ця різниця, яку називають біжучою похибкою або похибкою відхилення, піддається обробці згідно з архітектурою мережі.

Сьома складова — функція навчання. Основною метою функції навчання є налаштування вагових коефіцієнтів зв'язків на входах кожного елемента нейронної мережі відповідно до певного алгоритму з метою досягнення бажаного результату.

Шар, на який надходять зовнішні дані, є входним шаром. Шар, який видає кінцевий результат, є вихідним шаром. Між ними знаходяться нуль або більше прихованих шарів [5].

Будову ШНМ показано на Рисунок 1.4.

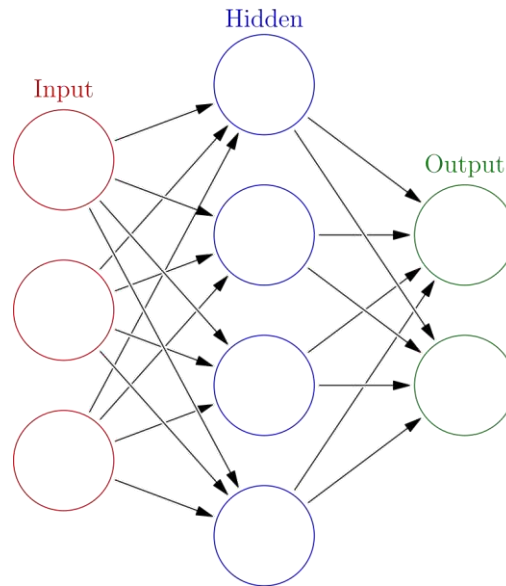


Рисунок 1.4 — Будова ШНМ з одним прихованим шаром [6]

### 1.3 Навчання штучної нейронної мережі

Вище в загальному було описано функцію навчання. Розглянемо навчання ШНМ більш детально.

По суті процес навчання можемо визначити як вибір архітектури мережі та ітеративної зміни (налаштування) сил синаптичних зв'язків, власне вагових коефіцієнтів, між усіма нейронами для виконання поставленого завдання.

Загальний алгоритм навчання зводиться до наступних пунктів.

1. Формується набір вхідних та вихідних даних для навчання
2. Синтезується нейронна мережа, кількість входів якої відповідає розмірності вхідних даних, а кількість виходів розмірності вихідних даних. Задається певна кількість прихованих шарів НМ та нейронів у них
3. Задається випадкове значення всіх вагових коефіцієнтів
4. Подаються на вхід мережі вхідні дані, проводиться прямий хід алгоритму навчання, тобто розраховується які дані будуть на виході нейронної мережі
5. Порівнюються вони із заданими, якщо не відповідають, то за певним правилом корегуються вагові коефіцієнти

6. Перебираються таким чином всі набори даних (після того, як всі набори даних навчальної вибірки були подані на вхід нейронної мережі вважається, що відбулася одна епоха навчання)
7. Перевіряється, чи правильно нейронна мережа прореагувала на випадковий набір з навчальної вибірки. Якщо так — завершується навчання, якщо ні, то повторити п. 4-7 [7].

Алгоритм навчання — це по суті правила за якими навчається штучна нейронна мережа. Відповідно не має такого алгоритму навчання НМ, який би підходив для всіх архітектур ШНМ. Такі алгоритми різняться між собою у підходах до налаштування вагових коефіцієнтів. Налаштування нейронної мережі, а точніше ваг зв'язків, відбувається за навчальною множиною.

Основними парадигмами навчання є:

1. Контрольоване (Навчання з вчителем), англ. supervised learning.
2. Неконтрольоване (Навчання без вчителя), англ. unsupervised learning
3. Навчання з підкріпленням, англ. reinforcement learning.

Покажемо схеми основних парадигм навчання на Рисунок 1.5.

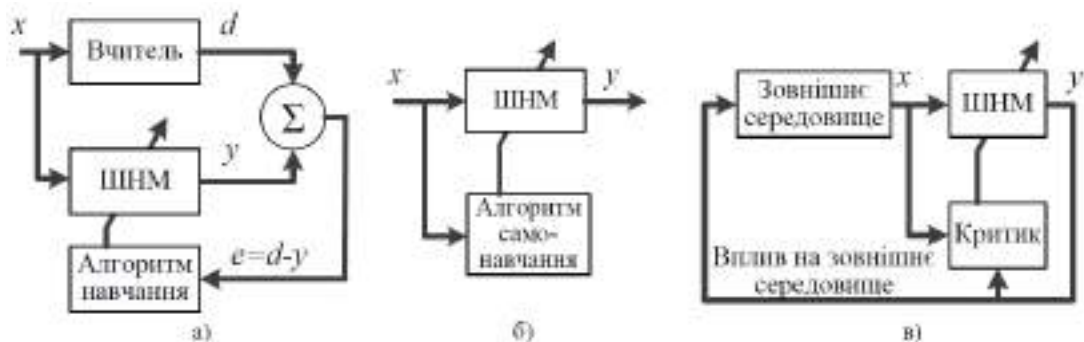


Рисунок 1.5 — Парадигми навчання а) Навчання з вчителем, б) Навчання без вчителя, в) Навчання з підкріпленням [7]

Однією з проблем нейронних мереж є перенавчання (англ. overfitting). Перенавчання відбувається тоді, коли модель НМ набуває здатності дуже добре класифікувати або прогнозувати на даних, які були включені в навчальну вибірку, але не так добре класифікує дані, на яких вона не навчалася [8]. Концепція перенавчання зводиться до того, що модель не здатна добре узагальнювати. Вона дуже добре вивчила особливості навчальної вибірки, але якщо ми

даємо моделі дані, які відрізняються від тих даних, що використовувалися під час навчання, вона не здатна точно передбачити вихідні дані.

Деякі способи уникнути перенавчання:

- Тренування на більшій кількості даних;
- Аугментація даних (Data augmentation);
- Додавання шуму до вхідних даних;
- Відбір ознак (Feature selection);
- Перехресне затвердження (Cross-validation);
- Спрощення даних;
- Регуляризація (Regularization);
- Асемблювання (Ensembling);
- Рання зупинка;
- Додавання dropout шарів [9].

#### **1.4 Алгоритм зворотного поширення помилки**

Алгоритм зворотного поширення помилки, відомий також як backpropagation являється алгоритмом навчання мереж прямого поширення. Цей ітеративний градієнтний алгоритм спрямований на зменшення помилки роботи багатошарової НМ та в підсумку отримання бажаного виходу. Його відносять до методів навчання з учителем, з цього випливає, що у навчальних прикладах мають бути задані саме цільові значення. Також цей алгоритм вважається одним із найбільш відомих алгоритмів в області машинного навчання [7].

В основі ідеї алгоритму лежить використання вихідної помилки нейронної мережі для розрахування величин корекції ваг нейронів в її прихованих шарах, яка визначається за формулою

$$E = \frac{1}{2} \sum_{i=1}^k (y - y')^2 \quad (1.1)$$

де  $k$  — кількість вихідних нейронів мережі;  
 $y$  — цільове значення;

$y^*$  — фактичне вихідне значення.

Даний алгоритм базується на покроковому навчанні і використовує ітеративний підхід, при цьому після того, як один навчальний приклад було подано на вхід, відбувається коригування вагових коефіцієнтів нейронів мережі [10].

Під час кожної ітерації реалізовується спочатку прямий прохід мережі, а потім зворотній. Вектор подається на входи мережі і проходить до її виходів, формуючи вже вихідний вектор, з урахуванням поточного стану вагових коефіцієнтів. Після цього вираховується помилка НМ шляхом знаходження різниці між фактичним та бажаним значеннями [10]. Далі відбувається зворотній прохід і ця розрахована помилка проходить від виходу НМ до її входів з подальшою корекцією вагових коефіцієнтів згідно з наступним правилом:

$$\Delta \omega_{j,i}(n) = -\eta \frac{\partial E_{av}}{\partial \omega_{ij}} \quad (1.2)$$

де  $\frac{\partial E_{av}}{\partial \omega_{ij}}$  — похідна помилки  $E$  по вагам  $i$ -го зв'язку  $j$ -го нейрону;

$\eta$  — параметр швидкості навчання;

$\Delta \omega_{j,i}$  — величина кроку корекції.

Параметр швидкості навчання дозволяє додатково керувати величиною кроку корекції з ціллю більш точного налаштування на мінімум помилки і підбирається експериментально в процесі навчання, змінюючись в інтервалі від 0 до 1 [10].

Вихідна сума  $j$ -го нейрона рівна

$$S_j = \sum_{i=1}^n \omega_{ij} x_i \quad (1.3)$$

де  $x_i$  — значення компоненти  $i$ -го вхідного вектора;

$\omega_{ij}$  — вага  $i$ -го зв'язку  $j$ -го нейрона.

В такому разі похідна помилки буде рівною

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial S_j} \frac{\partial S_j}{\partial \omega_{ij}} = x_i \frac{\partial E}{\partial S_j} \quad (1.4)$$

З даної формули випливає, що диференціал  $\partial S_j$  активаційної функції нейронів мережі не повинен дорівнювати нулю і має існувати в будь-якій точці. Таким чином активаційна функція має бути диференційованою. Тому для методу backpropagation використовують сигмоїдальні функції активації такі як логістична активаційна функція або гіперболічний тангенс. Простіше кажучи, алгоритм може працювати виключно з функціями активації, які мають похідну.

Алгоритм використовує метод стохастичного градієнтного спуску, тобто відбувається рух у багатовимірному просторі вагових коефіцієнтів у напрямку протилежному градієнту функції помилки з метою досягти мінімального значення цієї функції.

Зазвичай навчання продовжують не до точного налаштування мережі на мінімум функції помилки, а доти, доки не буде досягнуто достатньо точного його наближення. Це дасть змогу, з одного боку, зменшити кількість ітерацій навчання, а з іншого — уникнути перенавчання мережі [11].

## **1.5 Архітектури штучних нейронних мереж**

### ***1.5.1 Штучна нейронна мережа прямого поширення***

Нейронна мережа прямого поширення (англ. Feedforward neural network) вид нейронної мережі, в якій сигнали поширюються в одному напрямку, починаючи від вхідного шару нейронів, через приховані шари до вихідного шару і на вихідних нейронах отримується результат опрацювання сигналу. В мережах такого виду немає зворотніх зв'язків. Найпростішим прикладом мережі прямого поширення є перцептрон.

Приклад НМ прямого поширення зображений на Рисунок 1.6.

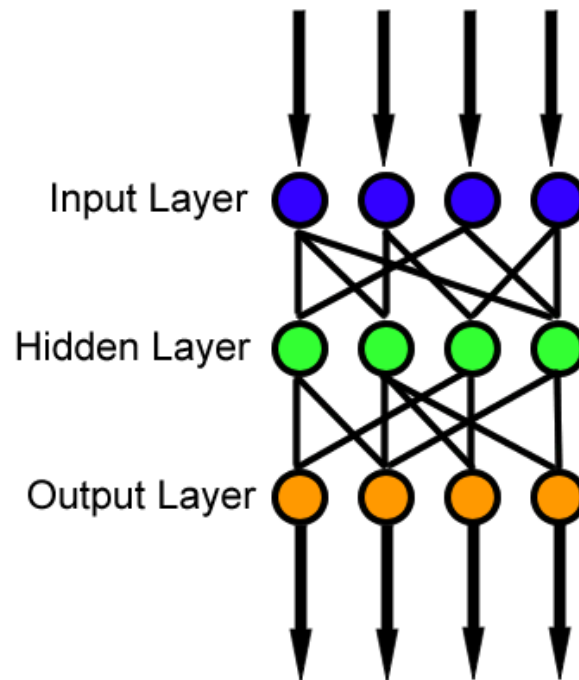


Рисунок 1.6 — НМ прямого поширення

Основними компонентами нейронної мережі прямого поширення є:

1. Вхідний шар.
2. Приховані шари.
3. Вихідний шар.
4. Ваги та зсуви.

### ***1.5.2 Рекурентна нейронна мережа***

Рекурентні нейронні мережі (РНМ, англ. recurrent neural networks, RNN) — це клас штучних нейронних мереж, у якому з'єднання між вузлами утворюють граф орієнтований у часі. Це створює внутрішній стан мережі, що дозволяє їй проявляти динамічну поведінку в часі. На відміну від нейронних мереж прямого поширення, РНМ можуть використовувати свою внутрішню пам'ять для обробки довільних послідовностей входів. Це робить їх застосовними до таких задач, як розпізнавання несеgmentованого неперервного рукописно го тексту та розпізнавання мовлення [12].

Представлення рекурентної НМ в двох варіантах зображене на Рисунок 1.7.



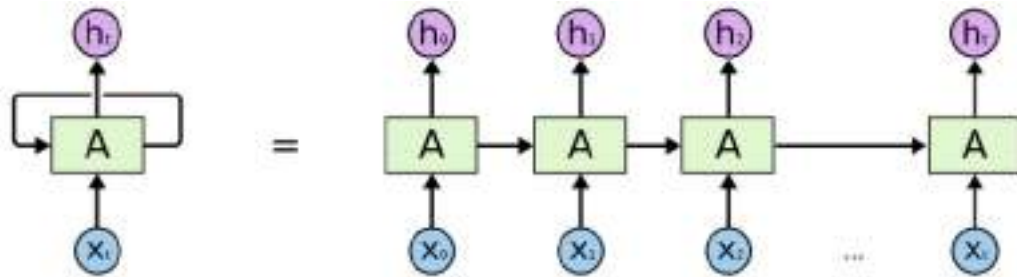


Рисунок 1.7 — Ліворуч: скорочені позначення, що часто використовуються для демонстрації RNN, праворуч: Розгорнуте представлення для RNN

Деякі види рекурентних нейронних мереж:

- Повнозв'язана нейронна мережа (англ. Fully recurrent);
- Мережа Елмана (англ. Elman networks);
- Мережа Хопфілда (англ. Hopfield network);
- Двонаправлена асоціативна пам'ять (англ. Bidirectional associative memory, BAM);
- Незалежна РНМ (англ. Independently RNN, IndRNN);
- Рекурсивна нейронна мережа (англ. Recursive neural network);
- Довга короткочасна пам'ять (англ. Longshort-termmemory, LSTM);
- Нейронний стискач історії (англ. Neural history compressor) [12].

### ***1.5.3 Згорткова нейронна мережа***

Згорткові нейронні мережі (англ. convolutional neural network CNN, ConvNet) в машинному навчанні — це клас глибоких штучних нейронних мереж прямого поширення, який успішно застосовувався до аналізу візуальних зображень [13].

Згорткова нейронна мережа складається з декількох шарів, таких як вхідний шар, згортковий шар, шар пулінгу (pooling layer) і повнозв'язні шари.

Архітектура такої мережі зображена на Рисунок 1.8.

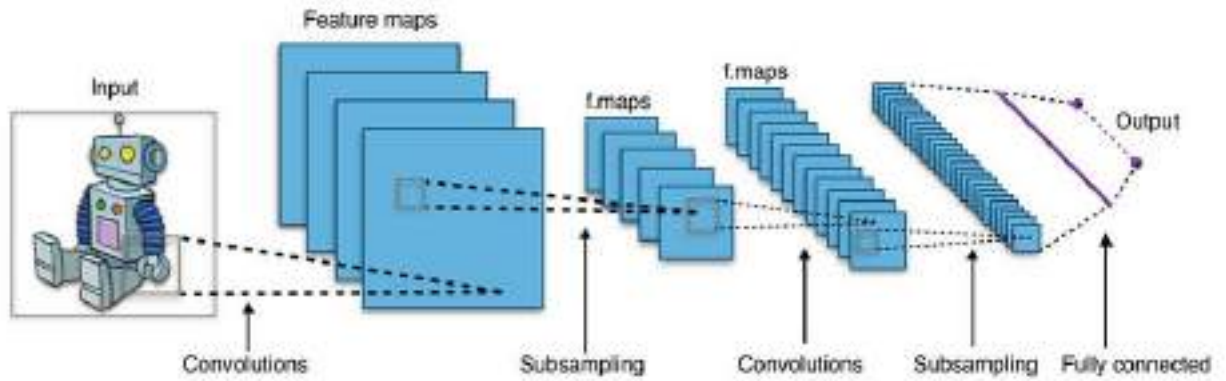


Рисунок 1.8 — Архітектура CNN

По суті CNN — це комбінація двох основних функціональних блоків:

Блок згортки — складається з шару згортки та шару пулінгу. Цей шар є основним компонентом отримання ознак.

Повнозв'язний блок — складається з повнозв'язної простої архітектури нейронної мережі. Цей шар виконує завдання класифікації на основі вхідних даних від блоку згортки.

Інтенсивність обчислень збільшиться, коли зображення досягнуть розмірів, припустимо, 8K (7680×4320). Роль CNN полягає в тому, щоб зменшити зображення до форми, яку легше обробляти, не втрачаючи особливостей, які є критично важливими для отримання хорошого передбачення. Це важливо, щоб архітектура не тільки добре вивчала особливості, але й масштабувалася до великих наборів даних [14].

Переваги та недоліки згорткових нейронних мереж

Переваги

1. Менша кількість вагових коефіцієнтів у порівнянні з повнозв'язною мережею
2. Краще проходить узагальнення інформації
3. Висока степінь розпаралелення розрахунків
4. Стійкість до зсуву зображення, часткова стійкість до повороту зображення

Недоліки

1. Складна архітектура з купою невідомих параметрів, які обираються емпіричним шляхом

## **1.6 Висновки до першого розділу**

В даному розділі було розглянуто засоби та методи зондування, їх історію, розглянуті основи та засади на яких будуються нейронні мережі також були розглянуті певні їх базові архітектури.

Таким чином, перший розділ дозволив більш детально вивчити та розібрати основні принципи дистанційного зондування, ознайомитися з методами та технологіями отримання та аналізу даних, а також засвоїти основні концепції штучних нейронних мереж та їхні застосування у вирішенні складних завдань, що буде корисним для подальших досліджень та практичного застосування.

## 2 АХИТЕКТУРИ НМ DEEP LEARNING

### 2.1 Мережі LeNet-5 та AlexNet

LeNet-5 — це згорткова архітектура нейронної мережі, створена Яном Леканном у 1998 році. Вона включає 7 шарів, на додачу до вхідного шару, в якому є навчальні параметри, звані вагами.

Архітектуру LeNet-5 CNN зображено на Рисунок 2.1.

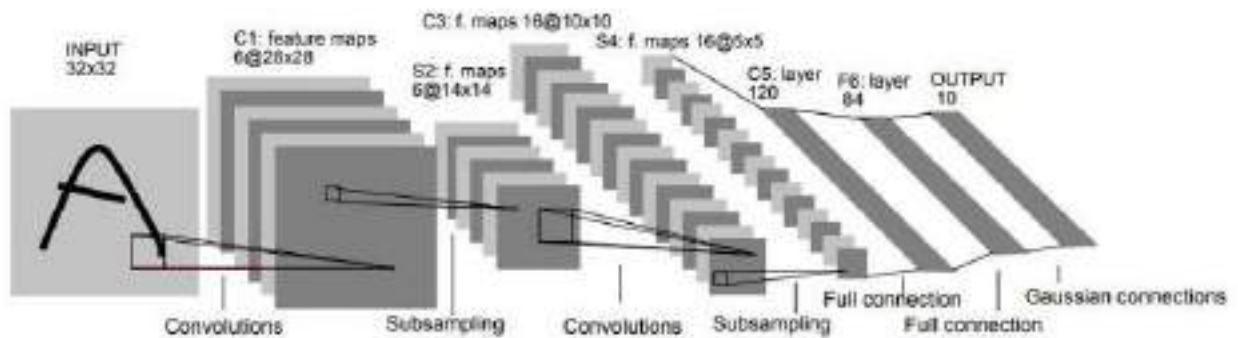


Рисунок 2.1 — Архітектура мережі LeNet-5

В архітектурі нейронної мережі, показаній вище:

1. Вхідними даними є зображення у відтинках сірого розміром  $32 \times 32$ , яке проходить через перший шар згортки C1. Шар C1 являє собою згортковий шар із шістьма картами ознак, розмір яких  $28 \times 28$ ;
2. Шар S2 — це шар підвибірки з шістьма картами ознак, розмір яких  $14 \times 14$ , з розміром фільтра  $2 \times 2$  і кроком два;
3. Шар C3 являє собою згортковий шар із шістнадцятьма характеристичними картами, розмір яких  $10 \times 10$ ;
4. Шар S4 є шаром підвибірки з шістнадцятьма характеристичними картами, розмір яких  $5 \times 5$  з розміром фільтра  $2 \times 2$  і кроком 2. Цей шар такий самий, як і другий шар (S2), за винятком того, що він має 16 карт ознак, тому вихідні дані будуть зменшені до  $5 \times 5 \times 16$ ;
5. Шар C5 являє собою згортковий шар зі 120 картами ознак, розмір яких становить  $1 \times 1$ . Кожна з 120 одиниць у C5 з'єднана з усіма 400 вузлами ( $5 \times 5 \times 16$ ) четвертого шару S4.;

6. Шар F6 містить 84 одиниці і повністю пов'язаний зі згортковим шаром C5.

7. Вихідний шар з активаційною функцією softmax є повністю підключений з 10 можливими значеннями, що відповідають цифрам від 0 до 9.

Архітектура AlexNet схожа зі мережею LeNet, яка була розглянута вище. Однак у AlexNet більше фільтрів в шарі та вкладених згорткових шарів. AlexNet містить 8 шарів з вагами (5 згорткових шарів та 3 повнозв'язних)

Архітектуру мережі показано на Рисунок 2.2.

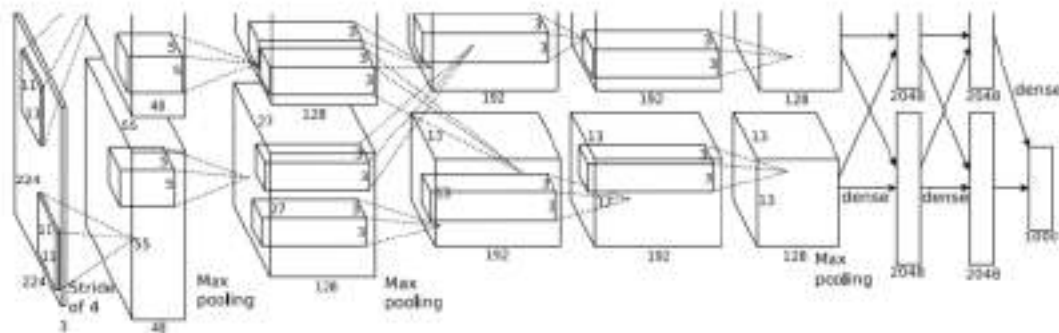


Рисунок 2.2 — Архітектура AlexNet

В кінці кожного шару виконується функція активації ReLu, за винятком останнього, який на виході реалізує активаційну функцію softmax, яка формує розподіл 1000 міток класів. Dropout застосовується в перших двох повнозв'язних шарах. Як показано на рисунку вище, також застосовується Max pooling після першого, другого та п'ятого згорткових шарів. Ядра другого, четвертого та п'ятого згорткових шарів з'єднуються лише з тими картами ядер попереднього шару, які знаходяться на тому ж графічному процесорі. Ядра третього згорткового шару з'єднані з усіма картами ядер другого шару. Нейрони в повнозв'язних шарах з'єднані з усіма нейронами попереднього шару [15].



Рисунок 2.3 — Порівняння згортки, підвибірки (pooling), «щільних» шарів

## 2.2 Мережі VGG-16 VGG-19

VGG, також відома як VGGNet — класична архітектура згорткових нейронних мереж (CNN). VGG було розроблено для збільшення глибини таких ШНМ з метою підвищення продуктивності моделі. VGG розшифровується як Visual Geometry Group; це стандартна архітектура глибокої згорткової нейронної мережі (CNN) з декількома шарами. "Глибока" означає кількість шарів: VGG-16 або VGG-19 складається з 16 і 19 згорткових шарів відповідно.

Архітектура VGG лежить в основі революційних моделей розпізнавання об'єктів. Модель VGG, або VGGNet, яка підтримує 16 шарів, також називається VGG16 — це модель згорткової нейронної мережі, запропонована А. Зіссерманом і К. Симоняном з Оксфордського університету. Ці дослідники опублікували свою модель у науковій роботі під назвою "Very Deep Convolutional Networks for Large-Scale Image Recognition" [16]. Модель VGG16 досягає майже 92,7% точності в топ-5 тестах ImageNet. ImageNet - це набір даних, що складається з понад 14 мільйонів зображень, які належать до майже 1000 класів. Більше того, це одна з найпопулярніших моделей, представлених

на ILSVRC-2014. Вона замінює великі фільтри розміром ядра на кілька фільтрів розміром  $3 \times 3$  один за одним, таким чином досягаючи значних покращень порівняно з AlexNet.

Архітектуру ШНМ VGG16 показано на Рисунок 2.4.

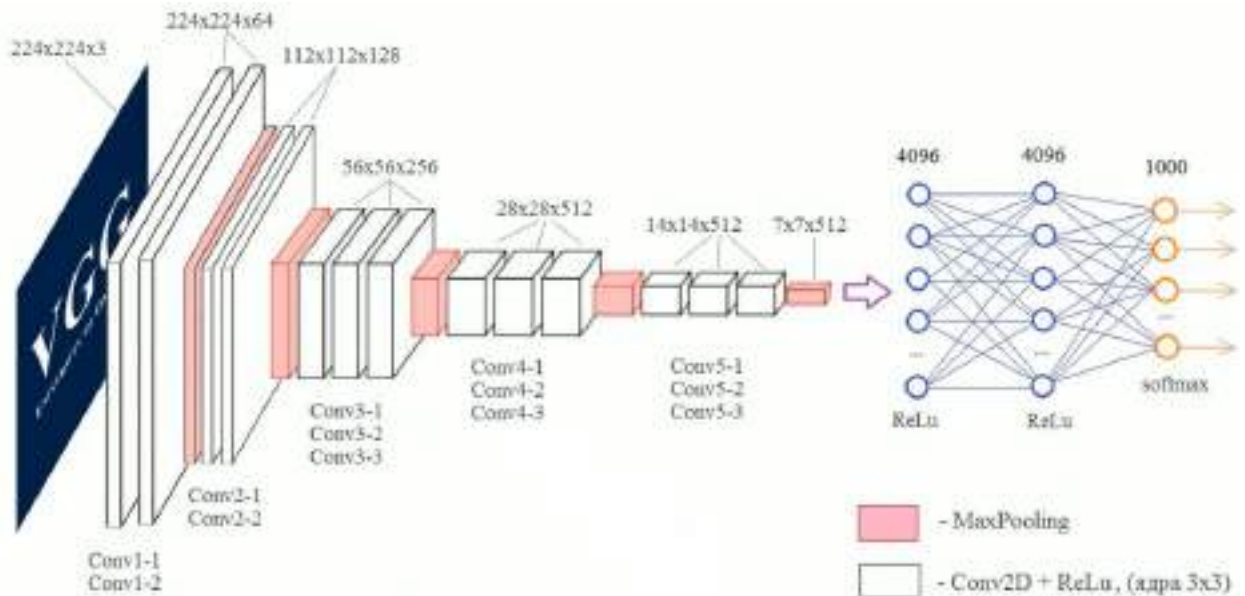


Рисунок 2.4 — Архітектура VGG16

1. Як зазначалось вище, цифра 16 у VGG16 означає кількість шарів, які мають ваги. У VGG16 є тринадцять згорткових шарів, п'ять шарів MaxPooling і три «щільні» шари, що в сумі складає 21 шар, але мережа має лише шістнадцять вагових шарів, тобто шар з параметрами, що навчаються.
2. VGG16 приймає розмір вхідного тензора 224 на 244 з 3 каналами RGB
3. Унікальність VGG16 полягає в тому, що замість великої кількості гіперпараметрів вони зосередилися на використанні шарів згортки фільтра  $3 \times 3$  з кроком 1 і завжди використовували однакові шари padding та шару MaxPool фільтра  $2 \times 2$  з кроком 2.
4. Шари згортки та MaxPooling послідовно розташовані в усій архітектурі
5. Шар Conv-1 має 64 фільтри, Conv-2 - 128 фільтрів, Conv-3 - 256 фільтрів, Conv-4 і Conv-5 - 512 фільтрів, тобто вони на виході дають 64, 128, 256, 512 каналів відповідно.

- б. За стеком згорткових шарів слідують три повнозв'язних (Fully-Connected, FC) шари: перші два мають по 4096 каналів, третій виконує 1000 вихідну ILSVRC класифікацію і, таким чином, містить 1000 каналів (по одному для кожного класу). Останнім шаром є шар з функцією активації softmax.

Концепція моделі VGG19 (також VGGNet-19) така ж, як і у VGG16, за винятком того, що вона підтримує 19 шарів. Цифри 16 і 19 означають кількість згорткових шарів моделі. Це означає, що VGG19 має на три згорткових шари більше, ніж VGG16 [17].

Наглядне порівняння моделей VGG16 та VGG19 показане на Рисунок 2.5.

VGG-16	VGG-19
input (224x244x3)	
conv3-64 conv3-64	conv3-64 conv3-64
maxpool	
conv3-128 conv3-128	conv3-128 conv3-128
maxpool	
conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256
maxpool	
conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool	
conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool	
FC-4096	
FC-4096	
FC-1000	
softmax	

Рисунок 2.5 — Порівняння мереж VGG16 та VGG19

Зі збільшенням кількості шарів у CNN збільшується і здатність моделі відповідати більш складним функціям. Отже, більша кількість шарів має за-



безпечити кращу продуктивність. Проте це не слід плутати зі ШНМ, де збільшення кількості шарів не обов'язково призводить до кращої продуктивності. Тепер питання в тому, чому б не використовувати VGGNet з більшою кількістю шарів, наприклад, VGG20, або VGG50, або VGG100? Осць тут і виникає проблема. Ваги нейронної мережі оновлюються за допомогою алгоритму зворотного поширення, який вносить невеликі зміни в кожне значення ваги, щоб зменшити втрати моделі. Однак, оскільки градієнт продовжує повертатися назад до початкових шарів, значення продовжує збільшуватися на кожен локальний градієнт. Це призводить до того, що градієнт стає все меншим і меншим, що робить зміни в початкових шарах дуже малими. Це, в свою чергу, значно збільшує час навчання. Проблема можна вирішити, якщо локальний градієнт стає рівним 1.

### 2.3 Мережа ResNet

Тут на допомогу приходить ResNet, оскільки він досягає цього за допомогою функції тотожності (Identity function). Отже, при зворотному поширенні градієнт не зменшується, оскільки локальний градієнт дорівнює 1.

Глибокі залишкові мережі (ResNet), такі як популярна модель ResNet-50, є ще одним типом архітектури згорткової нейронної мережі (CNN), яка має 50 шарів. Залишкова нейронна мережа використовує вставку коротких з'єднань для перетворення звичайної мережі в її аналог із залишковою мережею.

У порівнянні з VGGNet, ResNet менш складні, оскільки мають менше фільтрів. ResNet, яку також називають залишковою мережею, вирішує проблему затухаючого градієнта. В цьому питанні ключовим рішенням стали пропускні з'єднання (англ. skip connections або shortcut connections). У ResNet архітектурах вони також допомогли розв'язати іншу проблему — проблему деградації точності. Проблема деградації точності полягає в тому, що під час збільшення глибини мережі точність спочатку зростає, потім, всупереч очікуванням, ви-

ходить на плато, а потім навіть починає знижуватися аж до повної втрати здатності мережі до навчання. Це також одна з найважливіших причин, чому ResNet реалізується в таких версіях, як ResNet50, ResNet101 і ResNet152.

Фрагмент Skip connection показаний на Рисунок 2.6.

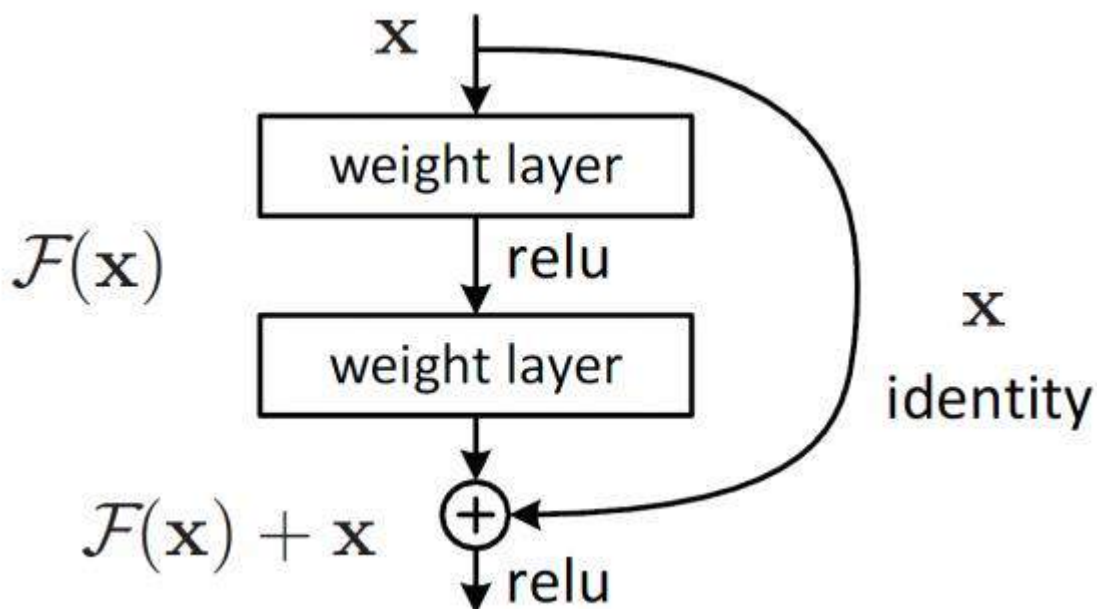


Рисунок 2.6 — Skip connection [18]

ResNet з 34 шарів у порівнянні зі звичайною 34-шаровою CNN показано на Рисунок 2.7.

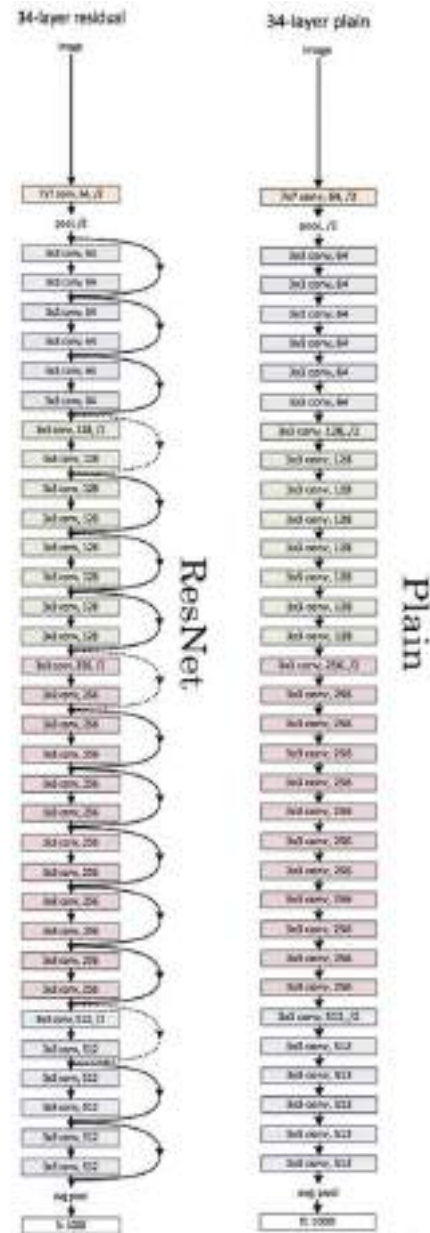


Рисунок 2.7 — ResNet з 34 шарами зліва, звичайна мережа з 34 шарами справа [19]

## 2.4 Мережі R-CNN, Fast R-CNN та Faster R-CNN

### 2.4.1 R-CNN

Іншою невід'ємною частиною комп'ютерного зору є виявлення об'єктів. Виявлення об'єктів допомагає в виявленні транспортних засобів, спостереженні тощо. Різниця між алгоритмами виявлення об'єктів і алгоритмами класифікації полягає в тому, що в алгоритмах виявлення ми намагаємося намалювати обмежувальну рамку навколо об'єкта, який нас цікавить, щоб знайти його

на зображенні. Крім того, у випадку виявлення об'єктів не обов'язково малювати лише одну рамку, на зображенні може бути багато рамок, що представляють різні об'єкти інтересу.

Головна причина, чому не можна вирішити цю проблему стандартною згортковою нейронною мережею з наступним повнозв'язним шаром, полягає в тому, що довжина виходу змінюється, оскільки кількість об'єктів, які нас цікавлять на зображенні, не є постійною. Простим підходом до вирішення цієї проблеми було б виділити окремі області інтересу на зображенні та використовувати ШНМ для класифікації присутності об'єкта в кожній області. Проте проблема полягає в тому, що об'єкти інтересу можуть розташовуватися по-різному на зображенні та мати різне співвідношення сторін. Це означає, що нам доведеться виділити велику кількість областей, що може призвести до збільшення обчислювальної складності. Тому були розроблені алгоритми, такі як R-CNN, YOLO та інші, які дозволяють ефективно виявляти об'єкти на зображеннях [20].

Архітектура мережі R-CNN (Regions With CNNs) була розроблена командою з UC Berkley для застосування Convolution Neural Networks до задачі розпізнавання об'єктів (object detection). Існуючі на той момент підходи до розв'язання таких завдань наблизилися до максимуму своїх можливостей, і значно поліпшити їхні показники не вдавалосьь.

CNN добре показували себе в класифікації зображень, і в даній мережі вони, по суті, були застосовані для того ж самого. Для цього на вхід CNN подавали не все зображення повністю, а попередньо виділені іншим способом області, на яких, ймовірно, мали б бути якісь об'єкти. На той момент таких підходів було кілька, автори вибрали Selective Search [20].

Щоб обійти проблему вибору величезної кількості областей, був запропонований метод, в якому використовується вибірковий пошук для вилучення лише 2000 областей із зображення, і він назвав їх region proposals. Таким чином, замість того, щоб намагатися класифікувати величезну кількість областей, можна було просто працювати з 2000 областями.

Принцип роботи R-CNN показаний на Рисунок 2.8.

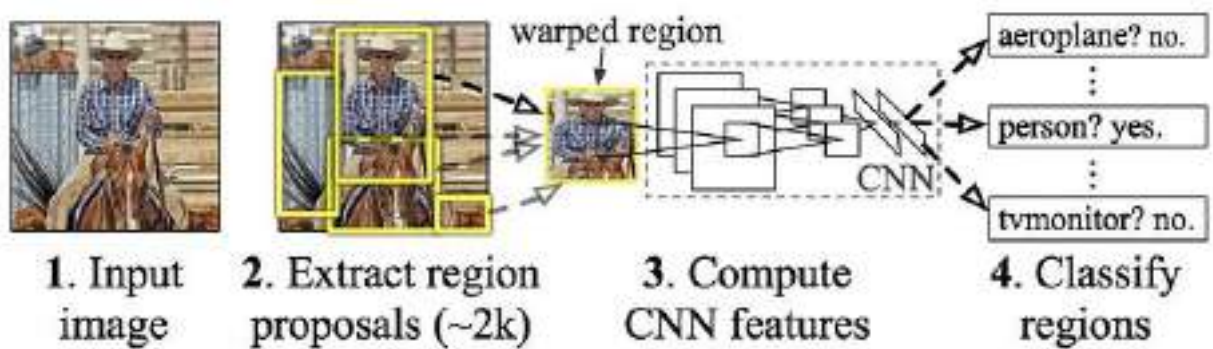


Рисунок 2.8 — R-CNN: Области з якими працює CNN [20]

Основними проблемами R-CNN було те, що навчання мережі все ще займало досить тривалий час, оскільки доводилось класифікувати 2000 пропозицій по області на одне зображення; також метод не може бути реалізований в реальному часі.

#### 2.4.2 Мережа Fast R-CNN

Той самий автор R-CNN вирішив деякі недоліки R-CNN, щоб побудувати швидший алгоритм виявлення об'єктів, який отримав назву Fast R-CNN. Підхід схожий на алгоритм R-CNN. Але замість того, щоб подавати пропозиції по області на CNN, подавалось вхідне зображення на CNN, щоб згенерувати згорткову карту ознак. На згортковій карті ознак визначається область пропозицій, вона розбивається на квадрати і, використовуючи шар RoI (Region of interest) pooling, перетворює їх у фіксований розмір, щоб його можна було подати на повнозв'язний шар. На основі вектора ознак RoI використовувався шар softmax для прогнозування класу запропонованої області, а також значень зсуву для обмежувальної рамки [21].

Принцип функціонування та побудови Fast R-CNN зображено на **Помилка! Джерело посилання не знайдено..**

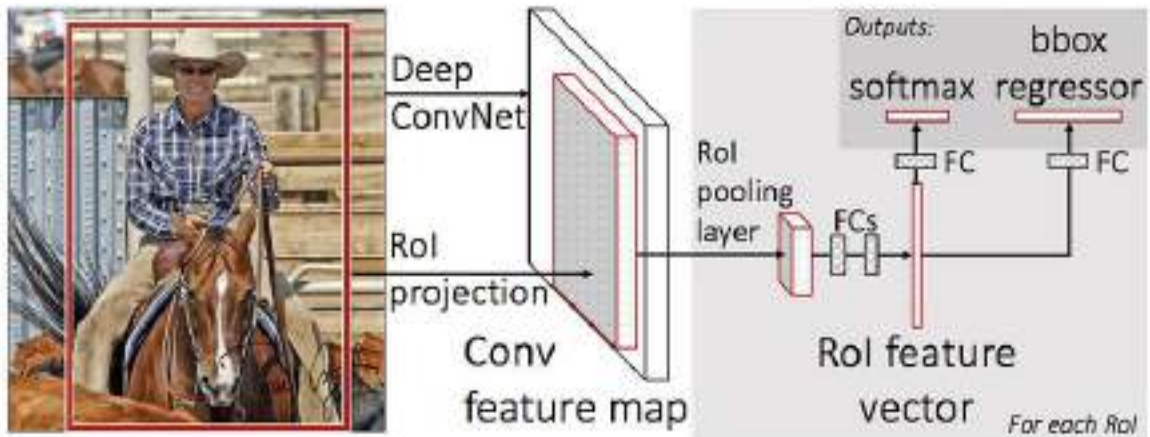


Рисунок 2.9 — Принцип побудови Fast R-CNN [21]

Причина, чому Fast R-CNN швидший за R-CNN, полягає в тому, що вам не потрібно щоразу подавати 2000 region proposals на згорткову нейронну мережу. Замість цього операція згортки виконується лише один раз для кожного зображення і на її основі генерується карта ознак.

### 2.4.3 Мережа *Faster R-CNN*

Обидва алгоритми (R-CNN та Fast R-CNN) використовували вибіркового пошуку для знаходження пропозицій по області. Вибірковий пошук є повільним і трудомістким процесом, що впливає на продуктивність мережі. Тому було розроблено алгоритм виявлення об'єктів, який усуває алгоритм вибіркового пошуку і дозволяє мережі вивчати пропозиції по області. Було запропоновано обчислювати області не за початковим зображенням, а знову ж таки за картою ознак, отриманих із CNN. Для цього було додано модуль під назвою Region Proposal Network (RPN) [22].

Функціонування алгоритму Faster R-CNN, його архітектура зображена на рисунку 2.10.

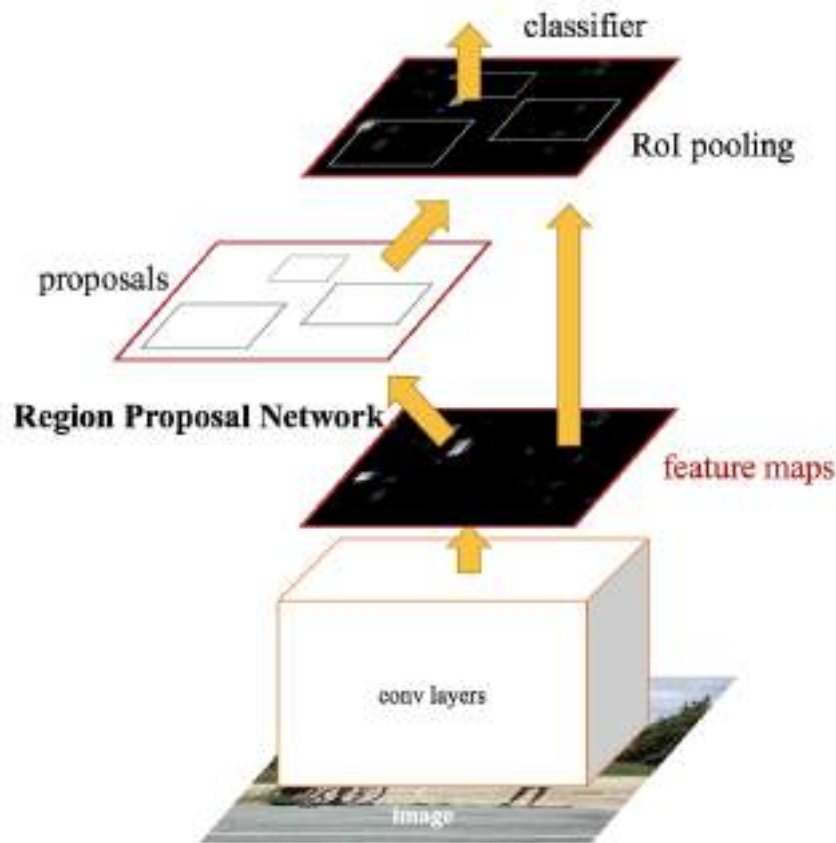


Рисунок 2.10 — Архітектура алгоритму Faster R-CNN [22]

Подібно до Fast R-CNN, зображення подається на вхід згорткової мережі, яка створює згорнуту карту ознак. Замість того, щоб використовувати алгоритм вибіркового пошуку на карті ознак для ідентифікації регіональних пропозицій, використовується окрема мережа для прогнозування пропозицій по області. Прогнозовані пропозиції по області потім змінюють форму за допомогою шару об'єднання RoI, який потім використовується для класифікації зображення в межах запропонованої області і прогнозування значень зміщення для обмежувальних рамок.

## 2.5 You Only Look Once — YOLO

You Only Look Once (YOLO) — це сучасний алгоритм виявлення об'єктів у реальному часі, представлений у 2015 році науковій роботі "You Only Look Once: Unified, Real-Time Object Detection". Автори розглядають проблему виявлення об'єктів як регресійну задачу, а не як задачу класифікації, просторово

розділяючи обмежувальні рамки і пов'язуючи ймовірності з кожним із виявлених зображень за допомогою однієї згорткової нейронної мережі (CNN).

Всі попередні алгоритми виявлення об'єктів використовують області для локалізації об'єкта на зображенні. Мережа не переглядає зображення повністю. Замість цього вона розглядає ті частини зображення, які мають високу ймовірність містити об'єкт. YOLO (You Only Look Once) — це алгоритм виявлення об'єктів, який значно відрізняється від алгоритмів, що базуються на регіонах, розглянутих вище. У YOLO одна згорткова мережа прогнозує обмежувальні рамки, граничні області та ймовірності класів для кожної області або ж рамки [23].

Принцип роботи YOLO полягає в наступному підході. Береться зображення і розбивається на сітку  $S \times S$ , в межах кожної сітки виділяється  $m$  рамок. Для кожного з них мережа виводить ймовірність класу  $C$  та значення зсуву для нього.

Ці прогнози розраховуються за наступною формулою:

$$S \times S \times (m \cdot 5 + C), \quad (2.1)$$

де  $m$  — кількість рамок;

$S$  — розмір сторони сітки;

$C$  — ймовірність класу.

Рамки, ймовірність класу яких перевищує порогове значення, обираються і використовуються для визначення місцезнаходження об'єкта на зображенні.

Розподіл рамок за ймовірністю класу показано на Рисунок 3.11.



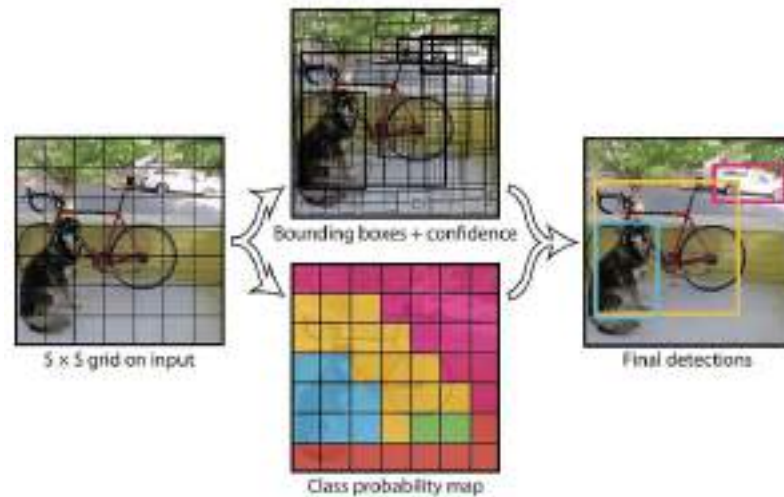


Рисунок 2.11 — Розподіл рамок за ймовірністю класу [23]

Архітектура працює наступним чином:

1. Зміна розміру вхідного зображення до 448x448 перед проходженням через згорткову мережу.
2. Спочатку застосовується згортка 1x1 для зменшення кількості каналів, за якою слідує згортка 3x3 для отримання кубоїдального результату.
3. Функція активації — ReLU, за винятком останнього шару, який використовує лінійну функцію активації.
4. Деякі додаткові методи, такі як батч-нормалізація та dropout, відповідно нормалізують модель та запобігають її надмірному перенавантаженню.

Архітектуру мережі зображено на Рисунок 2.12.

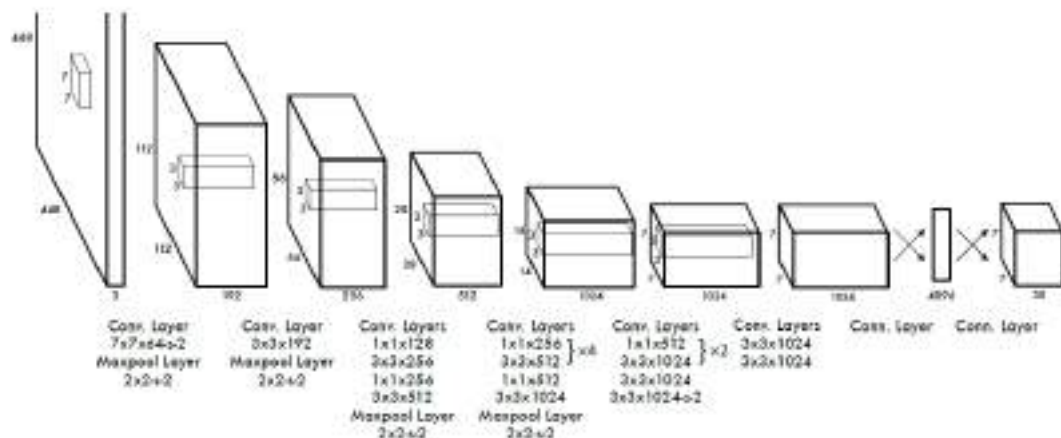


Рисунок 2.12 — Архітектура мережі [23]

Мережа, яка розглядається має кілька переваг над системами на основі класифікаторів. Вона розглядає все зображення під час тестування, тому її прогнози базуються на глобальному контексті зображення. Вона також робить прогнози за допомогою однієї оцінки мережі, на відміну від таких систем, як R-CNN, яким потрібні тисячі оцінок для одного зображення. Це робить його надзвичайно швидким, більш ніж в 1000 разів швидшим за R-CNN і в 100 разів швидшим за Fast R-CNN.

YOLO працює на порядок швидше (45 кадрів за секунду), ніж інші алгоритми виявлення об'єктів. Обмеженням алгоритму YOLO є те, що він має проблеми з невеликими об'єктами на зображенні, наприклад, у нього можуть виникнути труднощі з виявленням зграї птахів. Це пов'язано з просторовими обмеженнями алгоритму.

З моменту першої появи YOLO у 2015 році, він сильно еволюціонував, з'явилися різні версії:

1. YOLO або YOLOv1, відправна точка. Перша версія YOLO змінила правила гри у сфері виявлення об'єктів завдяки своїй здатності швидко та ефективно розпізнавати об'єкти.
2. YOLOv2 було створено у 2016 році з метою зробити модель YOLO кращою, швидшою та сильнішою.
3. Для створення YOLOv3 було проведено поступове вдосконалення YOLOv2. Зміни в основному стосуються нової мережевої архітектури: Darknet-53. Це нейронна мережа зі 106 нейронів, з мережами підвищеної вибірки та залишковими блоками. Вона набагато більша, швидша і точніша порівняно з Darknet-19, яка є основою YOLOv2.
4. Ця версія YOLO має оптимальну швидкість і точність виявлення об'єктів у порівнянні з усіма попередніми версіями та іншими сучасними детекторами об'єктів. YOLOv4 перевершує YOLOv3 у FPS та за швидкістю на 10% і 12% відповідно.

5. YOLOv5, на відміну від інших версій, не має опублікованої наукової роботи, і це перша версія YOLO, яка була реалізована в Pytorch, а не в Darknet.
6. Фреймворк YOLOv6 (MT-YOLOv6), призначений для промислових застосувань, з ефективним дизайном та високою продуктивністю, був випущений китайською компанією Meituan. Написана на Pytorch, ця нова версія не була частиною офіційного YOLO, але все одно отримала назву YOLOv6. YOLOv6 показала надзвичайно високі результати порівняно з попередніми версіями YOLO з точки зору точності та швидкості
7. YOLOv7 була випущена в липні 2022 року та задає нові сучасні тенденції для детекторів об'єктів в реальному часі. Ця версія робить значний крок у сфері виявлення об'єктів, і вона перевершила всі попередні моделі за точністю та швидкістю.
8. Ultralytics, компанія, що розробляла YOLOv5, випустила YOLOv8 у січні 2023 року [24] [25].

## 2.6 Single Shot Detector

SSD призначена для виявлення об'єктів у реальному часі. Faster R-CNN використовує мережу пропозицій по області для створення граничних рамок і використовує ці рамки для класифікації об'єктів. Хоча цей метод вважається передовим за точністю, весь процес виконується зі швидкістю 7 кадрів на секунду. Це набагато нижче, ніж потрібно для обробки в реальному часі. SSD прискорює процес, усуваючи потребу в мережі пропозицій по області. Щоб відновити втрачену точність, SSD застосовує кілька вдосконалень, зокрема, мультимасштабні функції (multi-scale features) та default boxes. Ці вдосконалення дозволяють SSD відповідати точності Faster R-CNN, використовуючи зображення з нижчою роздільною здатністю, що ще більше підвищує швидкість. Згідно з наведеним нижче порівнянням, він досягає швидкості обробки в реальному часі і навіть перевершує точність Faster R-CNN [26].

Порівняння ефективності різних методів зображено на Рисунок 2.13.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

Рисунок 2.13 — Результати тесту ефективності різних методів [26]

Виявлення об'єктів SSD складається з 2 частин:

1. Отримати карти об'єктів
2. Застосування фільтрів згортки для виявлення об'єктів.

Архітектура мережі SSD зображена на Рисунок 2.14.

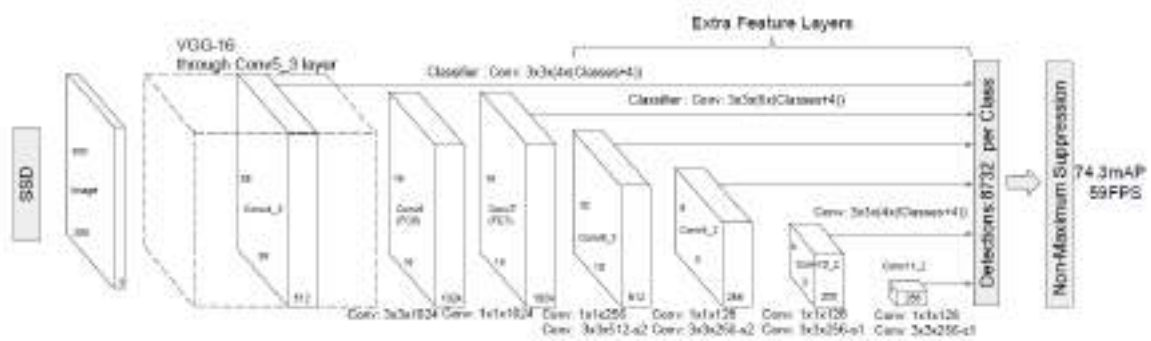


Рисунок 2.14 — Архітектура мережі SSD [26]

SSD використовує VGG16 для отримання карт ознак. Потім він виявляє об'єкти за допомогою шару Conv4\_3. Можемо бачити на рисунку вище Conv4\_3 розміром  $8 \times 8$  у просторі (він має бути  $38 \times 38$ ). Для кожної комірки (яку також називають локацією) програма робить 4 прогнози щодо об'єктів.

## 2.7 Порівняння ефективності архітектур для виявлення об'єктів

В попередніх пунктах було розглянуто досить велику кількість архітектур та підходів до вирішення задачі розпізнавання об'єктів на зображеннях. Однак було вирішено зупинитися на розгляді найпопулярніших та тих, що показують найкращу ефективність, а саме: Faster R-CNN, YOLO та SSD.

Нижче наведено діаграму розкиду швидкості і точності основних методів виявлення об'єктів (R-CNN, Fast R-CNN, Faster R-CNN, YOLO і SSD300) на Рисунок 2.15. Для справедливого порівняння використовувались однакові налаштування моделі (VGG16 як базова мережа, розмір батча 1 і тестування на наборі даних Pascal VOC2007). Варто зауважити, що YOLO і SSD300 є одноступеневими детекторами, тоді як інші є двоступеневими детекторами, що базуються на підході з пропозицією по області.

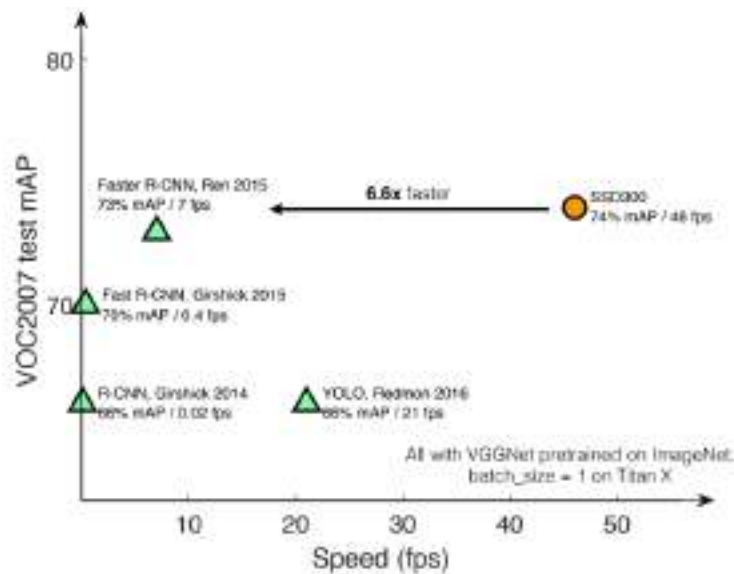


Рисунок 2.15 — Порівняння продуктивності SSD, Faster R-CNN і YOLO [26]

Можемо бачити, що SSD показує гарну точність, досягаючи досить непоганого значення fps, що дозволяє їй досить ефективно розпізнавати об'єкти в реальному часі з відповідною точністю. Оскільки SSD більше призначена для виявлення об'єктів у реальному часі, то можемо відкинути цей варіант в рамках завдання.

Крім того, з плином часу архітектура YOLO покращувалась, як зазначалось в попередньому пункті, з'являлись нові й нові версії, що показували кращу продуктивність:

Сімейство фреймворка Darknet

- YOLO
- YOLO 9000 (YOLOv2)
- YOLOv3

— YOLOv4

Сімейство на основі фреймворка PyTorch та інших

— Scaled-YOLOv4 (Масштабована)

— YOLOv5 (Ultralytics)

— YOLOv6

— YOLOv7

— YOLOv8 (Ultralytics)

— YOLOX

Загалом, моделі YOLO відрізняються від інших методів виявлення об'єктів, кількома ключовими особливостями:

— Single Shot Detection: YOLO використовує одну нейронну мережу для прогнозування як обмежувальних рамок, так і ймовірностей класів об'єктів на зображенні. Це робить YOLO швидшим за інші методи виявлення об'єктів, але й потенційно менш точним.

— Пряме передбачення: YOLO безпосередньо прогнозує ймовірності класів та обмежувальні рамки об'єктів на зображенні, не покладаючись на пропозиції по області. Це дозволяє YOLO виконувати виявлення об'єктів за один прямий прохід мережі, що набагато швидше, ніж інші методи, які виконують розпізнавання в декілька етапи.

— Прогнозування на основі сітки: YOLO розділяє вхідне зображення на сітку комірок і прогнозує наявність об'єктів у кожній комірці. Кожна комірка відповідає за прогнозування набору обмежувальних рамок і ймовірностей класів для об'єктів у межах своєї області. Це дозволяє YOLO обробляти декілька об'єктів різного масштабу на одному зображенні.

— Якірні рамки: Деякі з ранніх моделей YOLO використовують якірні рамки, які є попередньо визначеними рамками, щоб підвищити точність прогнозів. Якірні рамки використовуються для представлення попередніх знань мережі про форми і співвідношення сторін об'єктів, які вона, ймовірно, виявить. У деяких останніх моделях YOLO, таких як YOLOv8, якірні поля більше не потрібні, але використовується інший підхід.

Отже, і YOLO, і Faster R-CNN є ефективними алгоритмами виявлення об'єктів у глибокому навчанні. YOLO є швидким та ефективним, тоді як Faster R-CNN/RPN є більш точним, але повільнішим, до того ж YOLO має простішу структуру в порівнянні з Faster R-CNN.

Порівняльний графік YOLOv8, розроблену Ultralytics та попередніх моделей показаний на Рисунок 2.16.

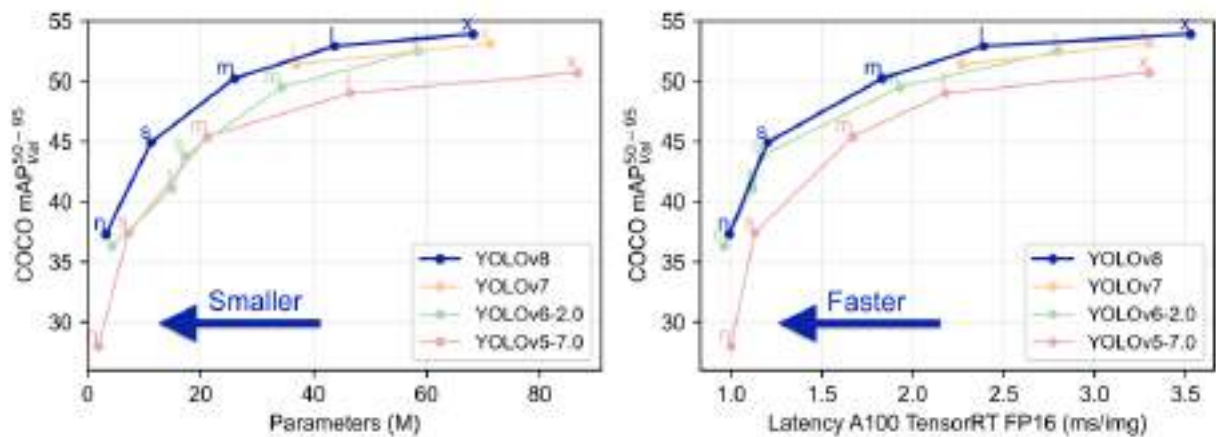


Рисунок 2.16 — Порівняння останніх 4-х версій YOLO [27]

Рисунок 2.16 демонструє, що модель YOLOv8 показує кращі результати ніж попередні версії за показниками mAP, розміру та швидкості при навчанні на наборі даних MS COCO.

Підсумовуючи, варто зазначити, що моделі YOLO пропонують компроміс між швидкістю і точністю порівняно з іншими методами виявлення об'єктів. Хоча вони швидкі і можуть виявляти об'єкти в режимі реального часу, їхній підхід прямого передбачення іноді може призвести до зниження точності порівняно з багатоетапними методами виявлення, таким як Faster R-CNN, проте вибір мережі YOLO дає можливість майбутнього розвитку проєкту в напрямку розпізнавання об'єктів на супутникових знімках в режимі реального часу.

## 2.8 Висновки до другого розділу

В даному розділі було досліджено різні архітектури deep learning, також було розглянуто основні характеристики, будову та особливості кожної з них. Окрім цього були розглянуті переваги та недоліки деяких моделей. На основі цього варто зазначити, що кожна архітектура має свої сильні та слабкі сторони в залежності від завдань, які вони вирішують. З цього випливає, що немає універсальної архітектури для всіх завдань комп'ютерного зору. Вибір моделі повинен базуватися на конкретних потребах та задачах, обчислювальних можливостях та інших складових. І як результат було вибрано кілька потенційно ефективних архітектур для виявлення об'єктів, проте вибір конкретної моделі був заснований на врахуванні специфіки задачі та ресурсів, а також на можливості подальшого покращення моделі та виконання нею більш комплексних задач в майбутньому.



## 3 ПІДХОДИ ДО ОБРОБКИ ЗНІМКІВ ТА ВИЯВЛЕННЯ ОБ'ЄКТІВ

### 3.1 Шум на супутникових знімках та методи його зменшення

В ході отримання та обробки супутникових знімків на кінцевому зображенні можуть мати місце навколишні або систематичні завади. Будь яка небажана інформація, що спотворює зображення вважається шумом. Основною причиною появи шумів на цифрових зображеннях є саме процес отримання цифрового зображення, при якому відбувається перетворення оптичного зображення в електричний сигнал, який згодом проходить через процес дискретизації. Залежно від способу отримання зображення, шум може спотворювати його різними способами.

#### 3.1.1 Типові шуми

Типові джерела появи шумів на супутникових знімках можна віднести до одного з наступних типів шумів:

1. Імпульсний шум випадкової варіації — він відомий також як гауссівський або звичайний шум. З'являється на цифровому зображенні у вигляді випадкових значень інтенсивності білого кольору.
2. Шум солі та перцю — такий тип шуму утворюється при перевищенні порогу зашумленості зображення. Цей шум являє собою чорні і білі пікселі, які виникають на зображенні випадково.
3. Spackle шум — він є поширеним явищем, що обмежує інтерпретацію оптичної когерентності на зображеннях дистанційного зондування. Являє собою зернисту шумову текстуру, що погіршує якість, внаслідок інтерференції між хвильовими фронтами в системах когерентного зображення.

Для обмеження або зменшення більшості шумів на супутникових знімках необхідно використовувати відповідні фільтри, оскільки вони можуть спотворювати зображення і ускладнювати його сприйняття. Такий підхід підвищує ймовірність більш точної інтерпретації вмісту зображення [28].

### 3.1.2 Підходи до зменшення шумів

Спочатку варто розглянути можливі варіанти використання фільтрів для зменшення впливу шумів на супутниковий знімок.

1. Фільтр гауса (Gaussian Filter) — зазвичай використовується в цифровому вигляді для опрацювання двовимірних сигналів (зображень) з метою зниження рівня шуму. Однак під час ресемплінгу він дає сильне розмиття зображення.
2. Mean filter — це простий, інтуїтивно зрозумілий і легкий у реалізації метод згладжування зображень. Його часто використовують для зменшення шуму на зображеннях.
3. Стандартний медіанний фільтр (Standard Median Filter) — у випадку, коли просторовий розподіл шуму на зображенні не є симетричним всередині вікна, медіанний фільтр, який є нелінійним фільтром, змінює середнє значення інтенсивності зображення. Медіанний фільтр зменшує дисперсію інтенсивностей на зображенні.
4. Адаптивний медіанний фільтр (Adaptive Median Filter) — ті недоліки, які були притаманні попередньому фільтру усуваються адаптивним медіанним фільтром. Це досягається за рахунок змінного розмір вікна адаптивного медіанного фільтра навколо кожного пікселя.
5. Адаптивний фільтр Вінера (Adaptive Wiener Filter) — фільтр змінює свою поведінку на основі статистичних властивостей зображення, включеного у вікно фільтра. Ефективність адаптивних фільтрів зазвичай краща, ніж у неадаптивних аналогів. Однак, покращена продуктивність диктує і додаткова складність фільтра [29].
6. Застосування методів на основі штучного інтелекту для аналізу і зменшення шумів на супутникових знімках. Наприклад, зменшення шуму на основі згорткових нейронних мереж, аналіз головних компонент і маскування даних для зменшення шуму або методи адаптивного вагового алгоритму [29].

### 3.1.3 Автокодери

Автокодери (англ. Autoencoders) з'явилися як одна з технологій і методів, що дозволяють комп'ютерним системам більш ефективно вирішувати проблеми стиснення даних. Також вони стали популярним рішенням для зменшення зашумлених даних. Прості автокодери представляють вихідні дані, які є такими самими або подібними до вхідних даних, тільки при цьому є стиснутими.

Автокодер — це тип штучної нейронної мережі, що використовується для навчання кодування даних у неконтрольований спосіб (без вчителя).

Метою автокодера є навчання представленню (кодуванню) даних у меншій розмірності для даних у більшій розмірності, як правило, для зменшення розмірності, шляхом тренування мережі виокремленню найбільш важливих частин вхідного зображення.

Автокодери складаються з 3 частин:

1. Кодер (англ. Encoder): Модуль, який стискає вхідні дані набору у закодоване представлення (представлення прихованого простору), яке зазвичай на кілька порядків менше за вхідні дані, генеруючи при цьому нове представлення даних із меншою розмірністю.
2. Вузьке місце / латентне представлення (англ. Bottleneck / Latent Representation): Модуль, який містить стиснене представлення даних і тому є найважливішою частиною мережі. Код вузького місця ретельно розроблено, щоб визначити найбільш релевантні частини спостережуваних даних або, інакше кажучи, особливості даних, які є найважливішими для реконструкції даних [30]. По суті вузьке місце існує для того, щоб обмежити потік інформації від кодера до декодера, таким чином, пропускаючи лише найбільш важливу інформацію. Оскільки вузьке місце сконструйовано таким чином, що в ньому фіксується максимум інформації, яку містить зображення, можна сказати, що вузьке місце допомагає нам сформуванню знання-представлення про вхідні дані.

3. Декодер: Модуль, який допомагає мережі "розпаковувати" подання знань і відновлює дані з їхньої закодованої форми, у представлення з тими ж розмірами, що й оригінальні, незмінні дані. Вихідні дані потім порівнюються з істиною в останній інстанції.

Відповідна архітектуру показано на Рисунок 3.1.

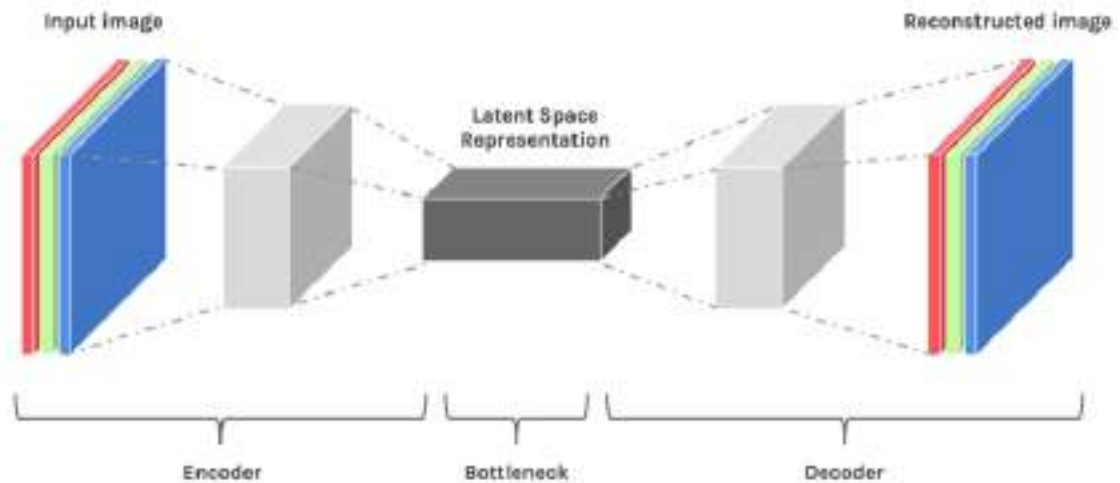


Рисунок 3.1 — Архітектура автокодера

Існують різні типи автокодерів, виділимо основні:

- Стискаючий (англ. Undercomplete) автокодер — це один з найпростіших типів автокодерів. Він отримує зображення і намагається передбачити те саме зображення на виході, таким чином реконструюючи зображення зі стисненого вузького місця.
- Розріджений (англ. Sparse) автокодер — розріджені автокодери схожі на стискаючі автокодери в тому, що вони використовують те саме зображення як вхідні дані та фундаментальну істину. Однак, засоби, за допомогою яких регулюється кодування інформації, суттєво відрізняються. Розріджений автокодер це кодер, у якого у функцію втрат доданий штраф за величини значень в коді (вузькому місці), тобто автокодер прагне мінімувати функцію помилки. Цей штраф, який називається функцією розрідженості, не дозволяє нейронній мережі активувати більше нейронів і слугує регулятором.

- Варіаційний (англ. Variational) автокодер — стандартні та варіаційні автокодери навчаються представляти вхідні дані у стиснутій формі, яка називається латентним простором або вузьким місцем. Він робить сильні припущення про розподіл латентних змінних і використовує стохастичний градієнтний варіаційний байєсівський оцінювач у процесі навчання.
- Стягувальний (англ. Contractive) автокодер — виконують завдання навчання представлення зображення під час проходження його через вузьке місце і реконструюють його в декодері. Він також має регуляризаційний елемент, щоб запобігти вивченню мережею функції ідентичності та відображенню вхідних даних на виході.
- Знешумлюючий (англ. Denoising) автокодер — як видно з назви, — це автокодери, які видаляють шум із зображення. На відміну від автокодерів, розглянутих вище, це перший тип автокодерів, який не використовує вхідне зображення як фундаментальну істину. Знешумлюючий автокодер, працює на частково пошкоджених вхідних даних і навчається відновлювати оригінальне неспотворене зображення. Як зазначалось вище, цей метод є ефективним способом стримати мережу від простого копіювання вхідних даних і, таким чином, вивчити основну структуру та важливі особливості даних.

### **3.2 Архітектура обраного методу зменшення шуму**

Перш за все, варто зазначити, що знешумлюючі автокодери являються найкращими детекторами країв та більших контурів з природних ділянок зображення та цифрових зображень відповідно [31] [32]. Знешумлюючі автокодери працюють краще в порівнянні з традиційними фільтрами для зменшення шумів, оскільки автокодер може бути модифікований на основі вхідних даних, на відміну від традиційних фільтрів.

Ефективність у видаленні шуму зі знімків звісно залежить від типу шуму, також від його розподілу та інших характеристик. Автокодери та традиційні методи фільтрації, розглянуті в попередніх розділах, мають свої переваги та

недоліки. Традиційні методи фільтрації, такі як медіанний фільтр, фільтр Гаусса, адаптивний фільтр та інші, можуть бути досить ефективними для видалення певних типів шуму, наприклад, білого шуму або шуму з доданою гауссівською складовою. Вони можуть добре працювати в ситуаціях, коли шум має сталий або відомий характер.

Автокодері ж показують гарні результати при роботі з шумами менш передбачуваних або складних типів та можуть ефективно працювати з різними видами шуму, включаючи шуми зі змішаними складовими. Тому у якості інструмента зменшення шуму було використано автокодер [32].

### ***3.2.1 Архітектура автокодера***

У даному підході було застосовано CNN у якості знешумлюючого кодувальника через ефективність цієї концепції у зменшенні впливу шумів та збереженні просторових відношень на зображенні. Крім того, вибір CNN базується на меті зменшення розмірності та обчислювальної складності, коли в якості вхідних даних використовуються зображення довільного розміру.

Розмір вхідного шару дорівнює розміру зображення (1024, 1024, 3). Як було зазначено вище, у зв'язку з тим, що дані являють собою зображення, тоді буде створений автокодер, що складається зі згорткових шарів. Необхідно створити дві функції: кодер і декодер.

Кодер складатиметься з двох згорткових шарів: перший з 32 фільтрами та розміром ядра цих фільтрів  $3 \times 3$  ( $32 \times 3 \times 3$ ) і відповідно другий з 32 фільтрами та розміром ядра цих фільтрів  $3 \times 3$  ( $32 \times 3 \times 3$ ) і двох шарів максупулінгу  $2 \times 2$ , що відображено на Рисунок 3.2.

```

dimension = 1024
input_img = keras.Input(shape = (dimension, dimension, 3))

x = layers.Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')(input_img)
x = layers.MaxPooling2D(pool_size = (2, 2), padding = 'same')(x)
x = layers.Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')(x)
encoded = layers.MaxPooling2D(pool_size = (2, 2), padding = 'same')(x)

```

Рисунок 3.2 — Кодувальна частина

Декодер, по суті, є повною протилежністю кодера, оскільки відбувається відновлення 2D представлення зображень. Він складатиметься з двох згорткових шарів по 32 фільтри та розміром ядра цих фільтрів 3x3 (32x3x3) відповідно і двох шарів із підвищувальною дискретизацією 2x2, це можна бачити на Рисунок 3.3.

```

x = layers.Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')(encoded)
x = layers.UpSampling2D(size = (2, 2))(x)
x = layers.Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')(x)
x = layers.UpSampling2D(size = (2, 2))(x)
decoded = layers.Conv2D(filters = 3, kernel_size = (3, 3), activation = 'sigmoid', padding = 'same')(x)
autoencoder = keras.Model(input_img, decoded)

```

Рисунок 3.3 — Декодувальна частина

Далі з'єднується вхід і вихід, для формування і компіляції автокодера. Варто зазначити, що у всіх згорткових шарах, крім останнього, використовується функція активації ReLU. Також в останньому шарі декодера використовується сигмоїдальна функція активації.

### 3.3 Задача виявлення об'єктів

Для отримання повного розуміння зображення, варто не тільки зосередитися на класифікації різних зображень, але й спробувати точно оцінити концепції та розташування об'єктів, що містяться на кожному такому зображенні. Це завдання називається виявленням об'єктів.

По суті виявлення об'єктів — це завдання комп'ютерного зору, яке передбачає ідентифікацію та визначення місцезнаходження об'єктів на зображеннях або відео. Це важлива частина багатьох застосувань, таких як відеоспостереження, безпілотні автомобілі або робототехніка.

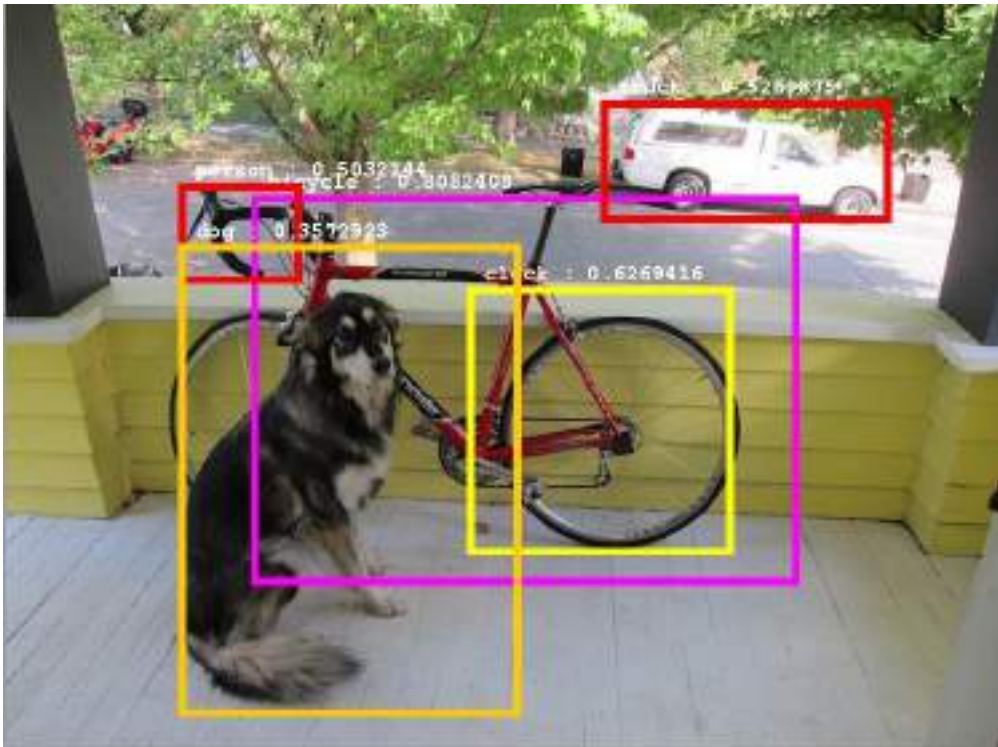


Рисунок 3.4 — Приклад розпізнавання об'єктів на зображенні та їх анотування

Алгоритми виявлення об'єктів поділяються на дві категорії залежно від того, скільки разів одне й те саме вхідне зображення пропускається через мережу, що відображено на Рисунок 3.5.

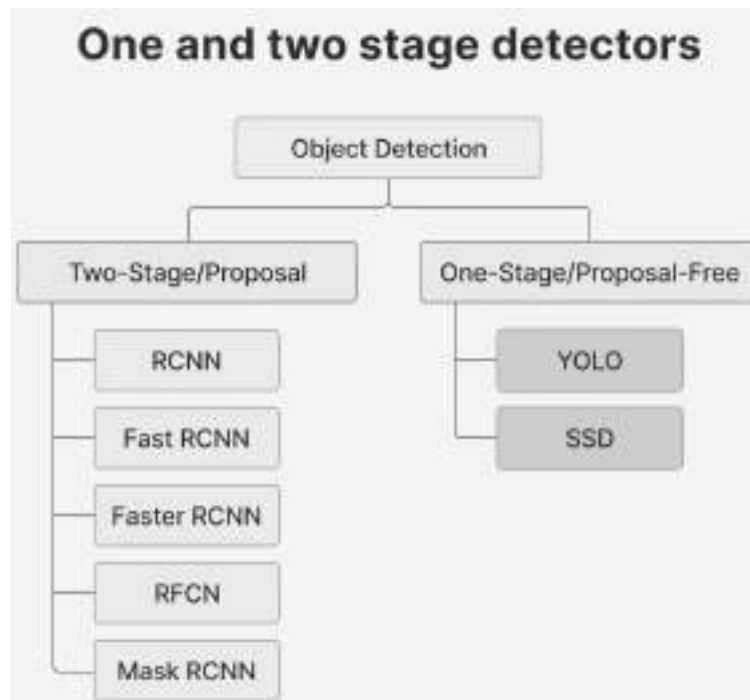


Рисунок 3.5 — Типи детекторів



Однією з перших успішних спроб вирішити проблему виявлення об'єктів за допомогою глибокого навчання була вже згадана модель R-CNN (Regions with CNN features), розроблена Россом Гіршиком та його командою в Microsoft Research у 2014 році. Ця модель використовує комбінацію алгоритмів пропозиції по області і згорткових нейронних мереж (CNN) для виявлення і локалізації об'єктів на зображеннях.

Single-shot object detection реалізовує один прохід вхідного зображення для прогнозування наявності або розташування об'єктів на цьому зображенні. Цей метод обробляє все зображення за один прохід, що позначається на обчислювальній ефективності. Однак виявлення об'єктів за один прохід, як правило, менш точне, ніж інші методи, і менш ефективно при виявленні невеликих об'єктів.

Two-shot object detection реалізовує два проходження вхідного зображення для прогнозування наявності або розташування об'єктів. Перший прохід використовується для генерації набору пропозицій або потенційних місць розташування об'єктів, а другий — для уточнення цих пропозицій і остаточного прогнозу. Цей підхід є більш точним, ніж Single-shot object detection, але він також є більш затратним в обчислювальному плані.

### **3.4 Метрики оцінки ефективності моделей виявлення об'єктів**

Для визначення і порівняння ефективності прогнозування різних моделей виявлення об'єктів, використовуються стандартні кількісні метрики.

Дві найпоширеніші метрики оцінки — це метрика Intersection over Union (IoU) та Average Precision (AP).

#### ***3.4.1 Intersection over Union***

Intersection over Union — метрика степені перетину між двома обмежувальними рамками. Вона є популярною метрикою для вимірювання точності локалізації та обчислення помилок локалізації в моделях виявлення об'єктів.

Для обчислення обчислити IoU між прогнозованою та істинною областями, спочатку береться площа перетину між двома відповідними областями для одного і того ж об'єкта.

Після цього обчислюється загальна площа, що покривається цими двома граничними областями — також відому як «Union», і площу перетину між ними, яка називається «Intersection».

«Intersection», поділений на «Union», показує відношення перекриття до загальної площі, що дає гарну оцінку того, наскільки близько прогнозований контур знаходиться до вихідного контуру, наглядно це показано на Рисунок 3.6.

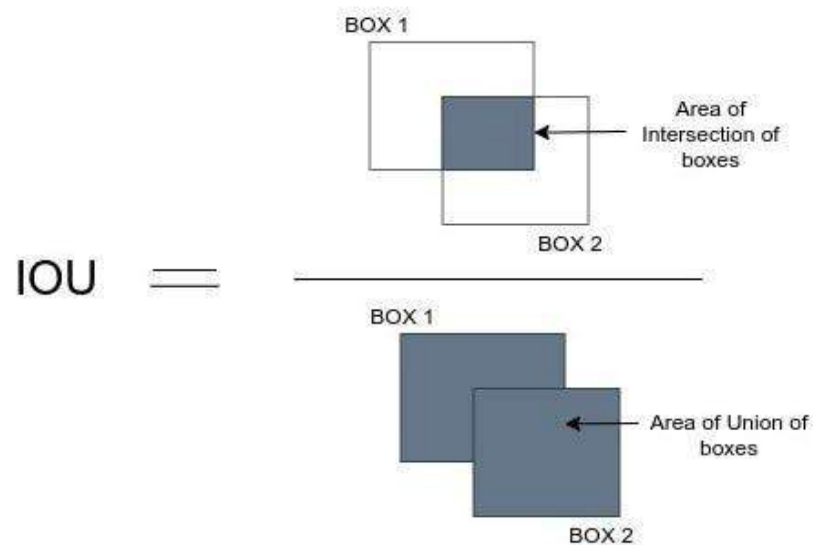


Рисунок 3.6 — Схематичне зображення формули для розрахунку IoU [33]

### 3.4.2 Average Precision

Average Precision (AP), середня точність — обчислюється як площа під кривою залежності Precision від Recall для набору передбачень.

Recall розраховується як відношення загальної кількості передбачень, зроблених моделлю для класу, до загальної кількості існуючих міток для цього класу. Precision — це відношення кількості істинно-позитивних прогнозів до загальної кількості прогнозів, зроблених моделлю.

Recall і Precision виражають компроміс, який графічно представлений у вигляді кривої при зміні порогу класифікації. Площа під цією кривою залежності Precision від Recall дає нам середню точність на клас для моделі. Середнє значення цього показника для всіх класів називається середньою точністю (Average Precision, mAP).

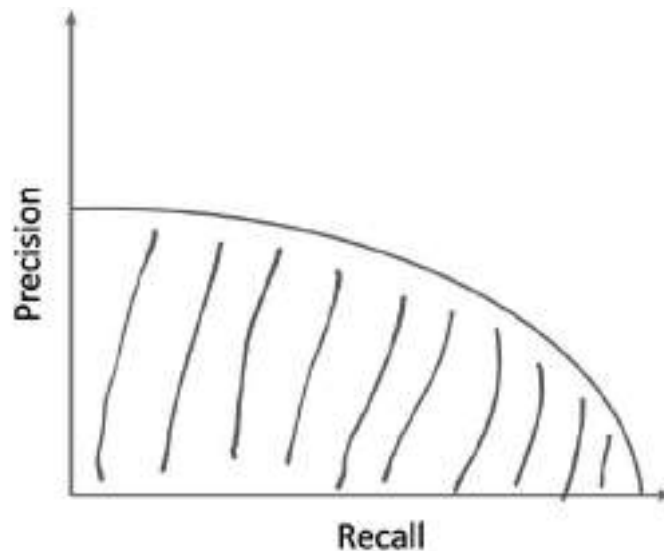


Рисунок 3.7 — Крива та площа під нею, mAP

mAP — це комбінація значень Precision та Recall, які розраховуються на основі декількох порогів довіри, які також відомі як IoU, що був описаний вище. Зміна цього порогу IoU призведе до різних результатів True Positives (TP) та False Positives (FP) [34].

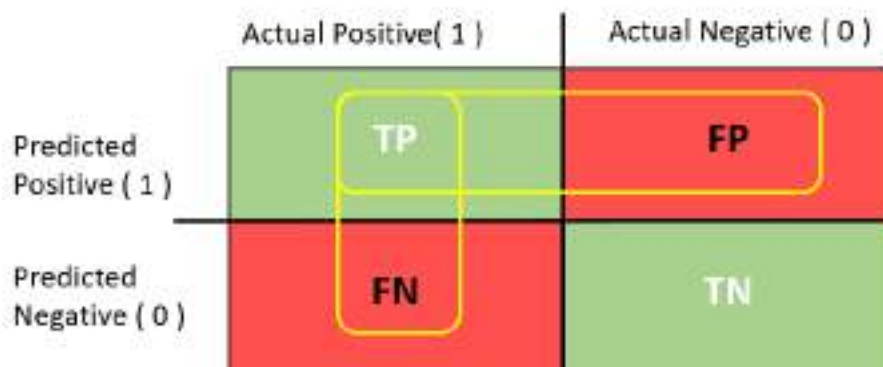


Рисунок 3.8 — Матриця помилок [34]

Precision це частка виявлення TP серед усіх виявлень, зроблених при певному пороговому значенні IoU як видно у наступній формулі:

$$Precision = \frac{TP}{TP + FP} \quad (3.1)$$

де  $TP$  — фактичний позитивний клас є прогнозовано позитивним;

$FP$  — фактичний клас є негативним, але передбачуваний клас є позитивним.

$Recall$  — це частка виявлення  $TP$  серед усіх можливих виявлень, зроблених при певному пороговому значенні, розраховується за наступною формулою:

$$Recall = \frac{TP}{TP + FN} \quad (3.2)$$

де  $TP$  — фактичний позитивний клас є прогнозовано позитивним;

$FN$  — Фактичний клас є позитивним, але передбачуваний клас є негативним.

Загальна формула для розрахунку  $mAP$  приведена нижче, у формулі:

$$mAP = \frac{1}{N} \sum_{n=1}^N AP_i \quad (3.3)$$

де  $N$  — кількість класів об'єктів;

$AP_i$  — середня точність для  $i$ -го класу.

Середня точність для одного класу розраховується шляхом попереднього сортування всіх виявлених об'єктів за показниками достовірності у порядку спадання, потім обчислюється  $Precision$  і  $Recall$  при кожному пороговому значенні, і, нарешті, проводиться обчислення площі під кривою ( $Area Under the Curve$ ,  $AUC$ ) шляхом інтегрування кривої  $Precision-recall$  при всіх можливих порогових значеннях.

### 3.5 You Only Look Once v8

У попередніх пунктах було обґрунтовано вибір на користь мережі YOLO. Також було показано порівняння ефективності відповідних версій такої мережі. Відповідно, в якості НМ для розпізнавання об'єктів буде застосовано найно-

вішу версію YOLO, v8. You Only Look Once v8 (YOLOv8) розроблений компанією Ultralytics у січні 2023 року і на даний момент є одним з найновіших доповнень до архітектур YOLO. У наступних розділах буде більш детально буде описано мережу, а також два алгоритми оптимізації для обчислення вагових коефіцієнтів під час навчання.

### 3.5.1 Архітектура YOLOv8

Оскільки ніякого офіційного документу від Ultralytics з архітектурою знайти не вдалось, то буде розглянуто схему, створену Range King на веб-сервісі GitHub, який, до речі, було підтверджено розробниками YOLOv8 [35].

На Рисунок 3.9 зображено огляд моделі, де ліворуч показана основна мережа (Backbone), яка слугує для отримання ознак із зображення, яке надходить на вхід, а праворуч модуль виявлення (Head). При цьому повну архітектуру приведено в ДОДАТКУ А.

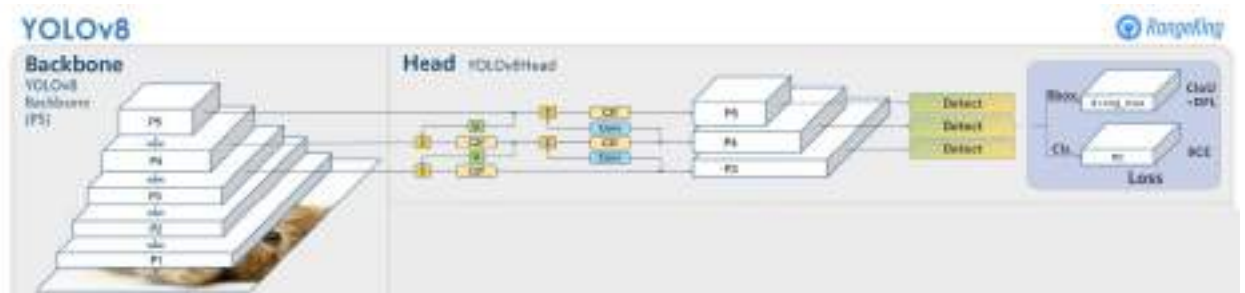


Рисунок 3.9 — Архітектура YOLOv8 [35]

Як показано в огляді моделі на Рисунок 3.9, основа (база) нейронної мережі складається з п'яти шарів (з P1 по P5), і проводить виявлення в трьох різних масштабах, використовуючи функції втрат, які будуть описані нижче. З Рисунок 3.9, також видно, що модуль виявлення (Head) моделі використовує комбінацію таких частин як Conv, Detect та C2f у поєднанні з конкатенацією та апсемплінгом, які також будуть розглянуті детальніше нижче.

#### Компонент Conv

Компонент Conv складається з трьох частин: Conv2d — це згортковий 2д фільтр із заданим розміром ядра (kernel size), кроком сканування (stride), доповнення (padding) і каналами; BatchNorm2d — блок пакетної нормалізації

та функції активації SiLU (Sigmoid-weighted Linear Unit). Conv приймає 4 параметри, які мають вплив на Conv2d, як показано на Рисунок 3.10.

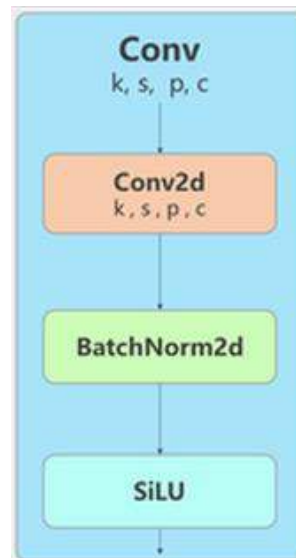


Рисунок 3.10 — Компонент Conv YOLOv8

### Компонент Detect

Компонент Detect, складається з двох гілок по чотири блоки в кожній, який зображено на Рисунок 3.11.

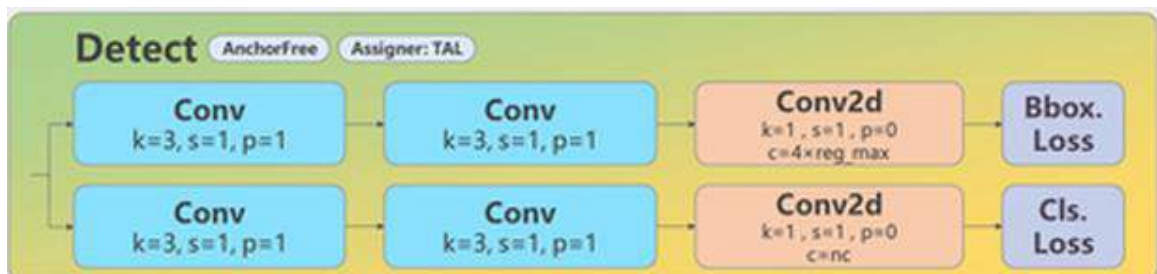


Рисунок 3.11 — Компонент Detect YOLOv8

Верхня гілка обчислює Bbox loss (втрати обмежувальних рамок) після проходження через два блоки Conv з розміром ядра рівному 3, кроком рівному 1 і доповненням рівному 1 кожен, та через блок Conv2d з відповідною кількістю каналів (4 помножене на максимальне абсолютне значення цілей регресії). Це гіперпараметр, який використовується для нормалізації цілей регресії під час навчання. Це гарантує, що прогнозовані зміщення знаходяться в межах розумного діапазону, що може покращити стабільність та збіжність процесу навчання [36]. Нижня гілка проходить через ті ж самі блоки, проте другий блок згортки має таку ж кількість каналів, як і кількість класів, які моделі повинні

передбачити. Модуль Detect є без'якірним, тобто немає необхідності використовувати попередньо визначені якорі під час прогнозування обмежувальних рамок, і він розглядається як регресійна задача [36].

### Компонент C2f

Компонент C2f, містить два модулі Conv, один модуль Split, один модуль Concat і може містити один або більше модулів Bottleneck, як показано на Рисунок 3.12.

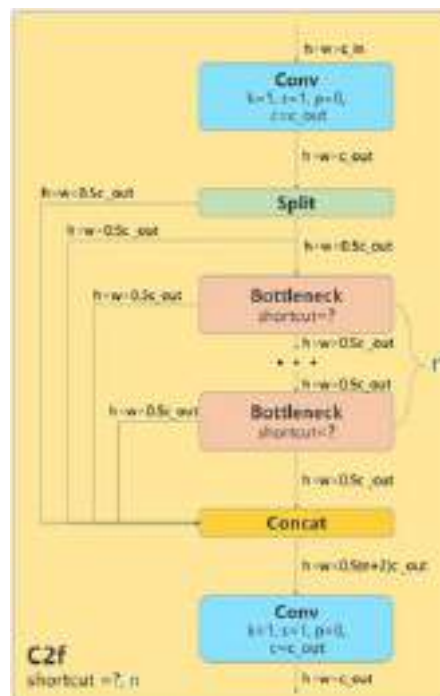


Рисунок 3.12 — Модуль C2f

Компонент Bottleneck може бути як із з'єднанням швидкого доступу (shortcut), так і без нього, але і в тому і в іншому варіанті, він містить два модулі Conv, які мають розмір ядра 3, крок 1 і доповнення 1. Різниця між ними полягає в наявності операції додавання, що справедлива для компоненту зі з'єднанням швидкого доступу зі встановленим True, як показано на Рисунок 3.13.

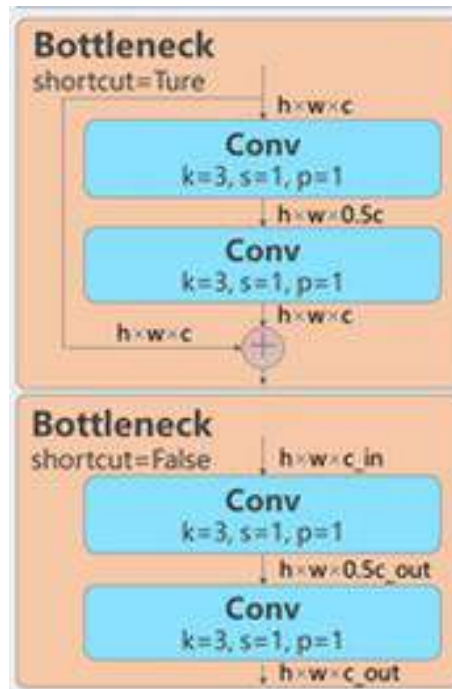


Рисунок 3.13 — Варіанти конфігурації компонента Bottleneck

### Основна мережа YOLOv8

Основна мережа або ж ядро (Backbone) складається з модуля SPPF (Spatial Pyramid Pooling Fast), п'яти модулів Conv і чотирьох модулів C2f з різними конфігураціями, що показано на Рисунок 3.14.





Рисунок 3.14 — Backbone архітектури YOLOv8

SPP складається з двох модулів Conv, трьох модулів MaxPool2d і модуля Concat, як показано на Рисунок 3.15. SPPF математично ідентичний SPP, але з меншою кількістю FLOPs (Floating Point of Operations).

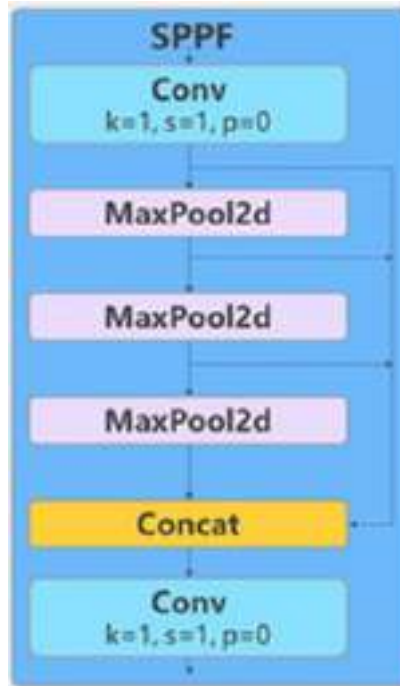


Рисунок 3.15 — модуль SPPF

Далі варто зазначити, що C2f і модуль SPPF мають виходи, з'єднані з частиною, що відповідає за детектування (Head), що показана на Рисунок 3.16.

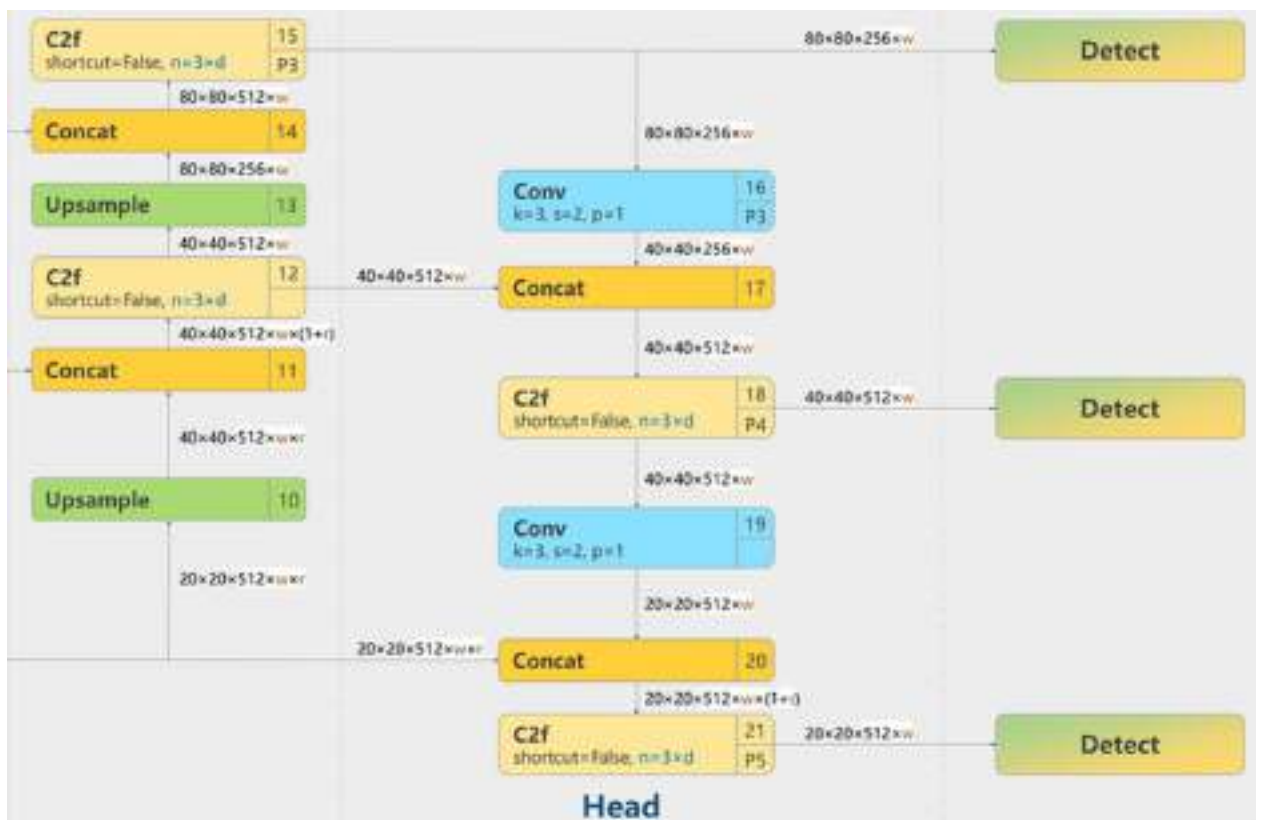


Рисунок 3.16 — Модуль виявлення (Head)

Розміри трьох шарів визначаються трьома величинами: шириною та висотою відповідного шару, а також кількістю каналів.

З Рисунок 3.14 та Рисунок 3.16 можна бачити, що кількість каналів залежить від вихідних даних попередніх шарів, а також від двох констант  $w$  і  $r$ .  $w$  і  $r$  залежать від розміру моделі і відповідно відображені на Рисунок 3.17.

model	d (depth_multiple)	w (width_multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Рисунок 3.17 — Доступні моделі YOLOv8 та їх параметри

У цій роботі використовувався розмір моделі n, і тому значення для  $w$  і  $r$  дорівнюють 0,25 і 2 відповідно.

### 3.6 Висновки до третього розділу

В даному розділі були розглянуті типові шуми та методи і техніки для зменшення їх впливу на зображеннях, зокрема були розглянуті стандартні методи, такі як фільтри та нейронні мережі, а саме автокодера. У підсумку був обраний метод зменшення шуму, який ґрунтується на використанні автокодера. Крім цього також була обрана та описана його оптимальна структура.

Також в третьому розділі було описано одну із важливих задач у обробці супутникових знімків, а саме задачу виявлення об'єктів, таких як будівлі, транспортні засоби, тощо. Були також розглянуті метрики оцінки ефективності моделей, та описана архітектура мережі для розпізнавання об'єктів.

В цілому, розділ включає аналіз проблеми шуму на супутникових знімках, методи його зменшення, важливість виявлення об'єктів на зображеннях, а також огляд метрик оцінки та обраної архітектури, яка використовується для вирішення цього завдання.

## 4 РЕАЛІЗАЦІЯ НЕЙРОННОЇ МЕРЕЖІ

### 4.1 Підготовка наборів даних

Для того, щоб навчити нейронну мережу виявляти об'єкти на супутникових знімках вкрай важливо навчати її на якісному наборі даних. Дуже вагомим нюансом є те, щоб набір даних був анотований горизонтальними рамками з мітками для кожної рамки, інакше час, витрачений на анотування набору даних вручну зайняв би дуже багато часу.

Крім того, набір даних повинен містити всі обрані класи (транспортні засоби, кораблі, літаки). З огляду на це, було досліджено кілька наборів даних і обрано декілька для подальшої роботи: DOTA [37], xView [38] [39] та DIOR [40]. За основу був взятий набір даних DOTA та до нього були додані деякі знімки з двох інших датасетів, саме ті, що підходили під обрані класи. До прикладу багато класів у наборі даних DIOR не є такими, що співпадали б з обраними класами, і тому для навчання моделей були використані лише зображення, що містять такі класи: літаки, кораблі та транспортні засоби.

Формат анотацій в датасеті DOTA показаний на Рисунок 4.1.

```
x1, y1, x2, y2, x3, y3, x4, y4, category, difficult
x1, y1, x2, y2, x3, y3, x4, y4, category, difficult
...
```

Рисунок 4.1 — Формат анотацій в наборі даних DOTA [37]

Де  $x_i$  та  $y_i$  відповідають вершині обмежувальної рамки, якою анотовано кожен об'єкт того чи іншого класу. Окрім цього, параметрами є клас (category), до якого відноситься об'єкт та складність (difficult), яка приймає значення 0 та 1, та вказує на те чи складно виявити об'єкт (1 — складно, 0 — не складно) [37]. Проте, для YOLO такий формат не підходить, тому анотації були змінені наступним чином як показано на Рисунок 4.2.

```
10 0.292129 0.306798 0.293161 0.305525 0.297032 0.306616 0.296258 0.307706
9 0.384774 0.729371 0.384258 0.731189 0.374452 0.729553 0.375226 0.727372
```

Рисунок 4.2 — Змінені анотації

Для навчання були обрані наступні класи, показані на Рисунок 4.3.

```
# number of classes
nc: 16

# class names          # class number
names: ['small-vehicle', # 0
        'large-vehicle', # 1
        'plane',         # 2
        'storage-tank',  # 3
        'ship',          # 4
        'harbor',        # 5
        'ground-track-field', # 6
        'soccer-ball-field', # 7
        'tennis-court',  # 8
        'swimming-pool', # 9
        'baseball-diamond', # 10
        'roundabout',   # 11
        'basketball-court', # 12
        'bridge',       # 13
        'helicopter',   # 14
        'container-crane'] # 15
```

Рисунок 4.3 — Класи, що використовувались при навчанні НМ

Можемо бачити 16 класів: малий транспортний засіб, великий транспортний засіб, резервуар, корабель, гавань (пристань), спортивний майданчик з біговими доріжками, футбольне поле, тенісний корт, басейн, бейсбольний майданчик, кільцева транспортна розв'язка, баскетбольний майданчик, міст, гелікоптер, контейнерний кран. Приклад зображення деяких класів показано на Рисунок 4.4.

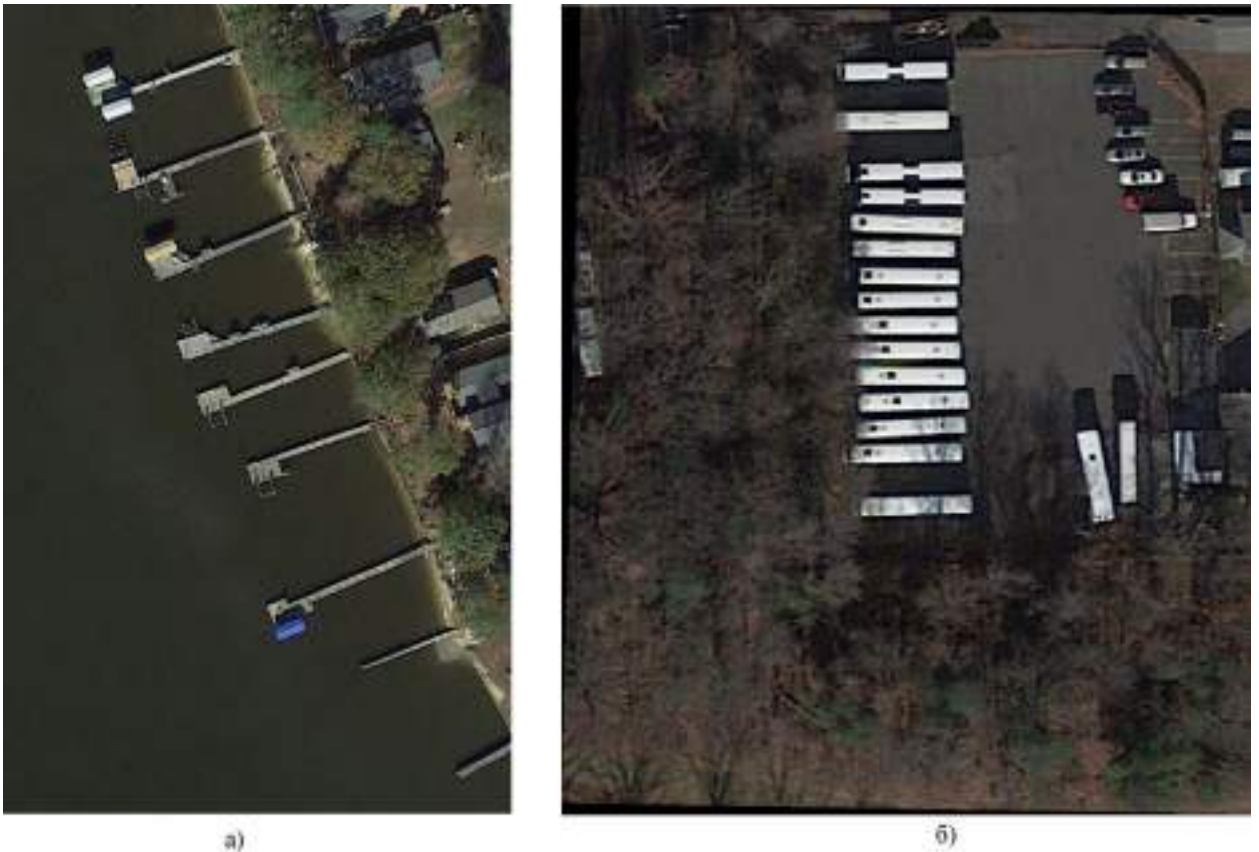


Рисунок 4.4 — Приклад знімків з набору DOTA: а) — зображення пристані (клас harbor); б) — зображення автомобілів (класи small-vehicle та large-vehicle)

#### ***4.1.1 Доповнення набору даних шляхом видалення фрагментів із зображення***

Також гарним варіантом є доповнення набору даних тими ж знімками, тобто застосовуються різні методи зміни оригінальних зображень різними способами. Всі методи доповнення даних штучно створюють нові зображення з оригінальних навчальних даних, ефективно збільшуючи розмір навчального набору і підвищуючи надійність моделі.

Однією з таких технік може бути видалення фрагментів із зображення, приклад такої техніки показано на Рисунок 4.5.



Рисунок 4.5 — Видалення деяких областей із зображення: а) — оригінальний знімок; б) — знімок з видаленням фрагментів

Можливий також варіант зміни зображень шляхом видалення більшого числа зон із зображення, як показано на Рисунок 4.6.



Рисунок 4.6 — Видалення великої кількості зон зі знімку

#### ***4.1.2 Доповнення набору даних шляхом обертання оригінального знімку***

При доповненні набору даних можна використати наступний підхід. Зображення змінюється шляхом обертання оригінального знімку на заданий кут, зазвичай в межах діапазону від -15 до 15 градусів. Обертаючи зображення, модель піддається впливу різної орієнтації об'єктів на знімках. Оскільки об'єкти на супутникових знімках можуть мати будь-яку орієнтацію, то знімки можна обертати в діапазоні від -180 до 180 градусів. Приклад такого підходу показано на Рисунок 4.7.



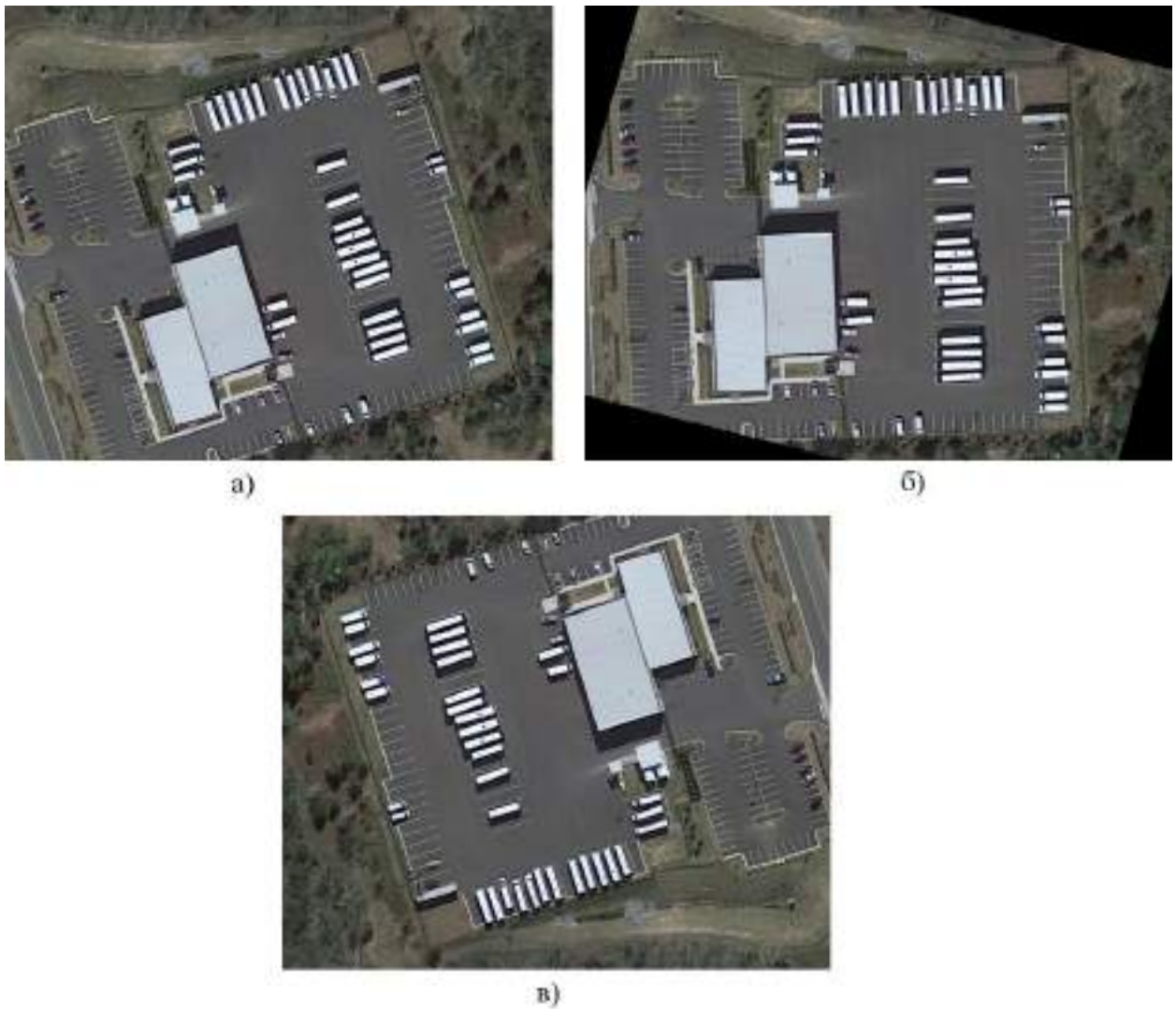


Рисунок 4.7 — Реалізація повороту знімку: а) — оригінальне зображення; б) — зображення повернуте на 15 градусів; в) — зображення повернуте на 180 градусів

#### ***4.1.3 Доповнення набору даних шляхом перекосу зображення***

Ще одним методом доповнення набору даних можливе додавання випадкового зсуву або перекосу до навчального зображення, тобто трансформація на деяку величину в горизонтальному або вертикальному напрямку. Завдяки зсуву зображень модель має різні кути спостереження на об'єкти на зображеннях. Такий підхід показано на Рисунок 4.8.

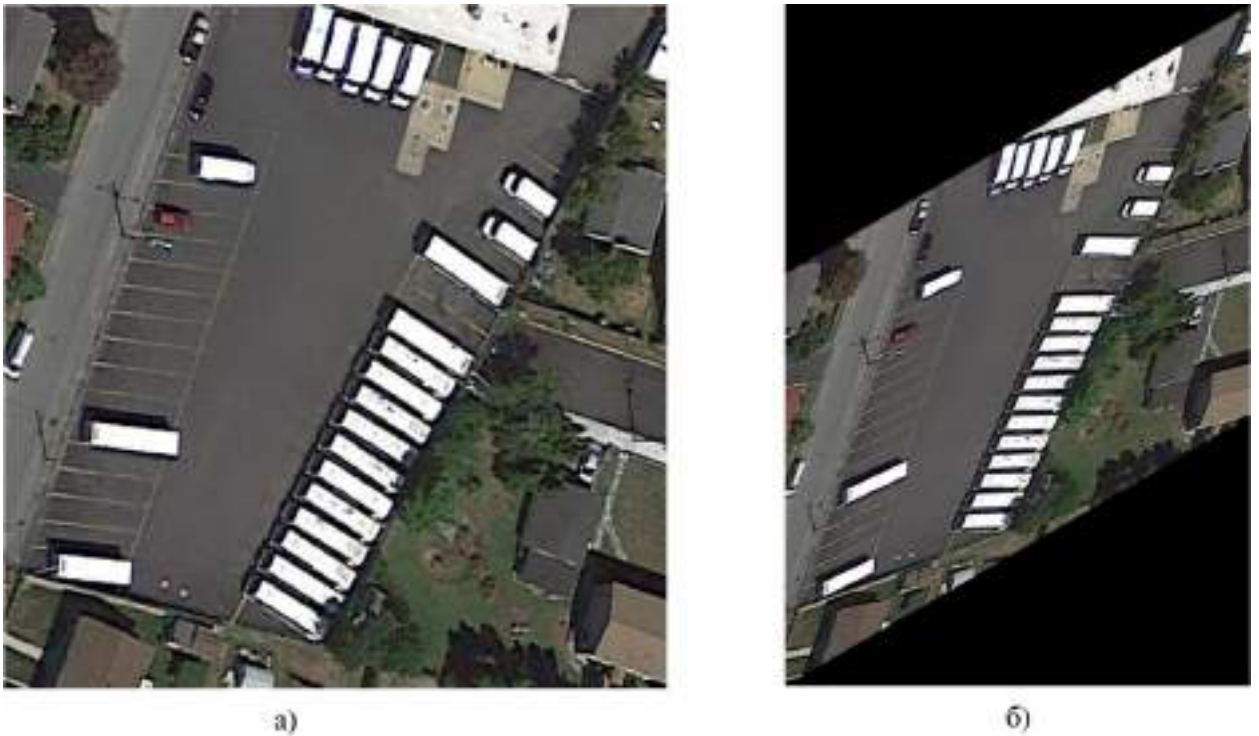


Рисунок 4.8 —Реалізація перекосу знімку: а) — оригінальне зображення; б) — застосування перекосу до зображення

## 4.2 Навчання автокодера

Навчання автокодера проводилось на зашумлених знімках. Тобто задача навчання мережі полягала в наступному: додати шум до знімків, а після цього на вхід автокодера подавати зашумлені знімки та на виході порівнювати з даними, в нашому випадку зі знімками без шуму.

Покажемо приклад оригінального зображення та зашумленого, які будуть використані для навчання, на Рисунок 4.9.

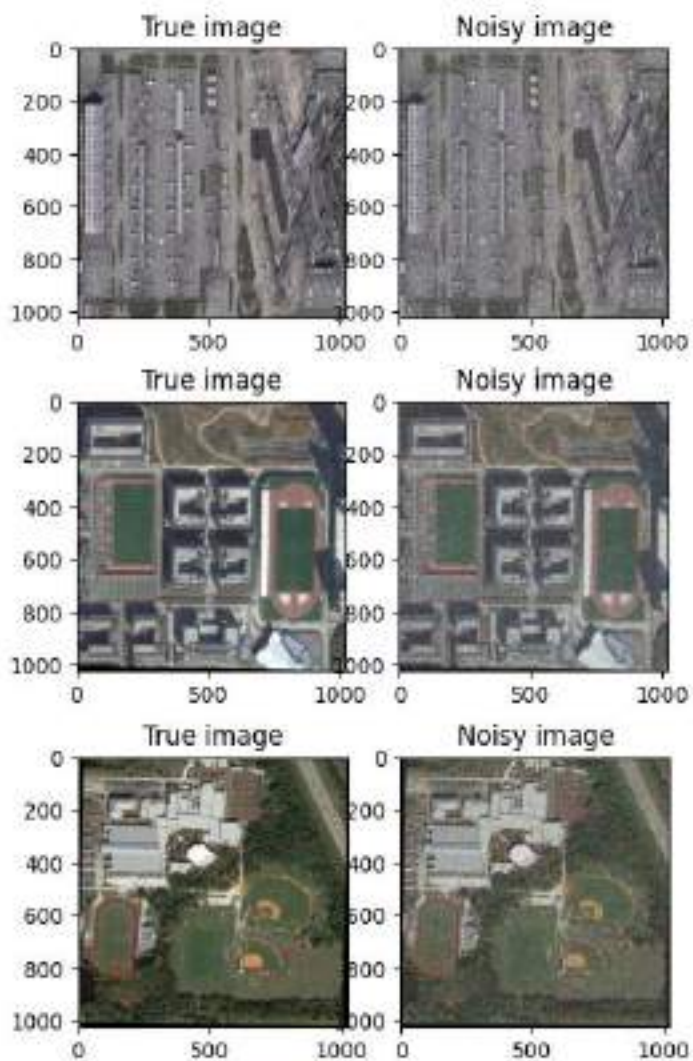


Рисунок 4.9 — Оригінальні (зліва) та зашумлені (справа) знімки для навчання автокодера

Увесь набір даних для навчання був розбитий на навчальну та валідаційну вибірки.

Зведемо параметри, які були використані для навчання в Таблиця 4.1

Таблиця 4.1

Параметр	Значення
epochs	50
batch size	2
shuffle	true
validation_split	0.4
optimizer	adam
loss	binary_crossentropy

Таким чином, кодер отримує на вхід зашумлені зображення і вчиться їх стискати, а потім декодер вчиться розпаковувати їх у очищені від шуму зображення.

Покажемо графік критерія якості, тобто метрики втрат для тренувальної та валідаційної вибірок на Рисунок 4.10.

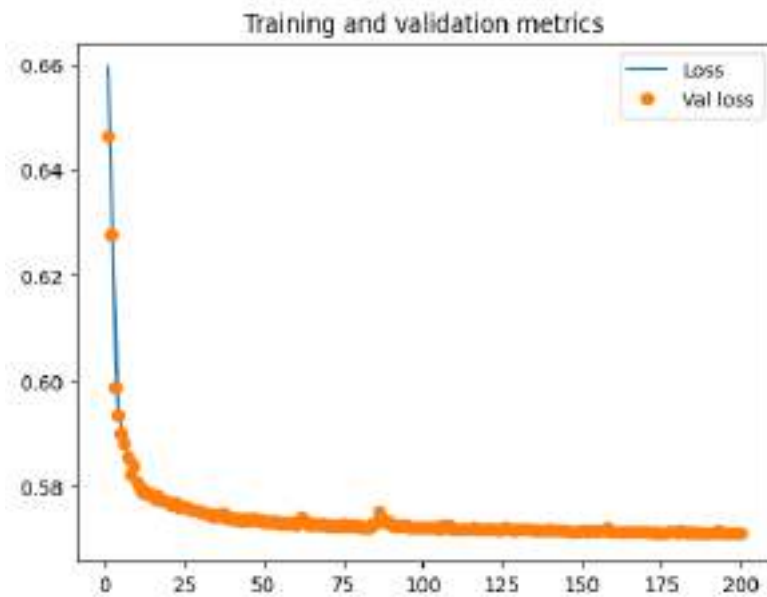


Рисунок 4.10 — Метрики тренувальної та валідаційної вибірок

Тепер можна візуалізувати отримані дані. Відобразимо вихідне зображення, тобто зашумлене (зліва) і відновлене (справа) зображення за допомогою автокодера на Рисунок 4.11.



Рисунок 4.11 — Зашумлене та відновлене зображення автокодером

### 4.3 Навчання мережі YOLOv8

Протягом етапу навчання, мережа приймає велику кількість вхідних даних, а відповідно отримані помилки поширюються у зворотному напрямку через всю мережу, коригуючи значення матриць на кожному шарі та рівні. Як і описувалось в попередніх пунктах цей процес називається зворотне поширення помилки (англ. *backpropagation*). Власне процес навчання був описаний в перших розділах роботи, тому тут зосередимось на деяких нюансах.

По перше, проводиться ініціалізація, тобто вагові коефіцієнти НМ ініціалізуються з попередньо навченої моделі. Після чого відбувається прямий прохід через мережу, в ході якого, вхідні дані проходять через шари НМ. Кожен шар виконує відведений йому набір операцій, це, наприклад може бути операція згортки або операція пулінгу, які використовуються для отримання певних ознак з вхідних даних. Після завершення процесу прямого проходження те, що було отримано на виході мережі порівнюється з бажаним виходом за допомогою функції втрат. Втрати розпізнавання моделі YOLOv8 обчислюються за допомогою функції бінарної крос-ентропії (BCE). Далі проводиться зворотній прохід по мережі, в ході якого обчислюються градієнти втрат по відношенню до ваг мережі. Отримані градієнти застосовуються для оновлення ваг мережі в напрямку, протилежному градієнту, з метою мінімізації втрат. Вагові коефіцієнти мережі змінюються у відповідності алгоритму оптимізації Adam. Після чого все це знову повторюється протягом кількох епох, поки втрати не будуть мінімізовані.

Модель YOLOv8 використовує різні параметри, що можуть бути змінені, щоб впливати на процес навчання мережі. Оскільки навчання нейронної мережі для такого завдання є трудомістким та затратним по часу процесом, а ресурси були обмежені, більшість значень були обрані за замовчуванням.

Більшість параметрів, що використовувались в навчанні була зведена в Таблиця 4.2

Таблиця 4.2

Параметр	Значення
Epochs	200
save_period	10
batch	4
optimizer	adamW
lr	0,0005
momentum	0,9
patience	50
Weigh_decay	0,0005
df1	1,5
warmup_epochs	3
box	7,5
cls	0,5

#### 4.4 Отримані результати

Для початку пересвідчимося, що наша мережа може розпізнавати зображення без шуму, тобто в даному випадку знешумлюючий автокодер не активний, покажемо результат на Рисунок 4.12.

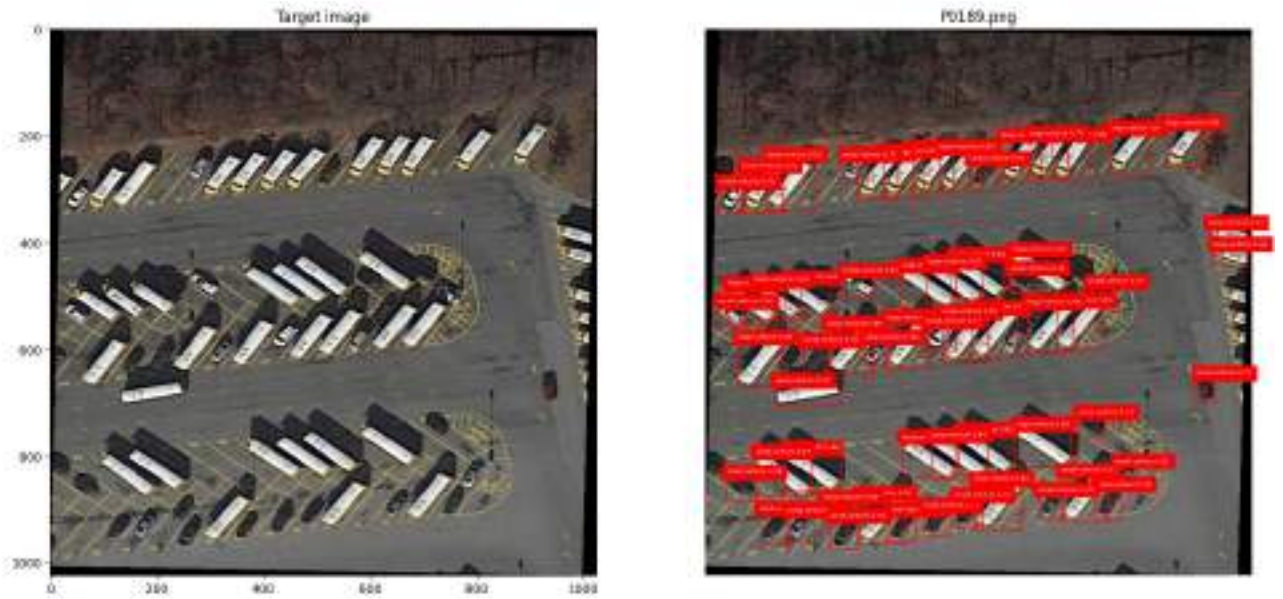


Рисунок 4.12 — Результат розпізнавання НМ об'єктів на знімку без внесеного шуму

Покажемо більш детально саме частину розпізнаних об'єктів на Рисунок 4.13.



Рисунок 4.13 — Розпізнані об'єкти на знімку без шуму

Далі внесемо шум на це ж зображення та пропустимо його через мережу без активованого автокодера, результат буде показано на Рисунок 4.14.

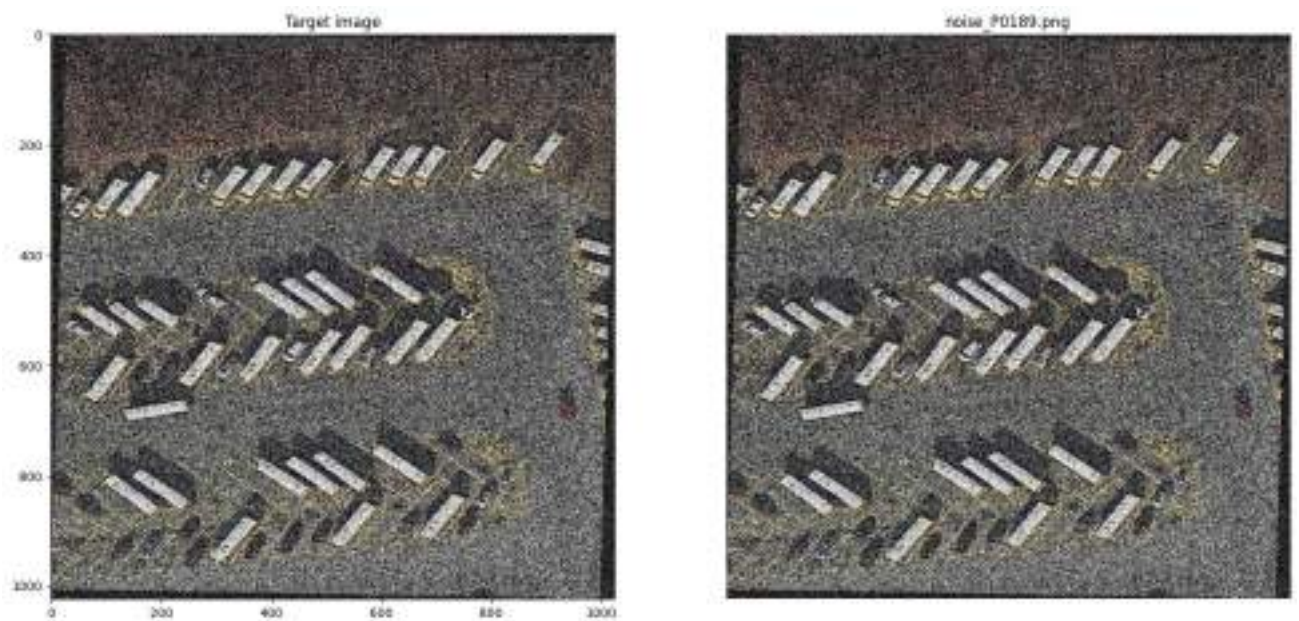


Рисунок 4.14 — Внесений шум на зображення та результат розпізнавання НМ

Як можемо бачити НМ без активованого знешумлюючого автокодера не здатна розпізнати об'єкти на зашумленому знімку.

Отже, наступним кроком буде активація автокодера та перевірка результату розпізнавання, що буде відображено на Рисунок 4.15.

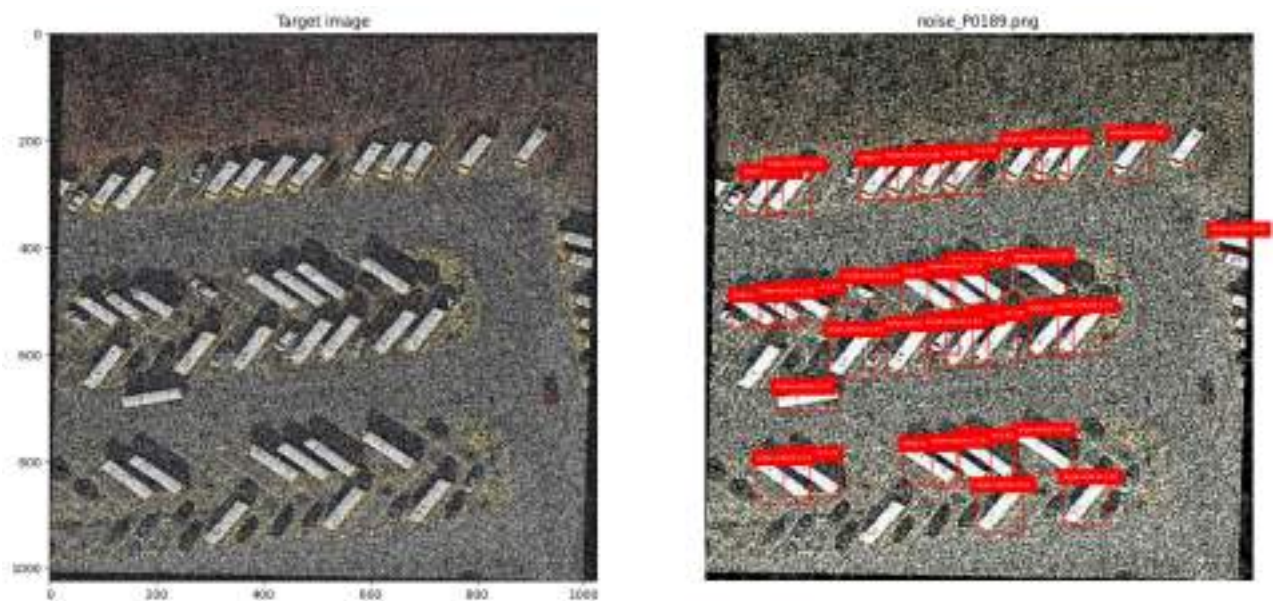


Рисунок 4.15 — Результат розпізнавання НМ з активованим автокодером



Покажемо більш детально частину розпізнаних об'єктів після роботи автокодера на Рисунок 4.16.

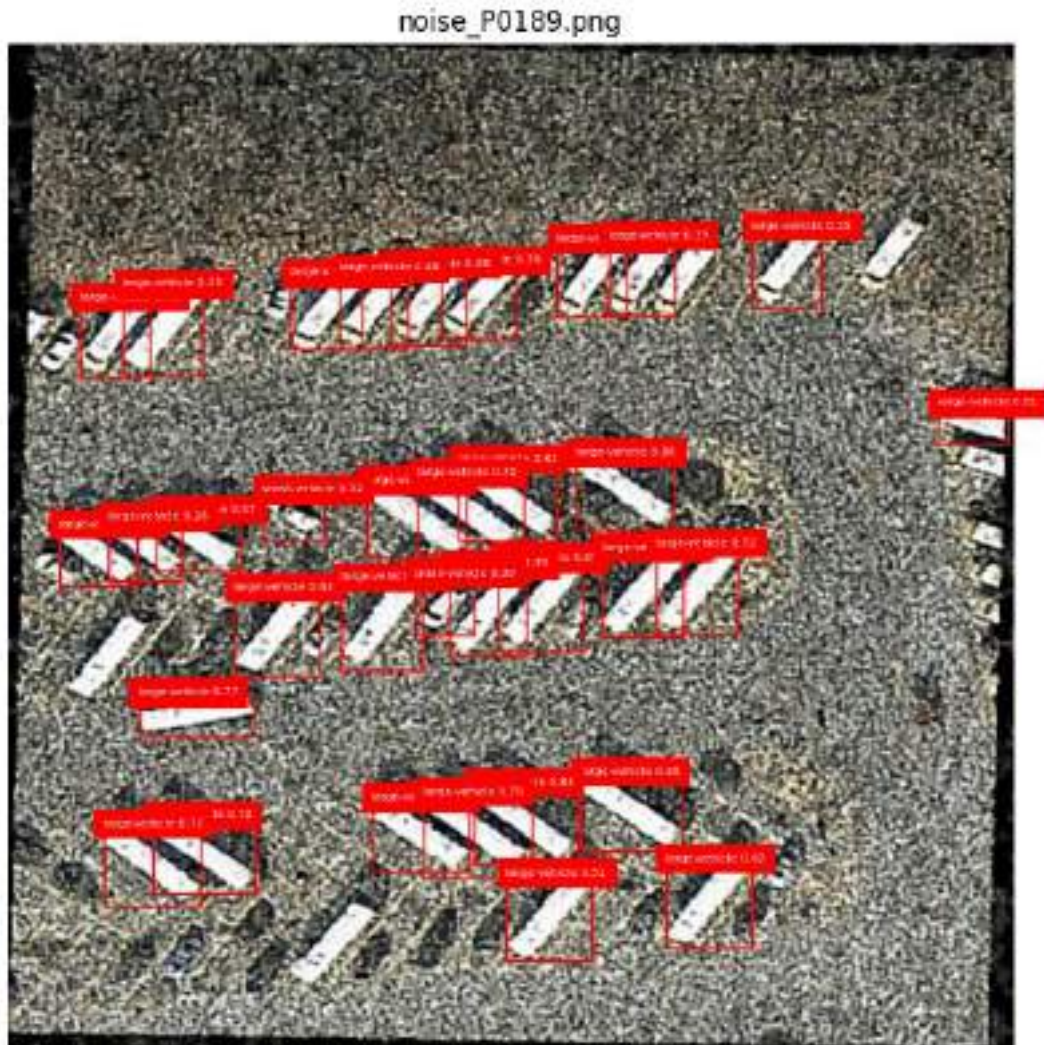


Рисунок 4.16 — Результат розпізнавання НМ з активованим автокодером (детальніше)

Можемо бачити, що НМ з одночасною роботою автокодера показує непогані результати, тобто проаналізувавши та порівнявши Рисунок 4.14 та Рисунок 4.15 можна зазначити, що запропонований метод справляється з поставленою задачею. Проте також слід зауважити, що деякі об'єкти залишились не розпізнаними НМ, зокрема це стосується класу *small-vehicle* та деяких об'єктів класу *large-vehicle*. Це можна пояснити тим, що у ході дослідження виявилось, що НМ YOLO може не справлятися з виявленням дрібних об'єктів, що перекриваються, а враховуючи, що шум вносить свій вплив на зображення та за до-

помогою знешумлюючого автокодера не можна досягти ідеального зменшення впливу шуму, то виходить так, що відбувається перекриття дрібних об'єктів, що й унеможлиблює здатність НМ їх розпізнати. Рішенням може бути подальше покращення роботи автокодера, шляхом його навчання на більш розширеному наборі даних або можливо використання інших нейронних мереж таких як GAN (Generative Adversarial Networks).

Покажемо розпізнавання на зашумлених знімках ще деяких класів:

Спочатку покажемо розпізнавання НМ об'єктів на знімках без шуму на Рисунок 4.17.



Рисунок 4.17 — Розпізнавання класу літак на зображенні без шуму (автокодер вимкнено)

Після чого додамо знову додамо шум до знімку та спробуємо розпізнати за допомогою НМ об'єкти, при цьому автокодер залишається вимкнутим, результат відобразимо на Рисунок 4.18.



Рисунок 4.18 — Розпізнавання класу літак на зображенні з шумом (автокодер вимкнено)

Можемо бачити, що НМ без автокодера не спроможна розпізнати клас «літак», лише один об'єкт і той помилково.

Далі ввімкнемо автокодер та пропустимо через мережу те ж зображення, результат покажемо на Рисунок 4.19.



Рисунок 4.19 — Розпізнавання класу літак на зображенні з шумом (автокодер увімкнено)

Для більшої наглядності повторимо те ж саме, але уже для інших класів та результати покажемо на наступних рисунках: Рисунок 4.20, Рисунок 4.21; Рисунок 4.22, Рисунок 4.23 та Рисунок 4.24.

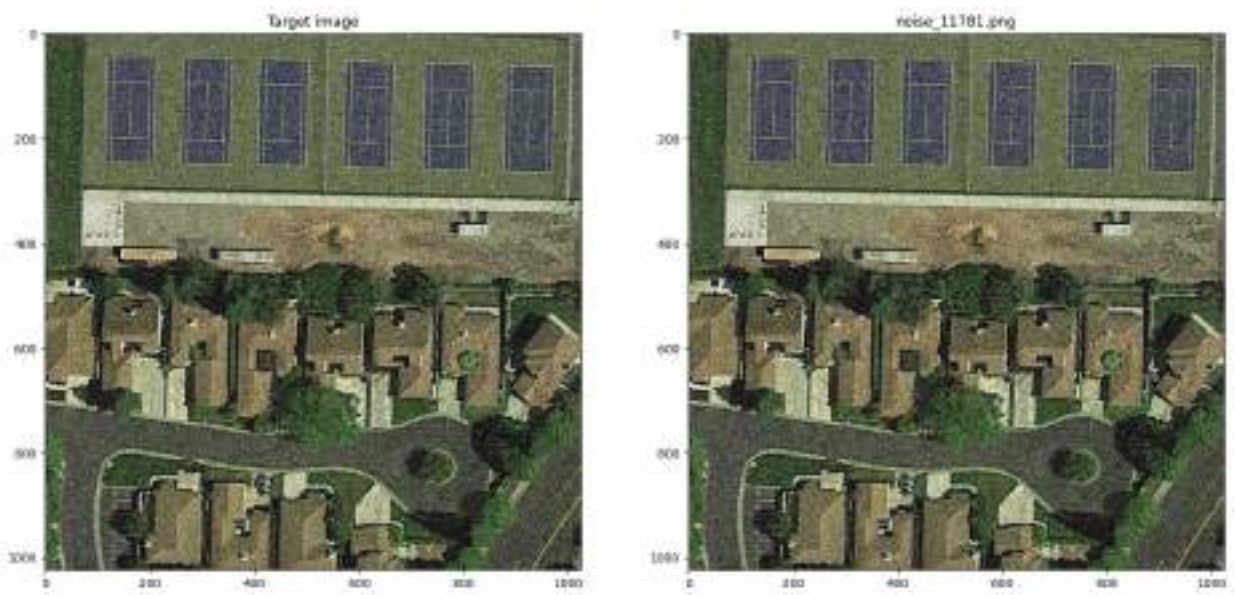


Рисунок 4.20 — Розпізнавання класу тенісний корт на зображенні з шумом (автокодер вимкнено)

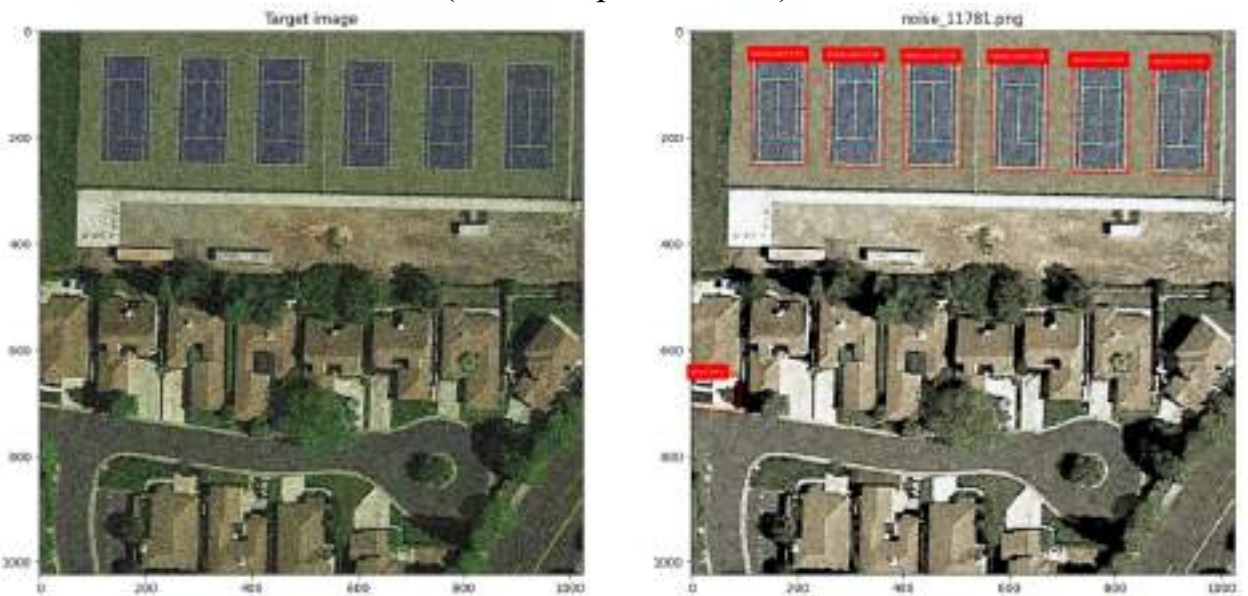


Рисунок 4.21 — Розпізнавання класу тенісний корт на зображенні з шумом (автокодер увімкнено)



Рисунок 4.22 — Розпізнавання класу спортивний майданчик з біговими доріжками та футбольне поле на зображенні з шумом (автокодер вимкнено)



Рисунок 4.23 — Розпізнавання класу спортивний майданчик з біговими доріжками та футбольне поле на зображенні з шумом (автокодер увімкнено)

Як можемо бачити з Рисунок 4.22, мережа розпізнала спортивний майданчик з біговими доріжками з ймовірністю 51%, враховуючи, що на зображенні наявний шум, а в свою чергу з Рисунок 4.23 видно, що після роботи автокодера, НМ розпізнала той самий спортивний майданчик з біговими доріжками з ймовірністю 76%, а також було розпізнано додатково футбольне поле 30%.

Окрім цього з останнього рисунка можемо помітити, не коректне розпізнавання класу «корабель» та «малий транспортний засіб», причини цього були описані вище.

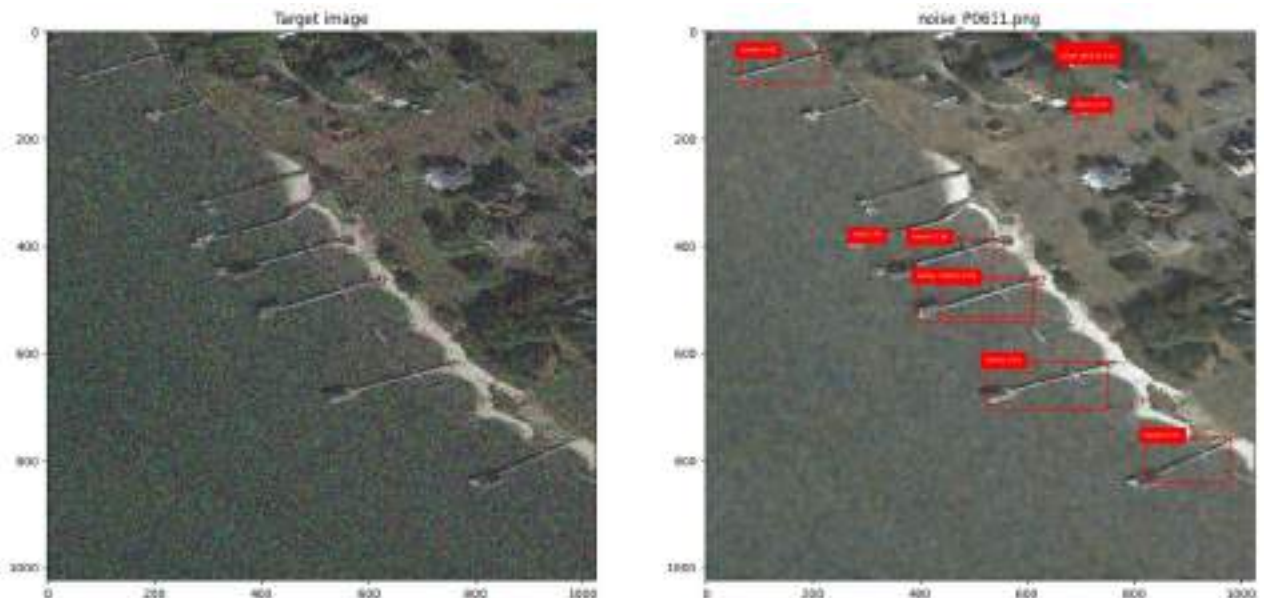


Рисунок 4.24 — Розпізнавання класу причал та корабель на зображенні з шумом (автокодер увімкнено)

З цього рисунку бачимо, що помилково виявлено було тільки клас літак, проте класи «причал» та «корабель» було виявлено коректно.

#### 4.5 Висновки до четвертого розділу

В даному пункті було описано процес підготовки наборів даних для навчання НМ, вказані основні датасети, що були використані, формат анотацій та власне класи об'єктів та їх кількість, окрім цього було описано процес доповнення набору даних шляхом зміни оригінальних зображень, а саме видалення фрагментів із зображень, обертання зображень та перекис. Після чого було описано процес навчання знешумлюючого автокодера та параметрів, які були застосовані у ході його навчання. Також було описано навчання мережі YOLOv8 для задачі розпізнавання об'єктів, вказані основні параметри, які були використані для навчання НМ. Останнім підпунктом була демонстрація отриманих результатів розпізнавання об'єктів на зашумлених супутникових знімках.

## ВИСНОВКИ

У даній роботі було запропоновано та реалізовано модель нейронної мережі для задачі розпізнавання об'єктів на зашумлених супутникових знімках, яка складається з автокодера для попередньої обробки зображень, а саме зменшення шуму та архітектури YOLOv8 для власне розпізнавання об'єктів певних класів. Для реалізації цього було проведено збір та підготовку даних, навчання автокодера та мережі YOLOv8, після чого було проведено тестування та оцінку отриманих результатів роботи моделі на супутникових знімках з шумом.

У ході тестування, спочатку було показано, що обрана мережа має здатність розпізнавати об'єкти на супутникових знімках без внесеного шуму при цьому автокодер було вимкнено: результати розпізнавання для класу «large vehicle» варіюються 0.48 – 0.91, для класу «small vehicle» 0.32 – 0.75, а для класу «plane» 0.80 – 0.88. Після додавання шуму на супутникове зображення, як можна було бачити з наведених рисунків, мережа без автокодера не була здатна розпізнати об'єкти. Після активації автокодера та подачі на НМ тих же знімків можна було бачити розпізнані об'єкти. Так, клас «тенісний корт» було розпізнано з ймовірністю 0.94 – 0.95 після активації автокодера, до його ж активації розпізнати ці об'єкти НМ не могла. Класи «спортивний майданчик з біговими доріжками» НМ розпізнала з ймовірністю 0.51 та клас «футбольне поле» розпізнати не вдалося, коли був вимкнений автокодер. Після активації автокодера мережа розпізнала той самий спортивний майданчик з біговими доріжками з ймовірністю 0.76, а також було розпізнано додатково футбольне поле з ймовірністю 0.3. Також 0.26 – 0.61 НМ розпізнала клас «причал» з активованим автокодером.

Також варто зазначити, що мають місце певні негативні результати. А саме, деякі об'єкти залишились не розпізнаними НМ після роботи автокодера. Це можна пояснити тим, що у ході дослідження виявилось, що НМ YOLO

може не справлятися з виявленням дрібних об'єктів, що перекриваються, а враховуючи, що шум вносить свій вплив на зображення та за допомогою знешумлюючого автокодера не можна досягти ідеального зменшення впливу шуму, то має місце перекриття дрібних об'єктів, що й унеможлиблює здатність НМ їх розпізнати. Також має місце не коректне розпізнавання (тобто розпізнавання хибних класів), в основному на оброблених автокодером знімках. Це вирішується розширенням навчальної вибірки для мережі YOLO. Такі результати спонукають до проведення подальших досліджень у цій сфері та удосконалення моделі в плані покращення точності та надійності розпізнавання об'єктів, шляхом удосконалення ланки автокодера, шляхом його навчання на більш розширеному наборі даних або застосування інших комбінацій підходів машинного навчання. Також можливе покращення моделі шляхом реалізації розпізнавання об'єктів в режимі реального часу.

В підсумку було отримано модель нейронної мережі, яка продемонструвала гарну точність у розпізнаванні об'єктів на супутникових знімках з шумом. Практичне застосування автокодера та моделі YOLOv8 підтверджує їхню ефективність у вирішенні задач зменшення шуму та розпізнавання об'єктів на супутникових зображеннях з шумом, що відкриває перспективи для подальших досліджень та розвитку в цій галузі.



## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

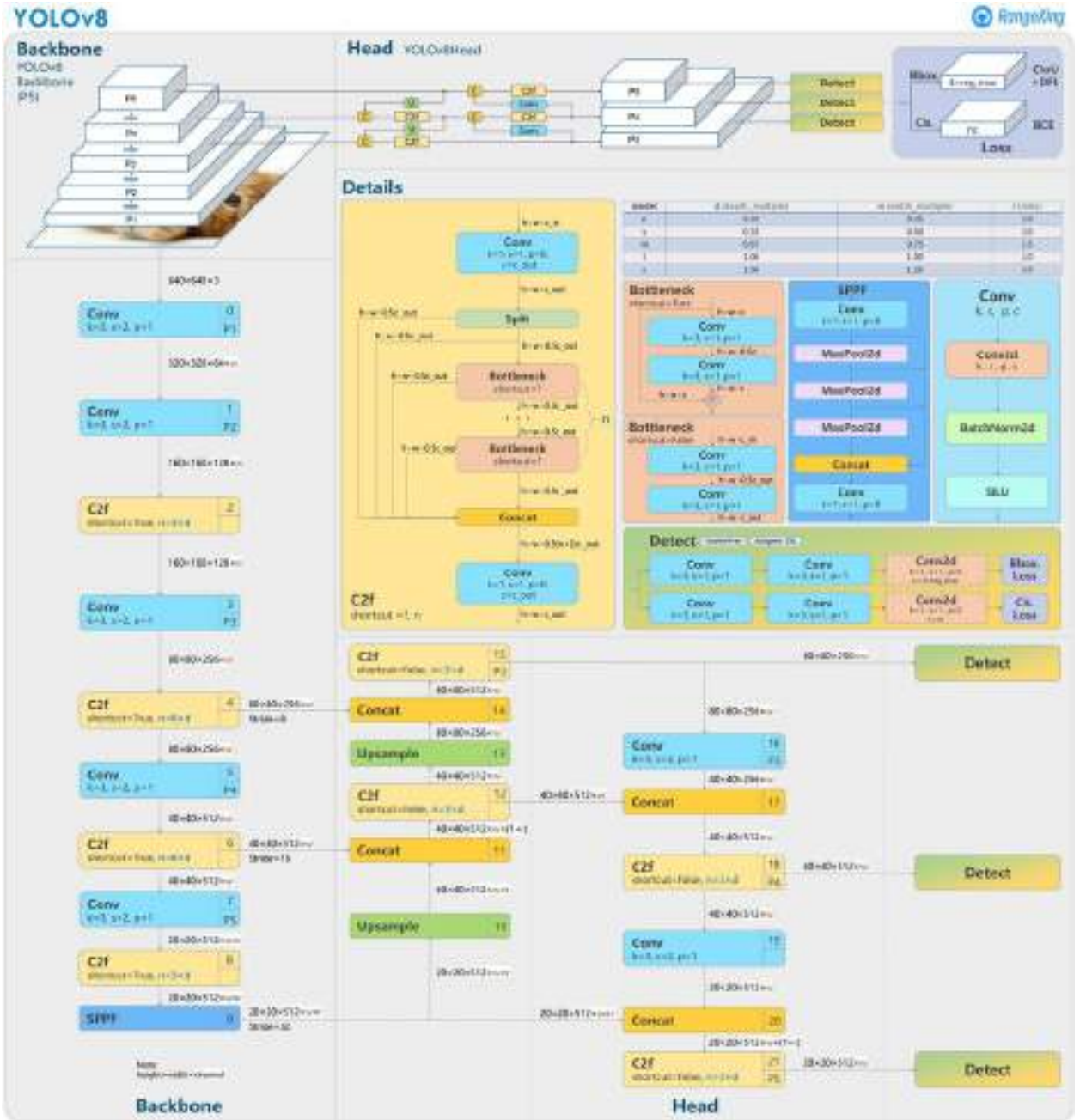
1. Зацерковний В. І. АЕРОКОСМІЧНІ ДОСЛІДЖЕННЯ ЗЕМЛІ: ІСТОРІЯ СТАНОВЛЕННЯ / В. І. Зацерковний, Н. П. Каревіна – Київ: Логос, 2014. – (Том 1). – С. 63–67.
2. Голубина фотозйомка [Електронний ресурс] – Режим доступу до ресурсу: <https://w.wiki/8fMr>.
3. Satellite imagery [Електронний ресурс] – Режим доступу до ресурсу: [https://en.wikipedia.org/wiki/Satellite\\_imagery](https://en.wikipedia.org/wiki/Satellite_imagery).
4. What is Remote Sensing? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.earthdata.nasa.gov/learn/backgrounders/remote-sensing>.
5. КУРС ЛЕКЦІЙ \ "НЕЙРОННІ МЕРЕЖІ\ " [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: <https://www.victoria.lviv.ua/library/students/nn/lecture.html>.
6. Artificial neural network [Електронний ресурс] – Режим доступу до ресурсу: <https://w.wiki/8fNr>.
7. Адаменко В. О. Штучні нейронні мережі. ТЕОРІЯ ТА АЛГОРИТМИ МАШИННОГО НАВЧАННЯ : конспект лекцій.
8. Deep Learning Fundamentals - Classic Edition [Електронний ресурс] – Режим доступу до ресурсу: <https://deeplizard.com/learn/video/DEMmkFC6IGM>.
9. Baheti P. What is Overfitting in Deep Learning [Електронний ресурс] / Pragati Baheti – Режим доступу до ресурсу: <https://www.v7labs.com/blog/overfitting#h4>.
10. McGonagle J. Backpropagation [Електронний ресурс] / J. McGonagle, G. Shaikouski, C. Williams – Режим доступу до ресурсу: <https://brilliant.org/wiki/backpropagation/>.
11. Метод зворотного поширення помилки [Електронний ресурс] – Режим доступу до ресурсу: <https://w.wiki/8etg>.
12. Адаменко В. О. Розпізнавання мовлення. ТЕОРІЯ ТА АЛГОРИТМИ МАШИННОГО НАВЧАННЯ : конспект лекцій.

13. Адаменко В. О. Оброблення зображень. Теорія та алгоритми машинного навчання : конспект лекцій.
14. Sumit S. A Guide to Convolutional Neural Networks — the ELI5 way [Електронний ресурс] / Saha Sumit – Режим доступу до ресурсу: <https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/>.
15. Pujara A. Concept of AlexNet:- Convolutional Neural Network [Електронний ресурс] / Abhijeet Pujara – Режим доступу до ресурсу: <https://medium.com/analytics-vidhya/concept-of-alexnet-convolutional-neural-network-6e73b4f9ee30>.
16. Zasserman A., Simonyan K. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2016.
17. Boesch G. VGG Very Deep Convolutional Networks (VGGNet) – What you need to know [Електронний ресурс] / Gaudenz Boesch – Режим доступу до ресурсу: <https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/>.
18. ResNet (34, 50, 101)...what actually it is ? [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: <https://medium.com/@aschandinip/resnet-34-50-101-what-actually-it-is-c63da24ba695>.
19. Shalgi C. Residual Net - ResNet [Електронний ресурс] / Carmel Shalgi. – 2020. – Режим доступу до ресурсу: <https://www.linkedin.com/pulse/residual-net-resnet-carmel-shalgi>.
20. Rich feature hierarchies for accurate object detection and semantic segmentation / R. Girshick et al. Computer Vision and Pattern Recognition. 2014.
21. Girshick R. Fast R-CNN / Ross Girshick. // Computer Vision and Pattern Recognition. – 2015.
22. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks / S.Ren, K. He, R. Girshick, J. Sun. – 2016.
23. You Only Look Once: Unified, Real-Time Object Detection / J.Redmon, S. Divvala, R. Girshick, A. Farhadi. – 2016.

24. Kundu R. YOLO: Algorithm for Object Detection Explained [Электронный ресурс] / Rohit Kundu. – 2023. – Режим доступа до ресурсу: <https://www.v7labs.com/blog/yolo-object-detection>.
25. Sahota H. The History of YOLO Object Detection Models from YOLOv1 to YOLOv8 [Электронный ресурс] / Harpreet Sahota. – 2023. – Режим доступа до ресурсу: <https://deci.ai/blog/history-yolo-object-detection-models-from-yolov1-yolov8/>.
26. SSD: Single Shot MultiBox Detector [Электронный ресурс] / [W. Liu, D. Anguelov, D. Erhan та ін.]. – 2016. – Режим доступа до ресурсу: [http://www.cs.unc.edu/%7Ewliu/papers/ssd\\_eccv2016\\_slide.pdf](http://www.cs.unc.edu/%7Ewliu/papers/ssd_eccv2016_slide.pdf).
27. Ultralytics YOLOv8 [Электронный ресурс] – Режим доступа до ресурсу: <https://github.com/ultralytics/ultralytics>.
28. Quality control and class noise reduction of satellite image time series / L. A. Santos et al. ISPRS Journal of Photogrammetry and Remote Sensing. 2021. Vol. 177. P. 75–88.
29. Sivarajah I. How to Reduce Noise in Satellite Imaging [Электронный ресурс] / Ilamaram Sivarajah. – 2023. – Режим доступа до ресурсу: <https://www.azooptics.com/Article.aspx?ArticleID=2384#>.
30. Bandyopadhyay H. Autoencoders in Deep Learning: Tutorial & Use Cases [2023] [Электронный ресурс] / Hmrishav Bandyopadhyay – Режим доступа до ресурсу: <https://www.v7labs.com/blog/autoencoders-guide>.
31. Oguntayo S. How to Use Autoencoders for Image Denoising [Электронный ресурс] / S. Oguntayo, M. Abrahamian. – 2022. – Режим доступа до ресурсу: <https://www.omdena.com/blog/denoising-autoencoders>.
32. DARICI M. B., ERDEM Z. A Comparative Study on Denoising from Facial Images using Convolutional Autoencoder. GAZI UNIVERSITY JOURNAL OF SCIENCE. 2022.
33. Subramanyam V. S. IOU (Intersection over Union) [Электронный ресурс] / Vineeth Subramanyam. – 2021. – Режим доступа до ресурсу: <https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef>.

34. Mittapally N. What's Precision and Recall? [Электронный ресурс] / Nitin Mittapally // *Becoming Human: Artificial Intelligence Magazine*. – 2019. – Режим доступа до ресурсу: <https://becominghuman.ai/whats-recall-and-precision-4a801b1ac0da>.
35. YOLOv8 model structure by RangeKing [Электронный ресурс]. – 2023. – Режим доступа до ресурсу: <https://github.com/ultralytics/ultralytics/issues/189>.
36. Anchor Free Detections, Reg\_max, and Prediction Distributions #6982 [Электронный ресурс]. – 2023. – Режим доступа до ресурсу: <https://github.com/ultralytics/ultralytics/issues/6982>.
37. DOTA dataset [Электронный ресурс] – Режим доступа до ресурсу: <https://captain-whu.github.io/DOTA/dataset.html>.
38. xView: Objects in Context in Overhead Imagery / [D. Lam, R. Kuzma, K. McGee та ін.]. // *Computer Vision and Pattern Recognition*. – 2018.
39. xView dataset [Электронный ресурс] – Режим доступа до ресурсу: <https://challenge.xviewdataset.org/download-links>.
40. Object detection in optical remote sensing images: A survey and a new benchmark / K. Li et al. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2020. Vol. 159. P. 296–307.

ДОДАТОК А



**ДОДАТОК Б**

```
import keras
from keras import layers
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')

%cd drive/My Drive/denoise_satellite

%pip install ultralytics
import ultralytics
ultralytics.checks()

from google.colab.patches import cv2_imshow
import cv2
from PIL import Image
import numpy as np
import os
import torch
from ultralytics import YOLO

model = YOLO('/content/drive/MyDrive/denoise_satellite/best.pt')

import locale
def getpreferredencoding(do_setlocale = True):
    return "UTF-8"
```

```
locale.getpreferredencoding = getpreferredencoding

dimension = 1024
input_img = keras.Input(shape = (dimension, dimension, 3))
x = layers.Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')(input_img)
x = layers.MaxPooling2D(pool_size = (2, 2), padding = 'same')(x)
x = layers.Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')(x)
encoded = layers.MaxPooling2D(pool_size = (2, 2), padding = 'same')(x)

x = layers.Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')(encoded)
x = layers.UpSampling2D(size = (2, 2))(x)
x = layers.Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu', padding = 'same')(x)
x = layers.UpSampling2D(size = (2, 2))(x)
decoded = layers.Conv2D(filters = 3, kernel_size = (3, 3), activation = 'sigmoid', padding = 'same')(x)

autoencoder = keras.Model(input_img, decoded)
autoencoder.summary()

checkpoint_path = "/content/drive/MyDrive/denoise_satellite/encoder_weights/cp.ckpt"
autoencoder.load_weights(checkpoint_path)

items = os.listdir('/content/drive/MyDrive/denoise_satellite/noised_images/')
sorted_items = sorted(items)
```

```
for name in sorted_items:

    # Open the image using PIL
    img = Image.open(f'/content/drive/MyDrive/denoise_satellite/noised_images/{name}') # Replace with the path to your image file
    image_array = img.resize((1024, 1024)) # Resize images to a specific size if needed

    # Convert the image to a NumPy array
    image_array = np.array(image_array, dtype=object)

    # Display the shape of the array
    print("Image shape:", image_array.shape)

    image_array = image_array[:, :, :3]
    print("Image shape:", image_array.shape)

    check_array_4d = image_array[np.newaxis, ...]
    norm_factor = 255.
    check_denoised_images = check_array_4d.astype('float32')/norm_factor

    check_norm = check_denoised_images

    # this lines activates encoder. uncomment if you want to use encoder
    check_denoised_images = autoencoder.predict(check_denoised_images)

    n = 1
    for i in range(n):
        fig, axes = plt.subplots(1, 2)
        fig.set_size_inches(18, 12)
```



```
axes[0].set_title('Target image')
im1 = axes[0].imshow(check_norm[i], cmap = 'Reds')
axes[1].set_title('Denoised image')
im2 = axes[1].imshow(check_denoised_images[i], cmap = 'Reds')
plt.savefig(f'comparison-{i}.png')

denoised_img = np.moveaxis(check_denoised_images[0], -1, 0)
img = torch.from_numpy(denoised_img).unsqueeze(0)

true_classes = {}
true_classes[0] = 'plane'
true_classes[1] = 'ship'
true_classes[2] = 'storage-tank'
true_classes[3] = 'baseball-diamond'
true_classes[4] = 'tennis-court'
true_classes[5] = 'basketball-court'
true_classes[6] = 'ground-track-field'
true_classes[7] = 'harbor'
true_classes[8] = 'bridge'
true_classes[9] = 'large-vehicle'
true_classes[10] = 'small-vehicle'
true_classes[11] = 'helicopter'
true_classes[12] = 'roundabout'
true_classes[13] = 'soccer-ball-field'
true_classes[14] = 'swimming-pool'
true_classes[15] = 'container-crane'

# name = '/content/drive/MyDrive/satellite/yolov5/heli.jpeg'

frame = check_denoised_images[0]
```

```
font = cv2.FONT_HERSHEY_SIMPLEX
fontScale = 1
color = (255, 0, 0)
thickness = 2

results = model(img, verbose=False)[0]

boxes = results.bboxes.data.tolist()

# Visualize the image and boxes
# plt.figure(figsize=(8, 8))
axes[1].set_title(name)

axes[1].imshow(frame)
plt.axis('on')

calculate_classes = {}

if boxes is not None:
    # Draw bounding boxes on the image
    for box in boxes:
        x1, y1, x2, y2, conf, cls = box
        box_w = x2 - x1
        box_h = y2 - y1
        plt.plot([x1, x1, x1+box_w, x1+box_w, x1], [y1, y1+box_h, y1+box_h,
y1, y1], color='r', linewidth=1)
        cls_true = true_classes[int(cls)]

        if cls_true in calculate_classes.keys():
            calculate_classes[cls_true] += 1
```

```
else:
    calculate_classes[cls_true] = 1
    plt.text(x1, y1, f'{cls_true} {conf:.2f}', color='white', fontsize=5, back-
groundcolor='red')

plt.show()
print(calculate_classes)
```