Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра моделювання складних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня бакалавра
за спеціальністю 113 «Прикладна математика»

на тему:

Пошук об'єктів в зображенні за допомогою нейронних мереж

студента 4 курсу
Мушти Іллі Андрійовича

_____

Науковий керівник:
доцент, кандидат фізико–математичних наук
Матвієнко Володимир Тихонович

_____

Робота заслухана на засіданні кафедри моделювання складних систем та рекомендована до захисту, протокол № від 2021 р.

Завідувач кафедри МСС

канд. фіз.-мат. наук, доцент Дмитро ЧЕРНІЙ

Київ – 2021

РЕФЕРАТ

Обсяг роботи 105 сторінок, 94 ілюстрації, 44 джерела посилання.

Ключові слова: навчання, зображення, шар, мережа, пошук, функція, алгоритм, об’єкт, метод, ознака, нейрон, рамка, модель, активація, пошук об’єктів в зображенні, ваги, дані, машинне навчання, виявлення, параметри, нейронна мережа, згортка, детектор.

Об’єктом роботи є процес пошуку об’єктів в зображенні за допомогою програмного засобу. Предметом роботи є програмний засіб для розв’язування задач пошуку об’єктів в зображенні.

Метою роботи є створення програмного засобу для розв’язування задач пошуку об’єктів в зображенні.

Інструменти розроблення: мова програмування Python, дистрибутив Anaconda, середовище Conda, Jupyter Notebook, бібліотека PyTorch, бібліотека numpy, бібліотека pandas, бібліотека PIL.

Результати роботи: виконано загальний огляд методів та алгоритмів пошуку об’єктів в зображенні; розглянуто основні поняття, будову, способи навчання та різні архітектури нейронних мереж; детально розглянуто згорткові нейронні мережі та детектор YOLO; розроблено програмний продукт, який дозволяє здійснювати пошук одного об’єкту вибраного класу на зображенні (single-object detection).

Програмний продукт для пошуку об’єктів в зображенні може застосовуватися у таких сферах як відеонагляд та безпека (розрізнавання облич, виявлення рухів), автомобільна галузь, споживчий ринок, медицина та охорона здоров’я, сільське господарство (наприклад, виявлення зрілості врожаю, забезпечення точності позиціонування засобів обробки землі), роздрібна торгівля, логістика, доставка товарів, виробництво, використання у військових цілях (наприклад, відеонагляд, автономні транспортні засоби, засоби знешкодження мінних полів, контроль якості виробництва боєприпасів).

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки. На сьогодні задачі автоматичного розпізнавання і пошуку положення об’єктів на забраженні та відео є надзвичайно важливими, адже їх вирішення надає можливість комп’ютерам взаємодіяти з зовнішним світом. Для того, щоб змусити комп’ютер працювати на людину або поруч із нею, необхідно щоб машина мала здатність розібратися в своїй сфері не гірше людини, а в деяких випадках навіть краще неї. Саме тому розв’язок проблеми розпізнавання об’єктів є ключем до розв’язку більш глобальної задачі змістового розуміння комп’ютером навколишнього світу.

Велика кількість науковців, представників промисловості та бізнесу вже зацікавились технологіями комп’ютерного зору,а саме їх застосуванням у таких напрямках як беспілотні автомобілі, приховане спостереження, надання допомоги при рятувальних операціях, розгортання роботів на підприємтвах, виявлення облич та пішоходів, розпізнавання брендів, створення візуальних ефектів на зображеннях, оцифровка тексту, обробка аерофотознімків і т. ін. [1].

Актуальність роботи та підстави для її виконання. Виявлення об’єктів в зображенні є складною задачею комп’ютерного зору, якій останні 20 років приділяють велику увагу особливо з розвитком глибокого навчання. Серед проблем пошуку об’єктів в зображенні, які впливають на точність виявлення, виділяють геометричні варіації, такі як зміна масштабу(наприклад, об’єкт має малі розміри порівняно з зображенням) та повортоти об’єктів, часткове перекриття об’єктів , умови освітлення (наприклад, зміни обумовлені погодніми умовами чи наявністю штучного світла). Також можлива комбінація всіх цих ефектів(наприклад, малий по відношенню до зображення, повернутий на певний кут об’єкт, який ще також перекривається іншими об’єктами). Іншим важливим аспектом крім точності виявлення об’єктів є також швидкість процесу виявлення [2].

Детектори об’єктів, які грунтуються на нейронних мережах, впродовж свого розвитку демонстрували найкращі результати на основних наборах даних. Ці детектори поділяють на дві категорії: двокрокові детектори та однокрокові детектори. Перший тип використовує мережу RPN (Region Proposal Network) для генерації областей інтересів на першому кроці, а далі відправляє ці пропозиції регіонів в конвеєр для класифікації об’єктів та регресії по обмежувальній рамці. Ці мережеві моделі забезпечують більш високу точність, але є порівняно повільними. До цієї групи відносяться Faster R-CNN та Mask R-CNN. З іншого боку, однокрокові детектори трактують задачу пошуку об’єктів як задачу регресії, приймаючи на вхід зображення та одночасно визначаючи ймовірності класів об’єктів та координати обмежувальних рамок. Ці моделі спершу давали гірші показники точності, але були набагато швидші за двокрокові моделі. До групи однокрокових моделей належать SSD(Single Shot MultiBox Detector) та YOLO(You Only Look Once) [2].

Саме тому актуальним є розгляд YOLO як достатньо швидкої та точної моделі детектору об’єктів.

Мета й завдання роботи. Метою кваліфікаційної роботи є створення програмного засобу для розв’язування задачі пошуку об’єктів в зображенні за допомогою нейронних мереж.

Для досягнення цієї мети поставлено такі завдання:

• Розглянути метод нейронних мереж для розв’язання задачі пошуку об’єктів в зображенні.

• Побудувати модель нейронної мережі для пошуку об’єктів в зображенні за умови, що кожне зображення містить лише один об’єкт. У якості тренувальних даних взяти набір даних Wildfire Smoke Dataset.

• Розробити програмний продукт.

Об'єкт, методи й засоби розроблення. Об’єктом стоврення програмного засобу є процес пошуку об’єктів в зображенні за допомогою програмного засобу.

Програмний продукт був написаний на мові Python. Для цього був використаний дистрибутив Anaconda, створене середовище Conda та

використаний Jupyter Notebook. У якості бібліотеки для машинного та глибокого навчання була використана PyTorch. PyTorch – це фреймворк машинного навчання для мови Python з відкритим вихідним кодом, який створений на базі Torch. Також були використані бібліотеки numpy (бібліотека з відкритим вихідним кодом для мови програмування Python, яка забезпечує підтримку багатовимірних масивів та високорівневих математичних функцій, призначених для роботи з багатовимірними масивами), pandas (програмна бібліотека для мови програмування Python, що пропонує структури даних та операції для маніпулювання чисельними таблицями та часовими рядами) та PIL (бібліотека з відкритим вихідним кодом для мови програмування Python, яка забезпечує роботу з растровою графікою).

Можливі сфери застосування. Програмний продукт для пошуку об’єктів в зображенні має надзвичайно широке поле для застосування. Серед можливих сфер застосування можна назвати такі як відеонагляд та безпека (розрізнавання облич, виявлення рухів, виявлення маршрутів переміщення людей, виявлення статичних об’єктів, захист приватності), автомобільна галузь (наприклад, виявлення бокового трафіку), споживчий ринок, медицина та охорона здоров’я, сільське господарство (наприклад, виявлення зрілості врожаю, забезпечення точності позиціонування засобів обробки землі), роздрібна торгівля(наприклад, відслідковування наявності повного асортименту товарів на полицях супермаркетів, контроль довжини черг на касах), логістика, доставка товарів, виробництво, використання у військових цілях(наприклад, відеонагляд, автономні транспортні засоби, засоби знешкодження мінних полів, контроль якості виробництва боєприпасів) [3].

1 МЕТОДИ ТА АЛГОРИТМИ ПОШУКУ ОБ‘ЄКТА В ЗОБРАЖЕННІ

1.1 Пошук об’єктів в зображенні як складова частина комп’ютерного зору, історія та розвиток дисципліни комп’ютерного зору.

Пошук об’єктів в зображенні – це комп’ютерна технологія, що відноситься до комп’ютерного зору та обробки зображень і пов’язана з виявленням екземплярів семантичних об’єктів конкретного класу (наприклад, люди, будівлі, автомобілі) в цифрових зображеннях та відео [4]. Найбільш дослідженими задачами пошуку об’єктів в зображенні є виявлення облич та виявлення пішоходів.

Як наукова дисципліна, комп’ютерний зір (а разом з ним і пошук об’єктів в зображенні, як його складова частина) є частиною теорії і технології створення штучних систем, що отримують інформацію з зображень [5]. Для таких штучних систем вхідні дані можуть надходити у вигляді великої різноманітності форм, таких як відеопослідовність, зображення з різних камер чи тривимірні дані, наприклад з медичного сканеру. Основним завданням комп’ютерного зору як технологічної дисципліни є застосування теорії та моделі комп’ютерного зору у створенні систем комп’ютерного зору. Прикладами застосування таких систем можуть бути такі [5]:

1. Системи управління процесами (промислові роботи, автономні транспортні засоби).
2. Системи відеонагляду.
3. Системи організації інформації(наприклад, для індексації баз даних зображень).
4. Системи моделювання об’єктів чи оточуючого середовища(аналіз медичних зображень, топографічне моделювання).
5. Системи взаємодії(наприклад, пристрої введення для системи взаємодії людина-машина).
6. Системи доповнюючої реальності.

7. Обчислювальна фотографія.

Також існує спосіб описання комп'ютерного зору як доповнення до біологічного зору. Створюються моделі роботи таких систем в термінах фізіологічних процесів, що спираються на біологічні дослідження зорового сприйняття людини та тварин. З іншого боку, комп'ютерний зір вивчає і описує системи комп'ютерного зору, які створені апаратно чи програмно. Обмін інформацією між дисциплінами біологічного та комп'ютерного зору є достатньо продуктивним для кожної з цих наукових областей. Підрозділи комп'ютерного зору включають, відтворення дій, виявлення подій, стеження за об'єктами, відновлення зображень і т. ін.

Область комп'ютерного зору є достатньо молодою і такою, що динамічно розвивається.

Можна стверджувати, що з кінця 1970-х років почалося швидке вивчення цієї проблеми, коли комп'ютери змогли керувати обробкою великої кількості даних, таких як зображення. Але ці дослідження починалися, як правило, з інших областей, і, відповідно, немає стандартного формулювання проблеми комп'ютерного зору. Також, що важливо, немає стандартного формулювання того, як повинна вирішуватися проблема комп'ютерного зору. Але замість цього існує велика кількість методів розв'язання різного виду чітко визначених задач комп'ютерного зору, де методи часто залежать від задач і рідко можуть бути узагальнені для широкого кола застосування. Багато методів все ще знаходяться на етапі фундаментальних досліджень, але що раз більша їх кількість застосовується в комерційних продуктах, де вони часто є складником якоїсь більшої системи яка може розв'язувати складні задачі (наприклад, в області медичних зображень або контролю якості в процесах випуску продукції) [5].

Важливою частиною в області штучного інтелекту є задача автоматичного планування чи прийняття рішень в системах, які можуть виконувати механічні дії (наприклад, переміщення робота через деяке середовище). Для цього типу обробки зазвичай необхідні вхідні дані, що надаються системами комп'ютерного зору, які діють як відеосенсор і надають високорівневу інформацію про

середовище і про робота. Інші області, які іноді розглядаються як такі, що належать до штучного інтелекту і використовуються відносно комп'ютерного зору, це розпізнавання образів та методи навчання. У результаті, комп'ютерний зір іноді розглядають як частину області штучного інтелекту чи області комп'ютерних наук взагалі [5].

1.2 Історія та розвиток дисципліни пошуку об'єктів в зображенні

Що стосується історії дисципліни пошуку об'єктів в зображенні, то її переважно поділяють на два великих періоди, відомі як традиційний період пошуку об'єктів в зображенні (до 2014 року) та період пошуку об'єктів в зображенні за допомогою глибокого навчання (після 2014 року). Сьогодні для задач пошуку об'єктів в зображенні використовують комп'ютери, тоді як для традиційного періоду було характерним використання людської праці для виділення ознак на зображеннях, а також була притаманна боротьба з обмеженими ресурсами і вирішення додаткових задач з вибору оптимального алгоритму для пришвидшення обрахунків. Однією з перших моделей для вирішення такого типу задач була модель детектора Віоли-Джонса (Viola-Jones(VJ) Detector). Він був створений ще у 2001 році та забезпечував виявлення людських облич у реальному часі. Це була перша модель, вільна від усілякого типу обмежень, що працювала на процесорі Pentium III 700MHz. Детектор мав у складі три важливих функціональних блоки: «Інтегральне зображення» , «Вибір характеристик» та «Каскади виявлення». Блок «Інтегральне зображення» забезпечував незалежність складності обрахунків від їх розмірів. Для вибору ознак використовувалася модель AdaBoost. Її завданням був вибір найбільш домінуючих ознак. В свою чергу блок «Каскади виявлення» допомагав визначати пріорітети для обрахунків на обличчях , а не на фоні [6].

Далі у 2005 році був створений HOG детектор (Histogram of Oriented Gradients (HOG) Detector). Цей детектор мав суттєві переваги над попередником, адже він збалансовував інваріантність функцій та нелінійність. Це вдалося за

допомогою проведення обчислень на сітці з рівномірно розташованих комірок та використанні нормалізації перекриваючого локального контрасту для збільшення точності [7]. Довгий час саме HOG був найбільш сучасною моделлю для пошуку об’єктів в зображенні.

Алгоритм HOG детектору зображений на рисунку 1.1.

Рисунок 1.1 – Алгоритм HOG детектору

Наступною після HOG була DPM модель (Deformable Part-Based Model (DPM)), запропонована у 2008 році. Були впроваджені деякі важливі методи, які містили «від’ємний аналіз», «регресію обмежувальних прямокутників», «контекстний праймінг». Був розроблений метод пришвидшення моделей пошуку об’єктів з використанням каскадної архітектури, який збільшує швидкість обчислень в 10 разів без жодних втрат у точності [8]. Узагалі в традиційний період було зроблено чимало зусиль, але вони у результаті не призвели до значних покращень.

З 2014 року починається новий період у розвитку дисципліни, так званий «Період пошуку об’єктів в зображенні за допомогою глибокого навчання». Була опублікована робота «Rich feature hierarchies for accurate object detection and semantic segmentation» [9]. Автори запропонували модель нейронної мережі RCNN (Region-based Convolution Neural Network) та техніку використання регіонів разом із згортковою нейронною мережею для пошуку об’єктів в зображенні.

Базова архітектура RCNN зображена на рисунку 1.2.

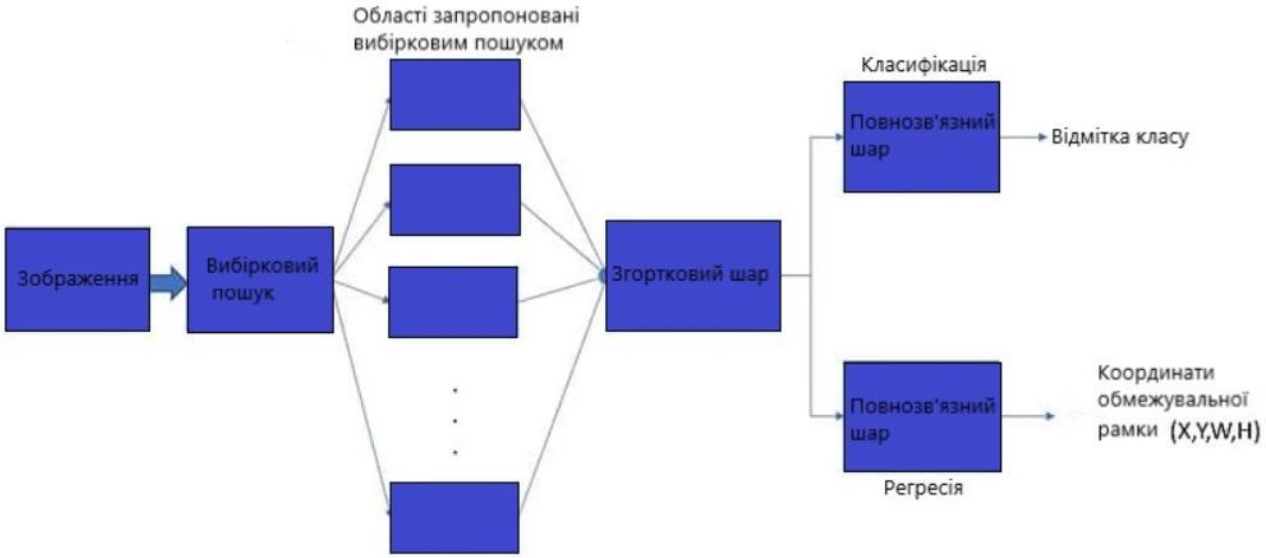


Рисунок 1.2 – Базова архітектура RCNN

RCNN використовувала алгоритм вибіркового пошуку для отримання 2000 областей інтересу з зображення, яке змінювалося до визначеного розміру і потім передавалося на вхід до згорткової нейронної мережі. Вихідні дані мережі подавалися в класифікатори SVM для класифікації об’єкту в кожній пропонованій області. Але модель вийшла надзвичайно повільною через надлишкові обчислення в області інтересу, що перекривається. Автори роботи запевняють, що запропонований алгоритм забезпечує відносне покращення результатів пошуку об’єктів на 30% порівняно з кращими попередніми алгоритмами. Такі результати були отримані завдяки впровадженню двох ідей. По-перше, використана модель згорткової нейронної мережі з великою пропускною спроможністю для локалізації та сегментації об’єктів. По-друге, використана парадигма навчання великої нейронної мережі за умов малої кількості помічених навчальних даних.

Наступною була запропонована модель нейронної мережі SPPNet (Spatial Pyramid Pooling Networks) [10]. Основною перевагою мережі SPPNet було впровадження Spatial Pyramid Pooling (SPP) шару, що дозволило згортковій нейронній мережі генерувати результат фіксованої довжини незалежно від розміру зображення.

На рисунку 1.3 зображена архітектура SPPNet.

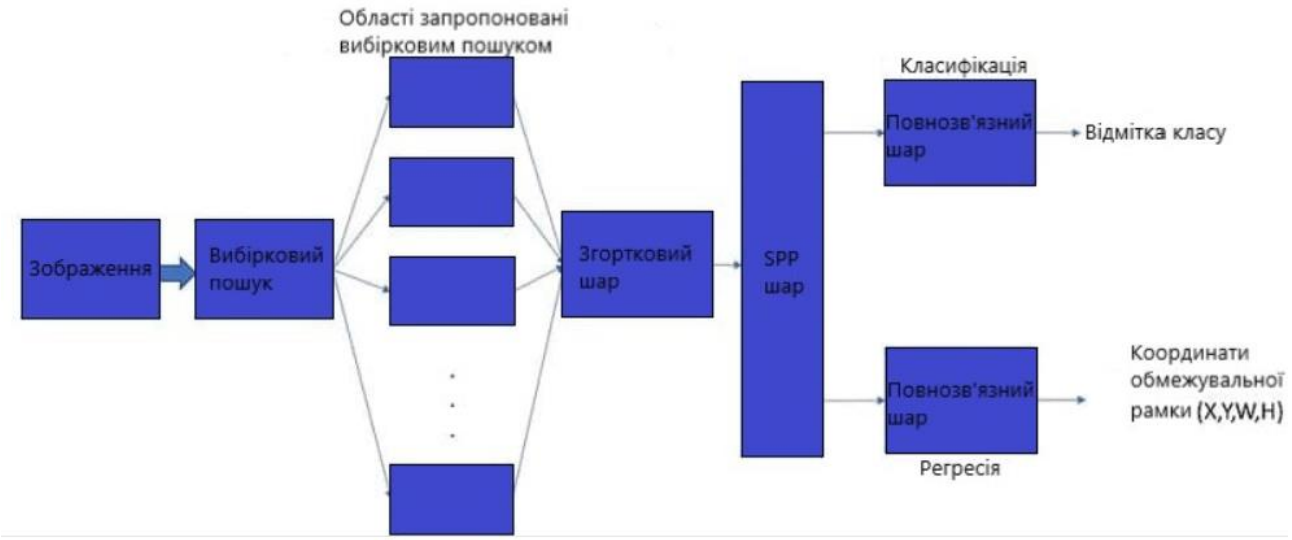


Рисунок 1.3 - Архітектура SPPNet

Тут карти ознак були обраховані лише один раз, а далі були створені представлення випадкової області фіксованої довжини для навчання детекторів, що дозволило уникнути повторного використання згорткових ознак. Архітектура SPPNet є більш ніж в 20 разів швидшою за архітектуру RCNN, при чому за таку швидкість не доводиться розплачуватися точністю. Серед недоліків архітектури можна вказати те, що навчання як і раніше відбувається в декілька етапів.

В кінці 2015 року був запропонований новий детектор FRCNN (Fast Region-based Convolution Neural Network), який долає деякі обмеження SPPNet та RCNN [11]. FRCNN дозволяє одночасно навчати детектор і регресор обмежувальної рамки за незмінних конфігурацій мережі. Цей детектор використовує згорткову нейронну мережу для надання карти ознак замість того, аби «пробігати» близько 2000 регіонів. Далі ця карта транслювалася у вихід алгоритму вибіркового пошуку і перенаправлялася на шар RoI Pooling, як показано на рисунку 1.4.

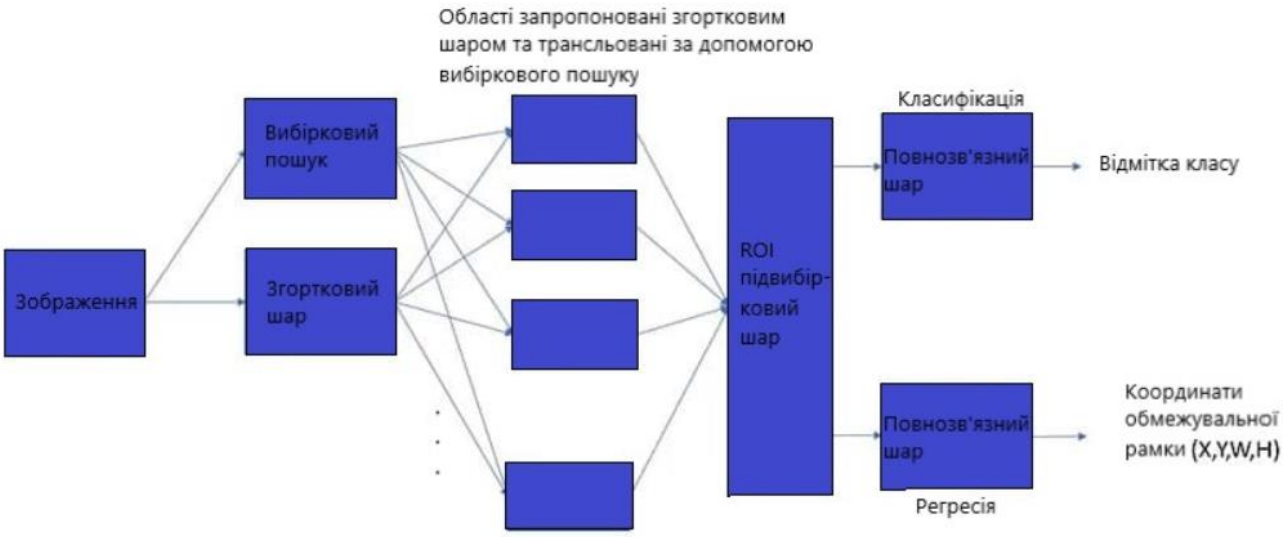


Рисунок 1.4 – Архітектура FRCNN

Тут шар RoI Pooling, який є модифіацією SPP шару, на відміну від SPP шару, використовує одношарове вікно. Це зменшило кількість обрахунків та затримку моделі. Але швидкість виявлення об’єктів як і раніше обмежувалася певними особливостями алгоритму. Це призвело до наступної розробки та появи Faster-RCNN.

У тому ж 2015 році був запропонований детектор Faster-RCNN, який був першим наскрізним детектором та першим детектором глибокого навчання, який працював майже у режимі реального часу [12]. Основною новацію детектора Faster-RCNN було введення мережі RPN (Region Proposal Network). Завдяки цьому час виявлення об’єкту для дектора Faster-RCNN становив лише 0.2 секунди.

На рисунку 1.5 показана архітектура Faster-RCNN.

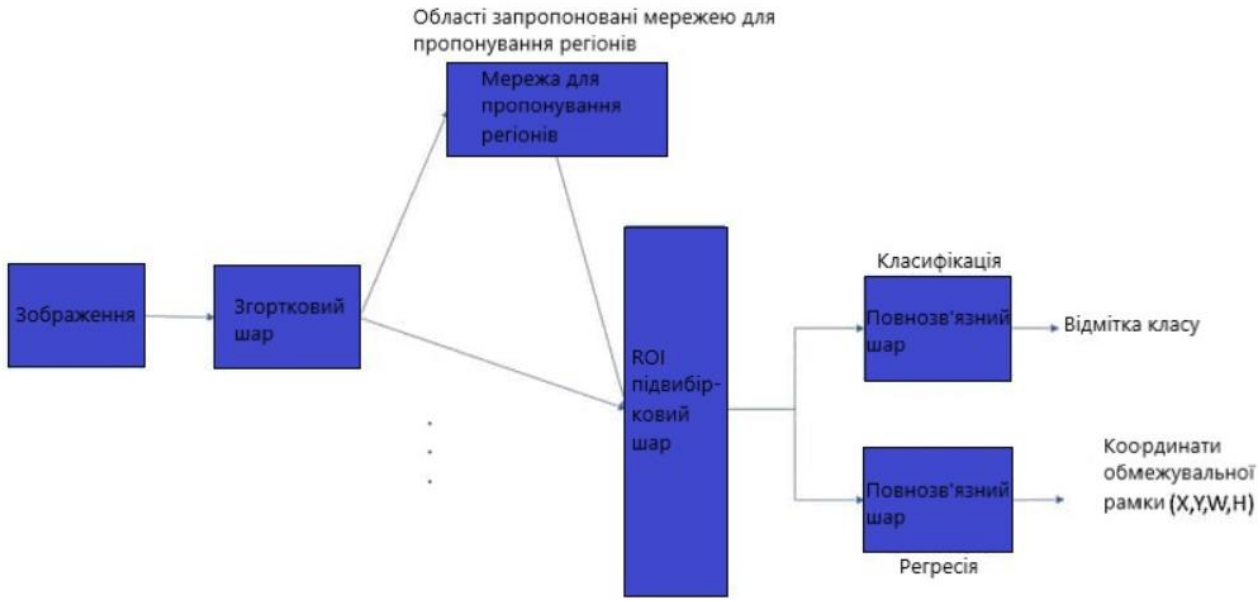


Рисунок 1.5 - Архітектура Faster-RCNN.

Також у 2015 році була запропонована модель YOLO (You Only Look Once) [13]. Не дивлячись на значне збільшення швидкості виявлення, для цієї моделі характерне падіння точності виявлення об’єктів у порівнянні з двокроковими детекторами, особливо для деяких невеликих об’єктів.

На рисунку 1.6 зображена архітектура моделі YOLO.

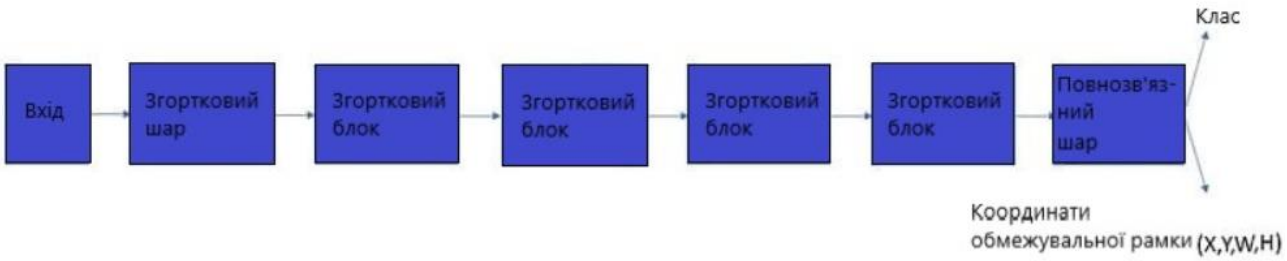


Рисунок 1.6 – Архітектура YOLO

Після моделі YOLO з’явилася модель SSD (Single Shot Multibox Detector) [14]. У моделі були застосовані нові техніки пошуку об’єктів, що значно покращило точність виявлення. Особливо це вплинуло на точність виявлення малих об’єктів, адже модель не включала багато шарів масштабування зображення у порівнянні з RCNN, Fast-RCNN та Faster-RCNN.

На рисунку 1.7 зображена архітектура моделі SSD.

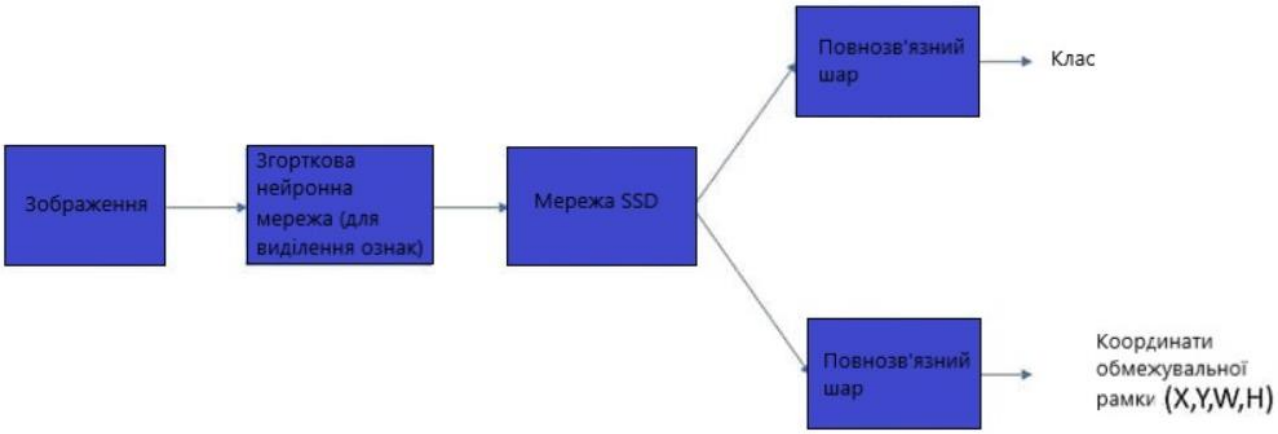


Рисунок 1.7 – Архітектура SSD

У 2017 році на основі моделі Faster-RCNN була запропонована модель FPN [15]. Було помічено, що виявлення ознак відбувалося на верхніх шарах мережі, тоді як більш глибокі шари не мали на це великого впливу. Відповідно була розроблена нова архітектура мережі з бічними з'єднаннями для побудови семантики високого рівня на всіх масштабах. Так як згорткова нейронна мережа природнім чином створює піраміду ознак у мережі прямого розповсюдження, єдиним варіантом було додавання бічних з'єднань, що допомогло покращити результати виявлення. Таким чином, FPN тепер є будівельним блоком побудови багатьох сучасних детекторів. У цьому ж році була введена нова функція втрат «Focal Loss», яка підвищила точність виявлення об'єктів моделями SSD і YOLO.

1.3 Методи та алгоритми пошуку об‘єкта в зображенні

1.3.1 Метод кольорових фільтрів

Одним з найпростіших методів пошуку об'єктів в зображенні є метод кольорових фільтрів [16]. Його можна застосовувати у тих випадках, коли об'єкт значно відрізняється від фону кольором а також освітлення рівномірне і не змінюється. Першим кроком методу є згладжування коливань кольору на зображенні, що пов'язані з технічними особливостями камерами та коливаннями освітлення. Для цього можна використати метод вирівнювання гістограми. У результаті вирівнювання гістограми зображення має багато невеликих за

розмірами областей різкої зміни інтенсивності кольорів. Для виправлення цього недоліку застосовують згладжування (розмиття). Після цих перетворень використовують кольорову фільтрацію, після чого зображення може містити точки, що не належать об’єкту, але близькі до нього за кольором. Для того, щоб позбутися цих точок можна перетворити зображення на чорно-біле і застосувати методи математичної морфології. Після цього виділяють краї та отримують контур шуканого об’єкта.

Процедура виконання даного алгоритму зображена на рисунках 1.8, 1.9, 1.10, 1.11, 1.12, 1.13.

Рисунок 1.8 – Початкове зображення

Рисунок 1.9 – Зображення після процедури вирівнювання гістограми

Рисунок 1.10 – Зображення після згладжування

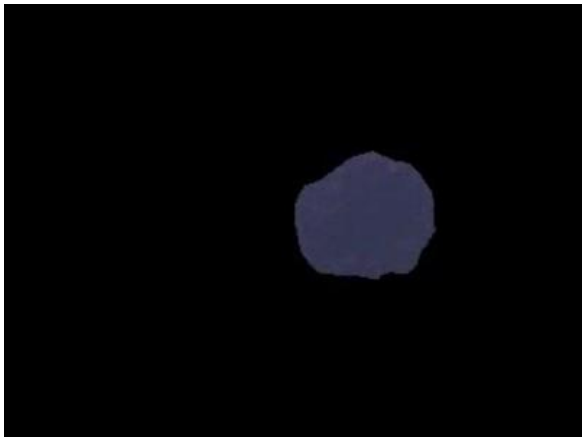


Рисунок 1.11 – Зображення після процедури кольорової фільтрації

Рисунок 1.12 – Зображення після очищення морфологією

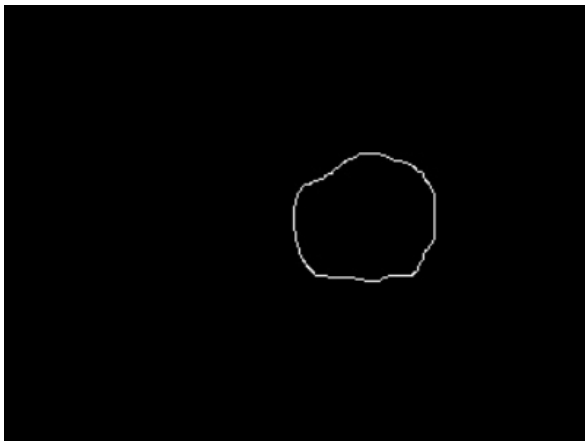


Рисунок 1.13 – Зображення після виділення країв

Рисунок 1.14 – Результат виконання алгоритму

1.3.2 Метод виділення та аналізу контурів

Якщо колір об’єкту суттєво не відрізняється від кольору фону, то використання методу кольорових фільтрів не дає гарних результатів. В такому випадку можна спробувати застосувати метод виділення і аналізу контурів [16]. Для цього виділяють границі об’єкта (місця різкої зміни градієнту яскравості). В основі більшості методів виділення контурів лежить обчислення похідних. Кожну точку зображення потрібно замінити її градієнтом з абсолютним значенням.

Основні функції обчислення градієнта для визначення контурів зображені на рисунку 1.15.

$$\nabla_1 f(x,y) = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2}$$

$$\nabla_2 f(x,y) = \left|\frac{\partial f(x,y)}{\partial x}\right| + \left|\frac{\partial f(x,y)}{\partial y}\right|;$$

$$\nabla_3 f(x,y) = \max\left\{\left|\frac{\partial f(x,y)}{\partial x}\right|, \left|\frac{\partial f(x,y)}{\partial y}\right|\right\}.$$

Рисунок 1.15 - Основні функції обчислення градієнта

Один з підходів підкреслення перепадів використовує наближене обчислення похідних у кожній точці зображення.

Формули наближеного обчислення похідних зображені на рисунку 1.16.

$$G_x = \frac{\partial f(x,y)}{\partial x} = \frac{\partial f}{\partial x} = f(x,y) - f(x-1,y)$$

$$G_y = \frac{\partial f(x,y)}{\partial y} = \frac{\partial f}{\partial y} = f(x,y) - f(x,y-1)$$

Рисунок 1.16 – Формули наближеного обчислення похідних

Також для цієї задачі на практиці часто застосовують маски операторів Собеля чи Превіта. Відповідні маски зображені на малюнках 1.17 та 1.18.

-1	-2	-1
0	0	0
1	2	1

G_x

-1	0	1
-2	0	2
11	0	1

G_y

Рисунок 1.17 – Маска оператора Собеля 3x3

-1	0	1
-1	0	1
-1	0	1

∇y

1	1	1
0	0	0
-1	-1	1

∇x

Рисунок 1.18 – Маска оператора Превіта

Ще одним способом виділення перепадів є використання операторів Кіршема, Уолліса чи Лапласа.

Наступним кроком після виділення контурів є перевірка знайдених ліній-границь на відповідність геометричним контурам об’єкта. Це можна зробити за допомогою методу Хафа [16].

Приклад роботи даного алгоритму зображений на рисунку 1.19.

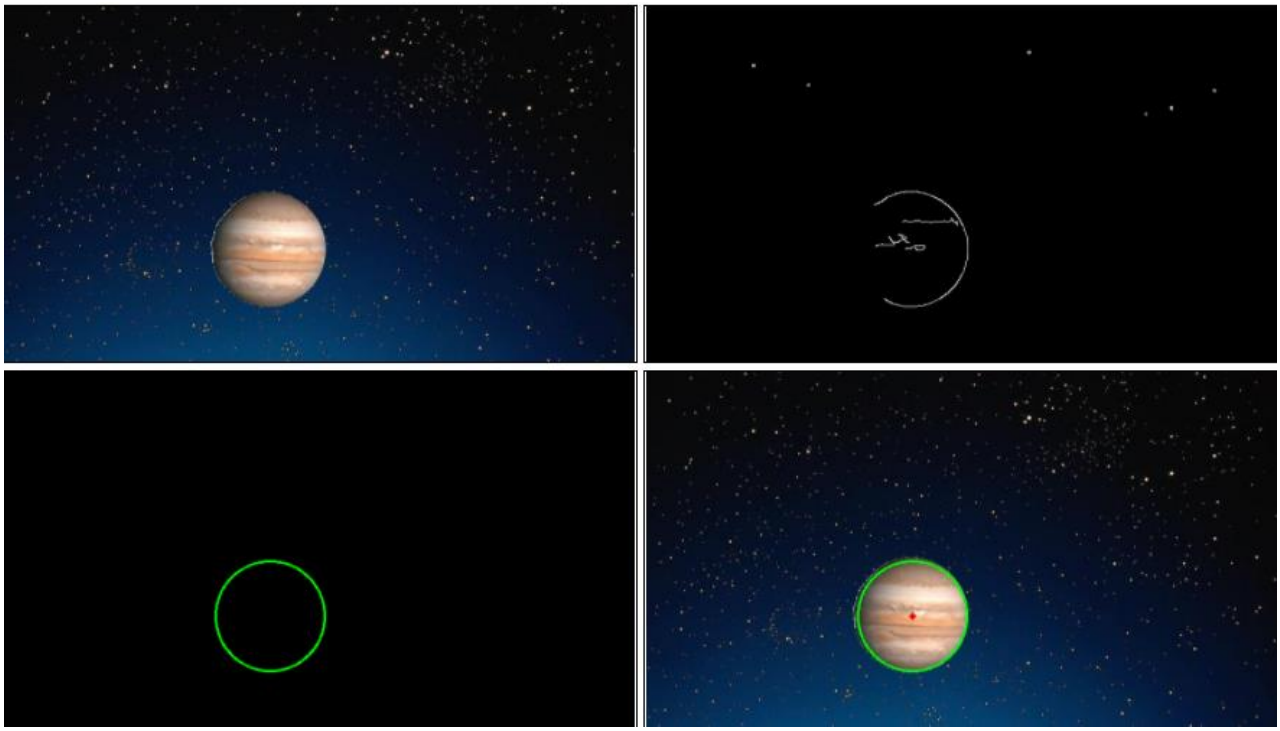


Рисунок 1.19 – Приклад роботи методу виділення та аналізу контурів

1.3.3 Метод співставлення з шаблоном

У випадку, якщо зображення має багато дрібних деталей, або якщо з будь-яких інших причин аналіз контурів не дає очікуваного реультату, можна застосувати метод співставлення з шаблоном (інша назва методу – метод кореляційного порівняння зображень). Для цього беруть зображення з об’єктом та шукають на великому зображенні області, які співпадають з зображенням об’єкту. Виконання цієї процедури відбувається за допомогою обрахунку коефіцієнту кореляції для кожної точки великого зображення.

Формула для обрахунку коефіцієнту кореляції зображена на рисунку 1.20.

$$\gamma(x,y) = \frac{\sum_s \sum_t (f(s,t) - \bar{f}(s,t))(w(x+s,y+t) - \bar{w})}{\sqrt{\sum_s \sum_t (f(s,t) - \bar{f}(s,t))^2 \sum_s \sum_t (w(x+s,y+t) - \bar{w})^2}}$$

Рисунок 1.20 – Формула для обрахунку коефіцієнту кореляції

Результат використання даного методу зображений на рисунку 1.21.

Рисунок 1.21 – Результат виконання методу співставлення з шаблоном

Детально цей метод описаний у роботі [17].

1.3.4 Метод особливих точок

Основою методу співставлення з шаблоном є пошук співпадінь точок шаблону з точками зображення. Якщо зображення повернуте, або ж його масштаб змінено відносно параметрів шаблону, то цей метод не дасть гарних результатів. У такому випадку краще застосувати метод особливих точок [16].

Особлива точка – це невелика область, яка суттєвим чином виділяється на зображенні. Існує багато методів визначення таких точок. У якості особливих точок можна розуміти кути на зображенні, або ж так звані блоби, тобто невеликі

області однакової яскравості, які мають достатньо чітку границю та виділяються на загальному фоні. Для особливої точки обраховують дескриптор – характеристику особливої точки. Його обраховують у заданому околі особливої точки, як напрямок градієнтів яскравості різних частин цього околу. Серед методів виділення дескрипторів виділяють такі, як SIFT, SURF, ORB і. т. ін. Частина цих методів є запатентованими і не доступні для комерційного використання. Маючи особливі точки можна розв'язати задачу пошуку об'єктів в зображенні. Алгоритм методу особливих точок наступний:

1. На зображенні з об'єктом шукають особливі точки об'єкту та обраховують їх дескриптори.

2. На тому зображенні, де шукають відповідні об'єкти, також шукають особливі точки та обраховують їх дескриптори.

3. Відбувається процедура порівняння дескрипторів особливих точок об'єкту та дескрипторів особливих точок, що знайдені на великому зображенні.

4. Якщо кількість співпадінь достатньо велика, то помічається область, яка містить відповідні точки.

На рисунку 1.22 зображений результат виконання даного алгоритму.

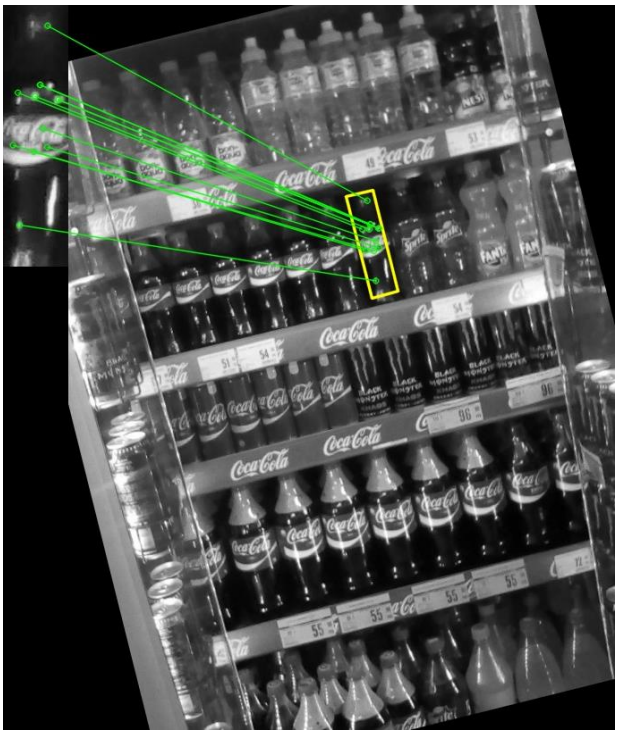


Рисунок 1.22 – Результат роботи методу особливих точок

1.3.5 Метод машинного навчання

Попередньо розглянуті алгоритми є прикладами методів, що були поширені та використовувалися у так званий традиційний період розвитку дисципліни пошуку об'єктів в зображенні. Ці методи є досить специфічними, вони опираються на певні характеристики об'єктів та зображень, на притаманні їм особливості. Тому задача пошуку для кожного об'єкту та зображення вимагала особливого підходу, попереднього аналізу та , можливо, попередньої цифрової обробки зображень. Усі ці завдання вимагали залучення людини на перших етапах розв'язку задачі. Крім цього дані моделі описують один конкретний об’єкт або дуже вузький клас об’єктів. Саме тому ці методи не можна узагальнити для розв'язку широкого кола задач пошуку об'єктів в зображенні.

Відповідно, була необхідна розробка методів та алгоритмів, які могли б без залучення людини проаналізувати зображення, виділити ознаки та за ними здійснити процедуру пошуку об'єктів. До того ж до нових методів ставилася вимога виявляти не лише один об'єкт, а й великі і різноманітні класи об'єктів.

Усі ці завдання, що постали перед дисципліною, сприяли переходу до нового етапу її розвитку, появи методів машинного навчання. Найбільш потужним та ефективним серед них є метод нерйонних мереж.

2 МЕТОД НЕЙРОННИХ МЕРЕЖ ДЛЯ РОЗВ‘ЯЗАННЯ ЗАДАЧІ ПОШУКУ ОБ‘ЄКТА В ЗОБРАЖЕННІ

2.1 Машинне навчання

Поняття «машинне навчання» найкраще охарактеризував Артур Семюель – піонер у галузі комп’ютерних ігор та штучного інтелекту і, власне, засновник даного терміну. За його словами машинне навчання – це область досліджень, яка надає комп’ютерам можливість навчатися не будучи при цьому безпосередньо запрограмованими [18].

На рисунку 2.1 зображене порівняння машинного навчання з класичним програмуванням.

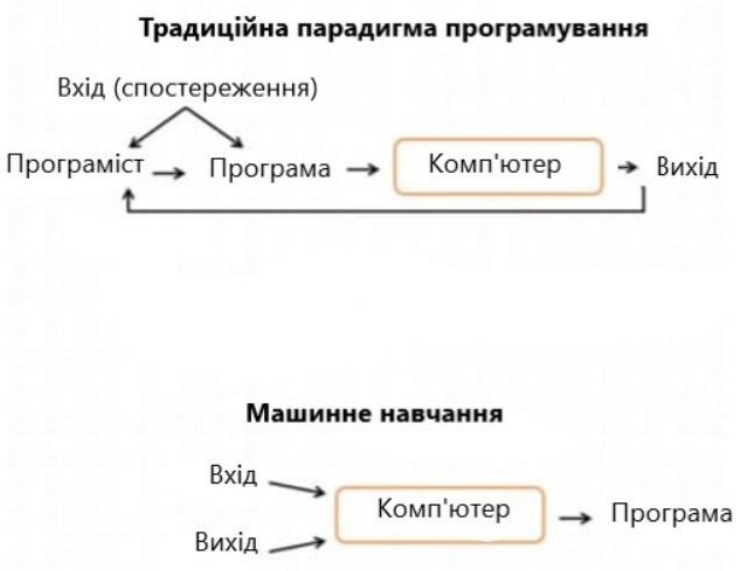


Рисунок 2.1 – Порівняння машинного навчання з класичним програмуванням

Машинне навчання – це сукупність методів штучного інтелекту, особливістю яких є не прямий розв’язок задачі, а навчання за рахунок застосування розв’язків множини схожих задач [19]. Алгоритми машинного навчання будують модель, засновану на вибіркових даних (тренувальних даних) для того щоб зробити припущення або прийняти рішення, не будучи при цьому безпосередньо запрограмованими. Машинне навчання застосовується у різних сферах, для вирішення різних задач, таких як фільтрація електронної пошти,

комп’ютерний зір, для яких важко або неможливо розробити традиційні алгоритми.

Підходи машинного навчання традиційно поділяються на три великі категорії: навчання з учителем, навчання без вчителя та навчання з підкріпленням [20].

Категорії машинного навчання зображені на рисунку 2.2.

Рисунок 2.2 – Категорії машинного навчання

2.1.1 Навчання з учителем

Алгоритми навчання з учителем – це клас алгоритмів машинного навчання, які використовують попередньо помічені дані для того, щоб вивчити їх ознаки. Відповідно вони можуть класифікувати подібні непомічені дані. Іншими словами, маючи вхідні дані у вигляді вектору спостережень «x», а також вектор «y» з мітками, який поставлений у відповідність вектору спостережень, алгоритм навчання з учителем намагається навчитися спрогнозувати вектор міток маючи вектор спостережень, зазвичай оцінюючи ймовірність p(y | x). Вектор міток і є тим «учителем», що вказує алгоритму що робити [21].

Одним з прикладів алгоритмів навчання з учителем є алгоритми лінійної та логістичної регресії. Лінійна регресія – це модель залежності однієї величини від іншої або декількох інших з лінійною функцією залежності [22]. Регресійний

аналіз намагається знайти значення параметрів функції залежності у такий спосіб, щоб ця функція якнайкраще описувала вхідні дані.

Отже, у алгоритмі лінійної регресії метою є мінімізація функції втрат шляхом знаходження відповідних параметрів функції, що залежить від вхідних даних і якнайкраще апроксимують значення міток. Функція втрат є мірилом того, як далеко апроксимовані значення знаходяться від реальних значень. Найбільш популярною і часто вживаною функцією втрат є середньоквадратична функція похибки (MSE). Вона дорівнює квадратові різниці прогнозованого і реального значення. Сума усіх таких функцій для усіх точок вхідних даних формує похибку алгоритму і є функцією втрат [23].

На початку алгоритму обраховується функція втрат. Далі використовується алгоритм градієнтного спуску для оновлення значень параметрів. Для цього знаходять частинні похідні від функції втрат по кожному параметру. Після оновлення параметрів процедура починається спочатку і триває доти, доки параметри не будуть знайдені з наперед заданою точністю.

У випадку, якщо мітками вхідних даних є числа від 0 до 1 і відповідно, прогонозовані значення також лежать у цьому діапазоні, то тут має місце логістична регресія. Логістична регресія є дуже подібною до лінійної регресії.
У лінійній регресії вихід дорівнює скалярному добутку вектора вхідних даних на вектор параметрів тієї ж довжини. А у логістичній регресії виходом є спеціальна логістична фунція від скалярного добутку вектора даних на вектор параметрів. Ця функція переводить значення скалярного добутку у значення з інтервалу (0:1). Процедура підбору оптимальних параметрів така сама як і для лінійної регресії [23].

Логістична регресія не є алгоритмом класифікації, але може бути перетоврена у нього. Для цього необхідно лише задати правило, яке визначає клас в залежності від виходу логістичної функції.

Приклад лінійної регресії показаний на рисунку 2.3.

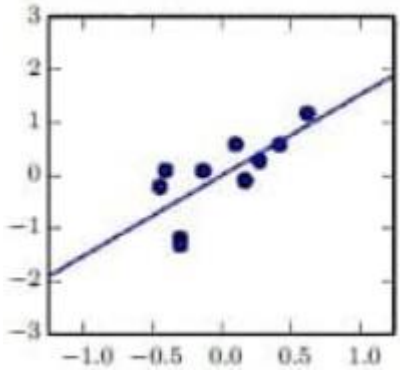


Рисунок 2.3 – Приклад лінійної регресії

Іншим прикладом алгортиму навчання з вчителем є метод опорних векторів. Здебільшого він використовується для задач класифікації. Задача класифікації може бути розглянута як процес знаходження гіперплощини, яка відділяє різні групи елементів в наборі даних. Так як кожен елемент даних має певні ознаки, то він може бути трактований як точка в багатовимірному просторі ознак. І задача алгоритму машинного навчання побудувати гіперплощину, яка б відділяла точки різних класів [23].

Отже, метод опорних векторів намагається побудувати гіперплощину, яка б відділяла групи елементів різних класів у наборі даних. У якості прикладу можна навести набір даних, який містить два класи точок у двовимірному просторі ознак. Якщо ці класи можна розділити гіперплощиною (у нашому випадку прямою), тоді ці класи лінійно-віддільні. Якщо ж не можна розділити - лінійно невіддільні [23].

Приклад лінійно-віддільних та лінійно-невіддільних класів точок зображений на рисунку 2.4

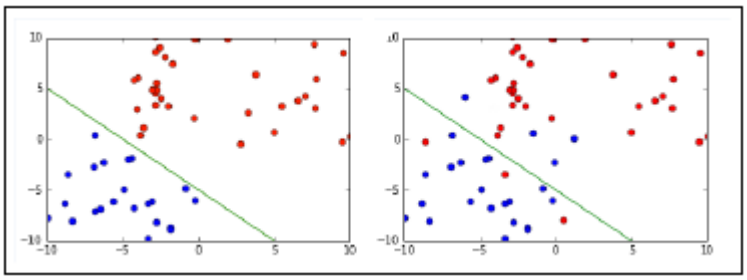


Рисунок 2.4 – Приклад лінійно-віддільних та лінійно-невіддільних класів точок

Метод опорних векторів намагається знайти таку гіперплощину, яка б максимізувала відстань від неї до кожної з точок. Також метод опорних векторів здатен працювати з даними, які не є лінійно-віддільними. У цьому випадку будують нелінійний класифікатор, в основі якого лежить перехід від скалярного добутку до випадкових ядер (kernel trick). Результуючий алгоритм є подібним до алгоритму лінійної класифікації [23]. Відмінність полягає лиш у тому, що скалярні добутки замінюються нелінійними функціями ядра (скалярним добутком у просторі з більшою розмірністю). У цьому просторі вже може існувати оптимальна гіперплощина. Так як розмірність результуючого простору може бути більшою за розмірність початкового простору, то функція, яка відповідає оптимальній гіперплощині у початковому просторі буде нелінійною.

Серед найбільш розповсюджених ядер виділяють такі: поліноміальне одноріде, поліноміальне неоднорідне, радіальна базисна функція, радіальна базисна функція Гауса, сігмоїд [24].

Приклад лінійно-невіддільних данних, які після перетворення стали лінійно-віддільними зображений на рисунку 2.5.

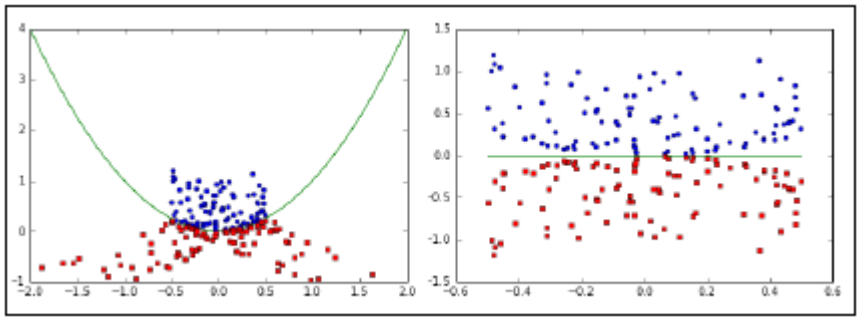


Рисунок 2.5 Приклад перевторення лінійно-невіддільних даних у лінійно-віддільні

Одним з простих та одночасно популярних алгоритмів класифікації, що є прикладом алгоритму навчання з учителем, є наївний баєсів класифікатор. Він грунтується на теоремі Баєса та жорстких припущеннях про незалежність різних подій [25]. Більшість алгоритмів машинного навчання намагаються оцінити ймовірність певної події Y , маючи умови X, тобто ймовірність $P(Y \mid X)$.Іноді,

відомо що відбулася подія Y . Також відома ймовірність виконання умов X. Для того, щоб знайти ймовірність $P(X \mid Y)$ використовують теорему Баєса [26]. Вона має вигляд $P(X \mid Y) = \frac{P(Y \mid X) \cdot P(X)}{P(Y)}$.

У багатьох випадках для оцінки параметрів для наївних баєсових моделей використовують метод максимальної правдоподібності.

Перевагою наївного баєсового класифікатора є мала кількість даних, які необхідні для навчання, оцінки параметрів та класифікації.

2.1.2 Навчання без учителя

Алгоритми навчання без учителя – це клас алгоритмів машинного навчання, у якому не використовуються попередньо помічені дані. Замість цього алгоритм навчання без учителя самостійно робить висновки, вивчаючи корисні властивості структури даних. Одним з найчастіше використовуваних та найпростіших прикладів задач навчання без вчителя є задача кластеризації. Вона дозволяє розділити дані на підмножини на основі схожості їх ознак. Для вимірювання схожості ознак використовують різні метрики. Також для більш складних алгоритмів можлива відсутність наперед заданої інформації про кількість класів [23].

На рисунку 2.6 показананий приклад розбиття точок на три класи.

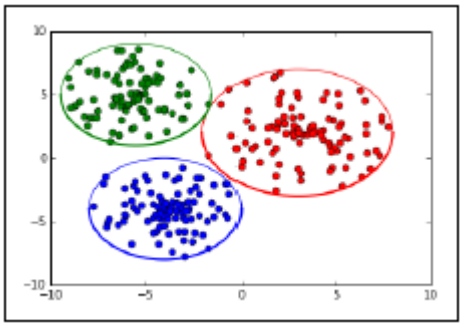


Рисунок 2.6 – Розбиття точок на три класи

Для розв’язку задачі розпізнавання образів експеримент навчання без учителя можна сформулювати як задачу кластерного аналізу. Вибірка об’єктів

розбивається на сепарабельні множини, які називають кластерами, у такий спосіб, щоб кожен кластер складався зі схожих об’єктів, а об’єкти різних кластерів істотно відрізнялися. Початкова інформація представлена у вигляді матриці відстаней.

Одним з найяскравіших представників алгоритмів кластеризації є алгоритм k-середніх. Цей алгоритм групує елементи набору даних у k різних кластерів. У алгоритмі послідовно виконуються такі дії [23]:

- Вибирається k випадкових точок (їх ще називають центроїдами) з простору ознак. Ці точки будуть відповідними центрами кожного з k кластерів.

- Для кожної точки з набору даних визначають її належність певному кластеру, до центроїду якого вона розміщена найближче.

- Для кожного кластеру відбувається перерахунок нових центроїдів шляхом обрахунку середнього арифметичного значень усіх точок кластеру.

- З новими центроїдами повторюють 2 та 3 кроки доки не виконається критерій зупинки.

Цей метод є чутливим до початкового вибору випадкових центроїдів. Тому варто повторити процедуру методу для різних початкових центроїдів [27].

Приклад роботи алгоритму для двовимірного випадку зображений на рисунках 2.7, 2.8, 2.9, 2.10.

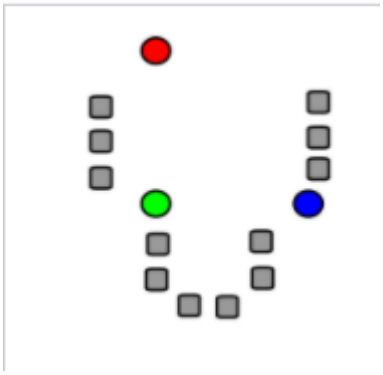


Рисунок 2.7 –Вихідні точки та випадковим чином вибрані початкові точки

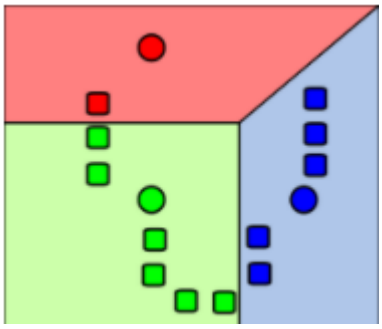


Рисунок 2.8 – Точки віднесені до початкових центрів, розбиття на площині відносно початкових центрів

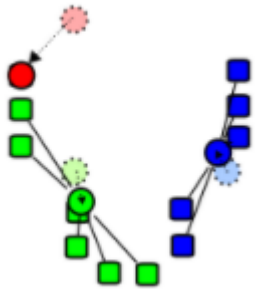


Рисунок 2.9 – Обрахунок нових центрів кластерів

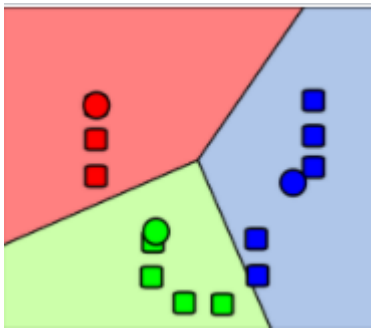


Рисунок 2.10 – Результат роботи алгоритму

2.1.3 Навчання з підкріпленням

Навчання з підкріпленням – це одина з категорій машинного навчання, яка грунтується на тому, що випробувальна система (агент) навчається в процесі взаємодії з деяким середовищем. Зворотнім зв’язком середовища на прийняті рішення є сигнали підкріплення. Тому таке навчання є частинним випадком навчання з учителем, але у цьому випадку вчителем є середовище або її модель. Також деякі правила підкріплення базуються на неявних учителях, через що її можна віднести до навчання без учителя [28].

У навчанні з підкріпленням агенту не відомо яку дію слід виконати , але за кожну дію призначається певна винагорода. Тобто замість того щоб працювати з міткми даних, алгоритму доводиться побудувати таку поведінку (послідовність дій), яка б максимізувала винагороду [18].

Алгоритм навчання з підкріпленням зображений на рисунку 2.11.

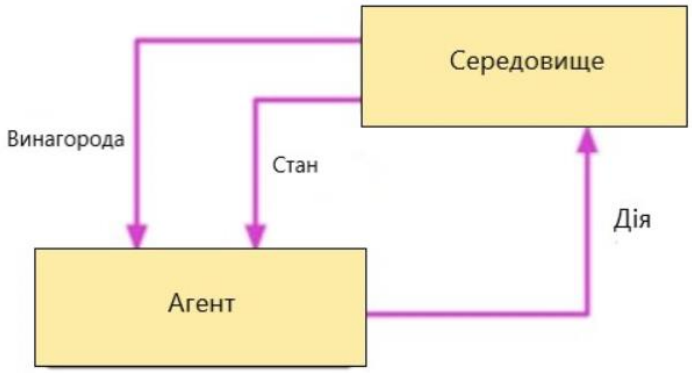


Рисунок 2.11 – Навчання з підкріпленням

Одним з прикладів методів, що відносяться до експериментів виду навчання з підкріпленням є Q-навчання. На основі отриманої від середовища винагороди агент формує функцію корисності Q, яка в подальшому впливає на стратегію поведінки агента шляхом врахування попереднього досвіду взаємодії з середовищем. Перевагою Q-навчання є те, що воно дозволяє порівняти очікувану корисність доступних варіантів дій, при цьому не формуючи моделі оточуючого середовища. Цей метод застосовують у ситуаціях, які можна подати у вигляді марківського процесу прийняття рішень [29].

Приклад процесу Q- навчання зображений на рисунку 2.12.

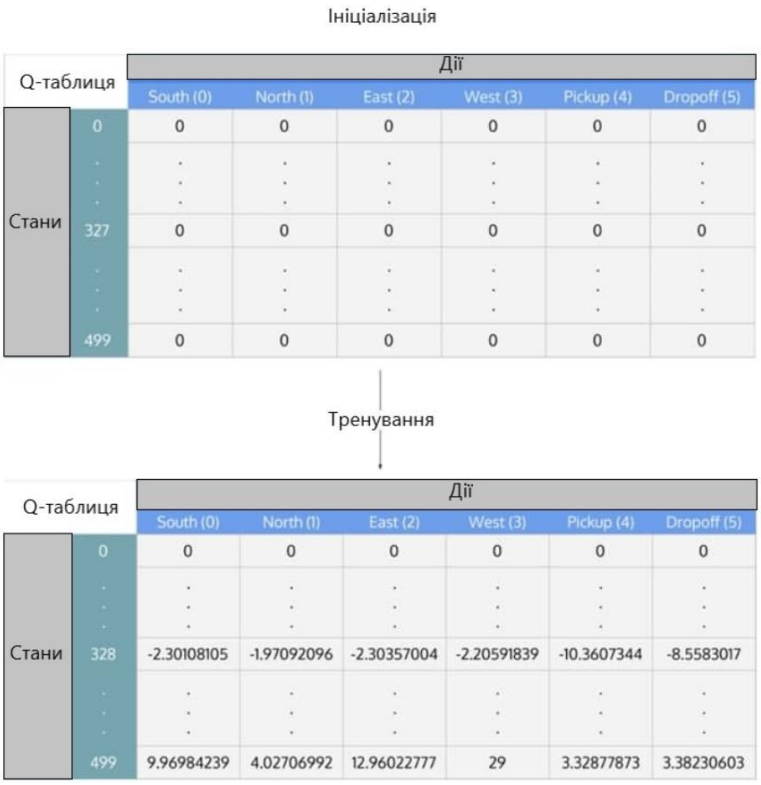


Рисунок 2.12 – Приклад процесу Q-навчання

2.1.4 Складники системи для вирішення задачі машинного навчання

Для вирішення задачі машинного навчання необхідна система, складовою частиною якої є власне алгоритм машинного навчання. Найбільш важливими частинами такої системи є [23]:

- Алгоритм навчання (його вбирають залежно від задачі, яку необхідно вирішити).

- Тренувальні дані (можуть бути як помічені, так і не помічені). Важливо мати достатню кількість даних для того , аби алгоритм зміг виявити їхню структуру.

- Представлення. Важливе правильне представлення даних у термінах їх ознак, які будуть подані на вхід алгоритму навчання. Це також має вплив на точність отриманих результатів.

- Ціль. Ціль визначає результат алгоритму навчання, що і як має використати алгоритм, а також яке представлення використати.

- Об’єкт. Об’єкт представляє те, що вивчається алгоритмом, а також кінцевий результат.

Задача, для розв’язання якої використовується класичний алгоритм машинного навчання може вимагати глибокого розуміння та попередньої обробки тренувальних даних. Для того, щоб вирішити задачу машинного навчання, необхідно виконати наступні кроки [23]:

- Збір даних. У випадку навчання з учителем крім збору даних необхідно їх також коректно помітити.

- Обробка даних. Необхідно очистити дані (наприклда, вилучити непотрібні ознаки) та зрозуміти, які ознаки визначають тренувальні дані.

- Розподіл даних на три категорії. Необхідно розділити дані на три категорії: тренувальні дані, дані перевірки та тестові дані. Тренувальні дані використовують для тренування алгоритму машинного навчання. Дані перевірки використовують для визначення точності алгоритму. Якщо точність недостатня, то можна змінити гіперпараметри алгоритму та почати процес навчання спочатку. Дані перевірки також допомагають визначити момент, коли варто завершити навчання. Тестові дані використовують для фінальної оцінки точності алгоритму, коли завершені цикли тренування та перевірки. Характерною особливістю тестових даних, так само як і даних перевірки, є те, що алгоритм не працює з ними в процесі тренування.

Існує багато причин створення тестових даних та даних перевірки. Алгоритми машинного навчання здатні лише побудувати апроксимацію очікуваного результату. Якщо використовується лише один набір даних, то модель може просто запам’ятати їх та у результаті мати дуже гарні значення точності на цих даних. Але на інших схожих наборах даних алгоритм може не досягнути бажаних результатів. Ключовою властивістю алгоритмів машинного навчання є їх здатність до узагальнення. Саме тому створюються дані перевірки та тестові дані, які надають можливість налаштувати параметри моделі під час тренування [23].

Під час тренування алгоритму машинного навчання обраховують похибку тренування. У процесі виконання алгоритму ця похибка має зменшуватись. Також обраховують тестову похибку, яка визначається як очікуване значення похибки на нових невідомих даних. Ця похибка також має зменшуватись у процесі виконанні алгоритму. Факторами, які визначають наскільки добре працює алгоритм машинного навчання є здатність алгоритму до зменшення похибки тренування та зменшення різниці між похибкою тренування та тестовою похибкою. Ці два фактори відносяться до понять недонавчання та перенавчання. Недонавчання відбувається тоді, коли алгоритм не здатен отримати малу похибку на тренувальних даних. Простими словами недонавчання відбувається тоді, коли алгоритм не здатен виявити структуру даних, пов'язати елементи даних певною залежністю. Перенавчання відбуваєтсья тоді, коли різниця між похибкою тренування та тестовою похибкою є занадто великою. Простими словами, перенавчання відбувається тоді, коли алгоритм трактує шуми як щось важливе, що визначає структуру даних. Ці процеси можна контрольювати шляхом зміни потужності моделі. Неформально, потужність моделі – це її здатність відповідати широкому колу функцій. Алгоритми машинного навчання дають гарний результат у тому випадку, коли потужність алгоритму відповідає сладності поставленої задачі та кількості вхідних даних. Моделі з недостатньою потужністю не здатні розв'язати складні задачі. Моделі з великою ємністю можуть розв'язати складні задачі, але якщо їх потужність більша ніж потрібно, то можливе перенавчання [21].

Приклад недонавчання та перенавчання зображений на рисунку 2.13.

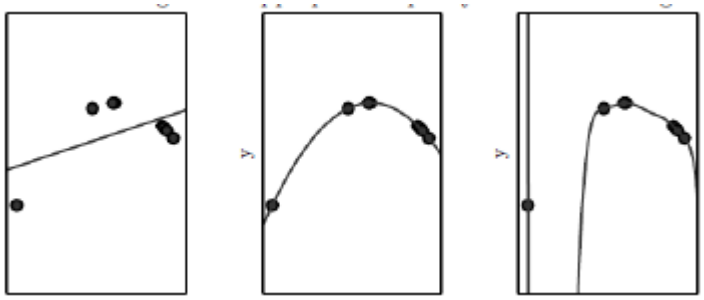


Рисунок 2.13 – Приклад перенавчання та недонавчання

2.2 Глибоке навчання

Глибоке навчання - це сукупність методів машинного навчання, які забезпечують новий підхід до пошуку представлення даних та опирається на вивченні послідовності рівнів (шарів) що раз більш важливих представлень. Ідея полягає у багатошаровому представленні моделі даних. Кількість рівнів, на які поділяється модель даних, називають глибиною моделі. Такі багаторівневі представлення вивчаютья у глибокому навчанні за допомогою моделей нейронних мереж, структурованих у вигляді шарів, накладених один на одний. Нейронна мережа поступово з кожним шаром перевторює вхідні дані в представлення, яке все більше відрізняється від початкового та яке містить все більше корисної інформації про результат. Прикладом поступової зміни предтавлення даних може бути модель для класифікації цифр на зображенні [30].

Відповідний приклад зображений на рисунку 2.14.

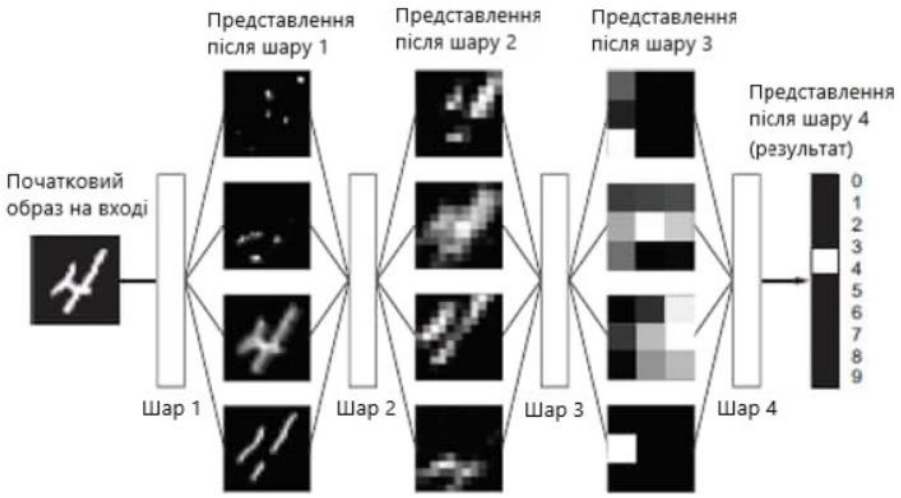


Рисунок 2.14 – Модель класифікації цифр на зображенні

Операції, які відбуваються на кожному наступному шарі з даними попереднього шару визначаються вагами, які фактично являються набором чисел. Завданням багаторівневої моделі є пошук такого набору значень вагів усіх шарів, при якому модель буде правильно відображати вхідні дані у відповідні їм результати. Для управління цим процесом та для коректування вагів необхідно вимірювати як далеко отриманий результат знаходиться від очікуваного. Для

цього використовують функцію втрат мережі (цільову функцію). Функція втрат залежить від двох аргументів – отриманого та очікуваного результатату. Вона обраховує оцінку відстані між цими аргументами. Саме ця оцінка використовується для коректування вагів з метою зменшення втрат. Для виконання процедури коректування використовують оптимізатор, який реалізує алгоритм зворотнього розповсюдження похибки. Отже, на кожному циклі навчання ваги коректуються, а оцінка втрат зменшується.

Глибоке навчання суттєво спрощує розв'язок задач, повністю автоматизуючи один з найважливіших кроків машинного навчання – конструювання ознак. Методика глибокого навчання має дві важливі характеристики – вона покроково, з кожним шаром конструює щоразу більш складні представлення даних та одночасно вивчає проміжні представлення, завдяки чому кожен шар оновлюється відповідно до необхідності представлення наступного шару та попереднього шару. У поєднанні ці дві властивості надають глибокому навчанню переваг у порівнянні з підходами машинного навчання.

Під час навчання модель глибокого навчання автоматично визначає ознаки досліджуваного об'єкта. Ці ознаки можуть бути неочевидними а також можуть не інтерпритуватися людиною. Те, що ознаки визначаються автоматично не лише спрощує задачу, але також робить отримані ознаки менш чутливими до шумів.

Алгоритми глибокого навчання суттєво відрізняються від алгоритмів неглибокго навчання за кількістю параметризованих перетворень, з якими стикається сигнал, який розповсюджується від вхідного шару до вихідного. Тут параметризованим перетворенням вважається той блок оробки даних, який містить параметри для навчання, такі як ваги [31].

На практиці усі алгоритми глибокого навчання є нейронними мережами, які мають певні спільні властивості. Усі вони складаються з нейронів які об'єднані в шари. Але кількість шарів, зв'язки між ними а також інші параметри можуть відрізнятися, породжуючи різні архітекури нейронних мереж. Найбільш розповсюдженими класами нейронних мереж є [23]:

- Багатошаровий перцептрон. Це нейронна мережа прямого розповсюдження з повнозв’язними шарами та хоча б одним прихованим шаром.

- Згорткова нейронна мережа. Це нейронна мережа прямого розповсюдження з декількома типами спеціальних шарів (наприклад, згоркові шари).

- Рекурентна нейронна мережа. Цей тип мережі має внутрішній стан, який базується на вхідних даних або їх частині, які вже передані до мережі.

2.3 Нейронна мережа

Нейронна мережа – це математична модель, а також її програмне та апаратне втілення, побудована на основі принципів організації та функціонування біологічних нейронних мереж – мереж нервових клітин живого організму [32]. Творці нейронних мереж намагалися створити модель, яка б емулювала роботу мозку. Але немає жодних підтверджень того, що процеси які протікають в мозку живого організму такі самі як і процеси всередині нейронної мережі. Будова нервової клітини живого організму показана на рисунку 2.15.

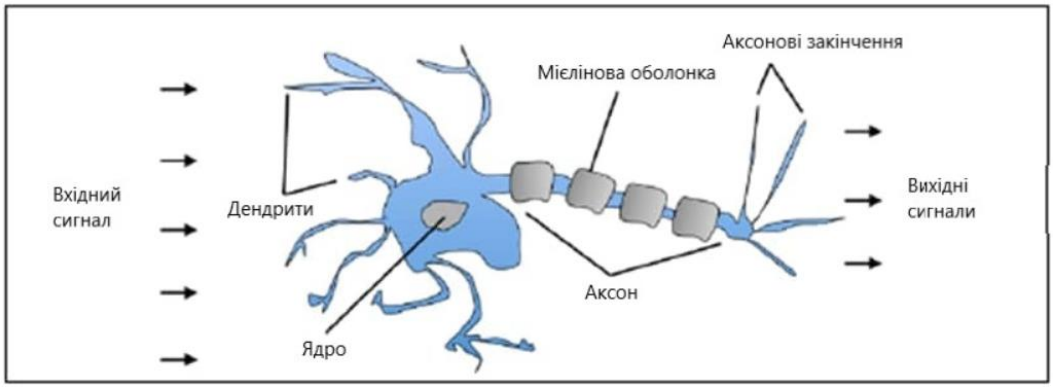


Рисунок 2.15 – Будова нервової клітини живого організму

Штучна нейронна мережа – це модель, яка оброблює інформацію та має наступні характеристики [23]:

- Обробка інформації відбувається в найпростішій формі над простими елементами-нейронами.

- Нейрони поєднані між собою та обмінюються сигналами через зв’язки.

- Зв’язки між нейронами можуть бути сильнішими та слабшими, що визначає обробку інформації.

- Кожен нейрон має початковий стан, який визначається усіма вхідними зв’язками від інших нейронів.

- Кожен нейрон має свою функцію активації, яка перетворює його стан та визначає вихідний сигнал.

Більш загальним представленням нейронної мережі є її представлення у вигляді обчислювального графа математичних операцій.

Головними характеристиками нейронної мережі є [23]:

- Архітектура. Архітектура визначає множину зв’язків між нейронами, кількість шарів та кількість нейронів у кожному шарі.

- Навчання. Навчання відбувається під час тренування нейронної мережі. Найбільш розповсюдженим шляхом тренування мережі є застосування градієнтного спуску та зворотнього розповсюдження похибки.

2.3.1 Перцептрон та штучний нейрон, їх будова

Штучний нейрон – це математична функція, яка приймає один або декілька значень і повертає єдине числове значення [23].

На рисунку 2.16 зображені складові частини штучного нейрона.

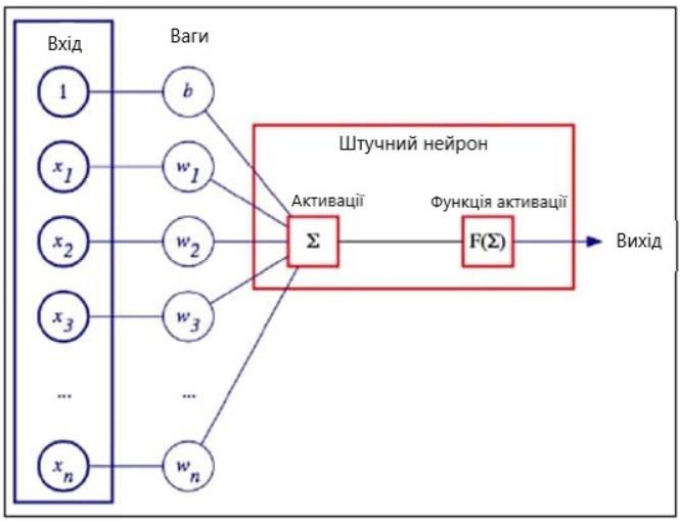


Рисунок 2.16 – Складові частини штучного нейрона

Штучний нейрон визначається як функція виду:

$$y = f(\sum_i x_i w_i + b)$$

Спочатку обраховується зважена сума вхідних даних x_i та вагів w_i. Тут x_i – це числові значення, які представленням вхідних даних, або вихідні значення інших нейронів. Ваги w_i – це числові значення , які відображають силу зв'язків між нейронами, b – це спеціальне значення, яке називають зміщенням і яке дорівнює одиниці [23].

Далі зважена сума подається на вхід функції активації, яку ще називають передатною функцією. Існує багато типів передатних функцій, але усі вони задовольняють вимогам нелінійності.

Відповідно до сучасної термінології, перцептрон може бути класифікований як шутчна нейронна мережа з одним прихованим шаром, пороговою передатною функцією та прямим поширенням сигналу.

У загальному вигляді перцептрон являє собоюсистему елементів трьох типів: сенсорів, асоціативних елементів та реагуючих елементів [33].

Загальна будова перцептрона зображена на рисунку 2.17.

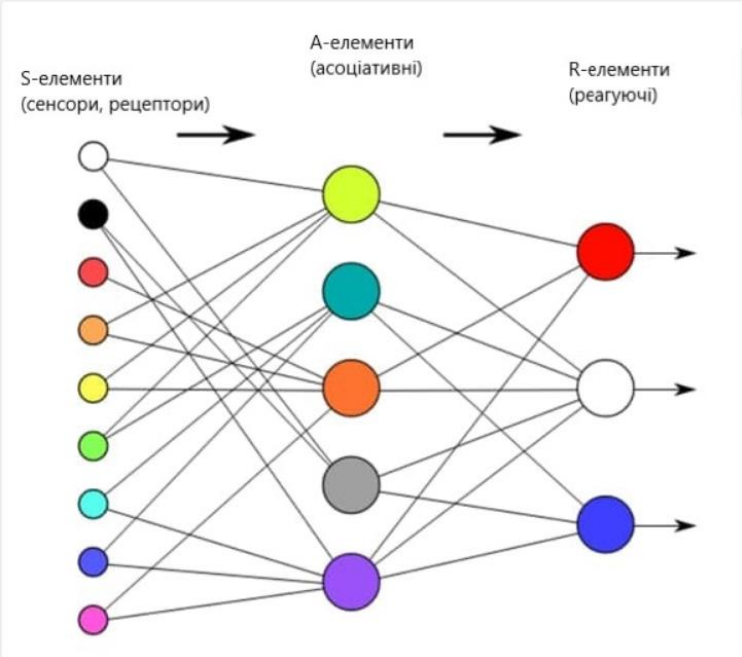


Рисунок 2.17 – Загальна будова перцептрона

У перцептроні S-елементи можуть знаходитися у стані спокою (сигнал дорівнює 0), або у стані збудження (сигнал дорівнює 1). Сигнали від S-елементів передаються до A-елементів по S-A зв’язках. Ці зв’язки можуть мати значення 1, 0 або -1. Якщо сигнали, які надійшли на A-елемент в сукупності перевищують певне порогове значення, тоді цей елемент видає одиничний сигнал. У іншому випадку генерується нульовий сигнал [33].

Далі сигнали від A-елементів надходить до R-елементів по A-R зв’язкам. Тут зв’язки можуть приймати будь-які значення. R-елемент підсумовує зважені сигнали і якщо отрмане значення перевищує певну величину, то генерується одиничний сигнал. У іншому випадку генерується сигнал -1. R-елемент визначає вихід перцептрона в цілому [33].

Одним з видів перцептронів є одношаровий перцептрон. Його особливість полягає у тому, що кожен S-елемент однозначно відповідає одному A-елементу, усі S-A зв’язки мають значення +1, а порогове значення A-елементів дорівнює 1.

Будова одношарового перцептрона зображена на рисунку 2.18.

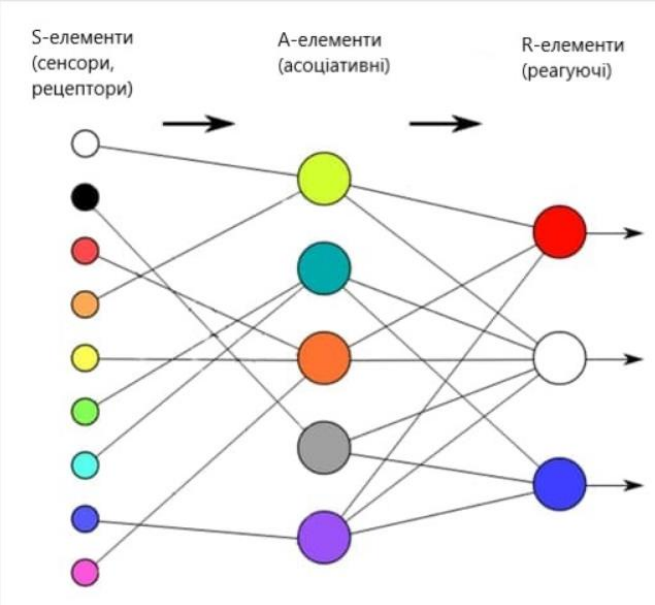


Рисунок 2.18 – Будова одношарового перцептрона

Таким чином, частину одношарового перцептрона можна подати у вигляді штучного нейрона. На відміну від штучного нейрона , у одношарового

перцептрона вхідні сигнали можуть набувати фіковані значення 0 або 1. У штучного нейрона на вхід можна подавати будь-які значення [33].

Одношаровий перцептрон у вигляді штучного нейрона зображений на рисунку 2.19.

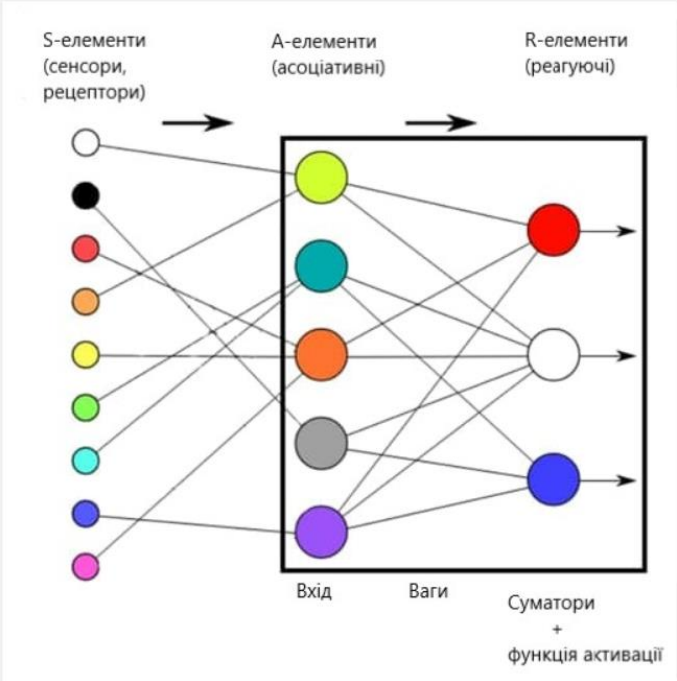


Рисунок 2.19 – Перцептрон у виглядіі штучного нейрона

Перцептрон визначає гіперплощну в просторі ознак. Характерною властивістю є те, що він працює з лінійно-віддільними класами. Для того, аби подолати ці обмеження, необхідно поєднати нейрони у нейронну мережу.

Нейронна мережа може мати необмежену кількість нейронів, які організовані у взаємозв’язані шари. Вхідний шар представляє набір даних. Вихідний шар може мати більше одного нейрона (наприклад, в задачах класифікаії кількість вихідних нейронів дорівнює кількості класів) [23].

На рисунку 2.20 зображена одношарова мережа прямого розповсюдження.

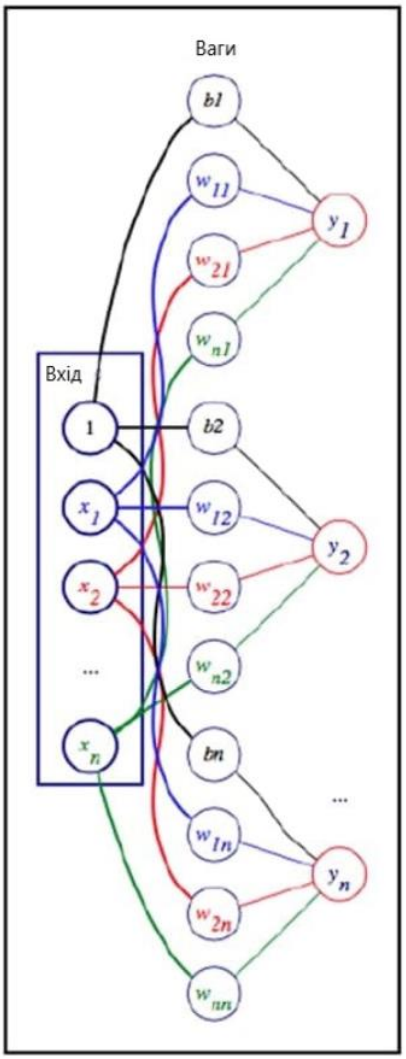


Рисунок 2.20 – Одношарова мережа

Тут ваги w позначають зв’язки між нейронами. Вага w_{ij} зв’язує і-тий попередній нейрон з j-тим наступним нейроном. Перше вхідне значення – це значення зсуву, а перша вага b1 – вага зсуву.

Нейрони одного шару можуть бути зв’язаними з нейронами інших шарів, але не з нейронами цього самого шару [23].

2.3.2 Багатошарова нейронна мережа

Недоліком одношарової нейронної мережі є те, що вона здатна класифікувати лише лінійно-віддільні класи. Для подолання цього обмеження створюють більше проміжних шарів між вхідним та вихідним шаром. Ці проміжні шари також називають прихованими шарами.

На рисунку 2.21 зображена нейронна мережа з двома прихованими шарами.

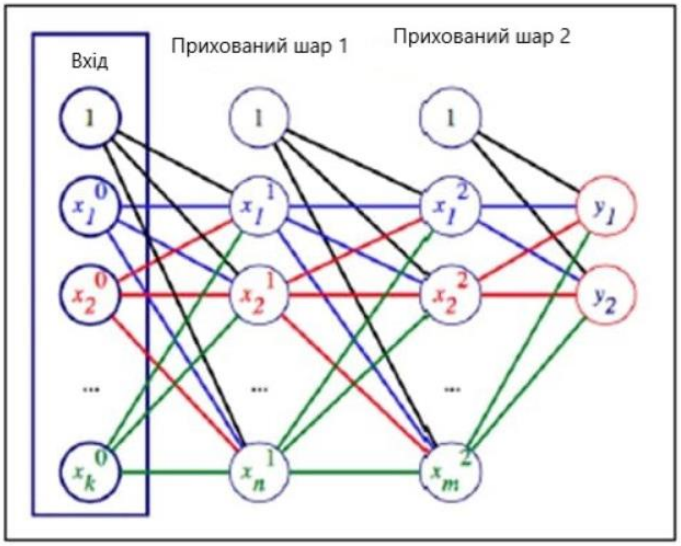


Рисунок 2.21 – Нейронна мережа з прихованими шарами

Тут вхідний шар складається з к нейронів, перший прихований шар містить n прихованих нейронів, другий прихований шар містить m нейронів. Вихідний шар складається з двух нейронів, тобто визначає два класи y1 і y2. Першим нейроном шару є нейрон зсуву. Кожен нейрон попереднього шару зв’язаний з усіма нейронами наступного шару (тому мережа повнозв’язна). Кожен зв’язок між нейронами має свою вагу w [23].

Узагалі, немає обмежень щодо кількості прихованих шарів.

Нейрони та зв’язки між ними формують направлений циклічний граф. У цьому графі інформація не може проходити двічі з одного й того самого нейрона (граф не має циклів) та її рух відбувається лише у одному напрямку, зі входу до виходу [23].

На рисунку 2.22 зображений приклад представлення мережі у вигляді графу.

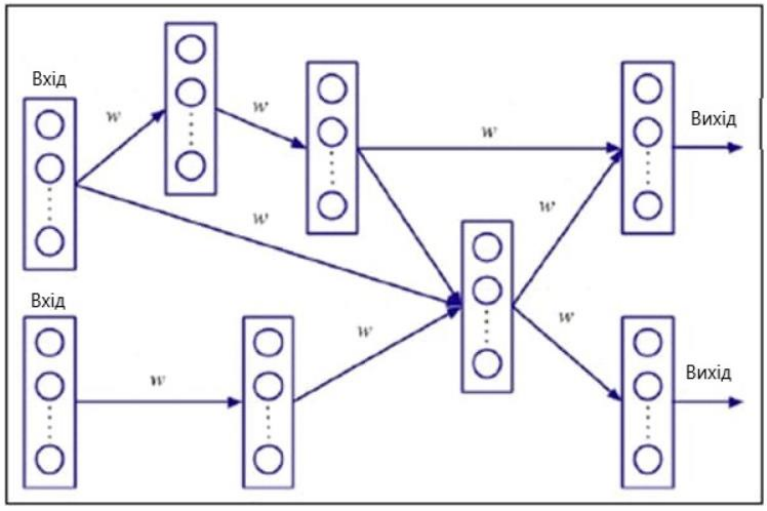


Рисунок 2.22 – Приклад представлення мережі у вигляді графу

На цьому рисунку представлена мережа з двома вхідними шарами, двома вихідними шарами та випадковим чином взаємопов’язаними проміжними шарами.

2.3.3 Типи функцій активації

Відомо, що багатошарові нейронні мережі здатні класифікувати лінійно-невіддільні класи. Але для цього необхідно щоб мережа була нелінійною функцією.

Якщо нейрони не мають функцій активації, то їх вихід буде являти собою зважену суму вхідних значень, що є лінійною функцією. А отже і вся мережа являє собою лінійну функцію як композиція лінійних функцій. Тому для того щоб перетворити мережу на нелінійну функцію застосовують нелінійні функції активаціїї для нейронів [23].

Зазвичай нейрони одного шару мають однакві функції активації, але різні шари можуть мати різні функції активації.

Таким чином, функції активації поділяють на два класи [34]:

- Лінійні функції.
- Нелінійні функції.

Найбільш поширеними функціями активації є такі [23]:

- Тотожня функція активації.

- Порогова функція активації. Ця фунція активує нейрон, якщо значення активації вище певного значення.

- Логістична функція активації (або логістична сігмоїда). Цю функцію активації використовують найчастіше, адже її вихід належить проміжку (0,1) і це значення може бути інтерпритоване як ймовірність ктивації нейрона.

- Біполярна сігмоїда. Ця функція є дуже схожою на логістичну сігмоїду та приймає значення на проміжку (-1,1).

- Гіперболічний тангенс.

- ReLU (Rectified Linear Unit). Ця функція поєднує властивості тотожньої та порогової функції. Існує багато різновидів даної функції, наприклад, Noisy ReLU, Leaky ReLU, ELU (Exponential Linear Unit).

На рисунку 2.23 зображені основні функції активації.

Однією з найважливіших характеристик функції активації є її похідна (градієнт). Похідна функції активації має важливе значення у процесі тренування мережі.

Похідні деяких функцій активації зображені на рисунку 2.24.

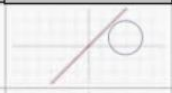
Назва	Графік	Рівняння	Похідна
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Рисунок 2.23 – Основні функції активації

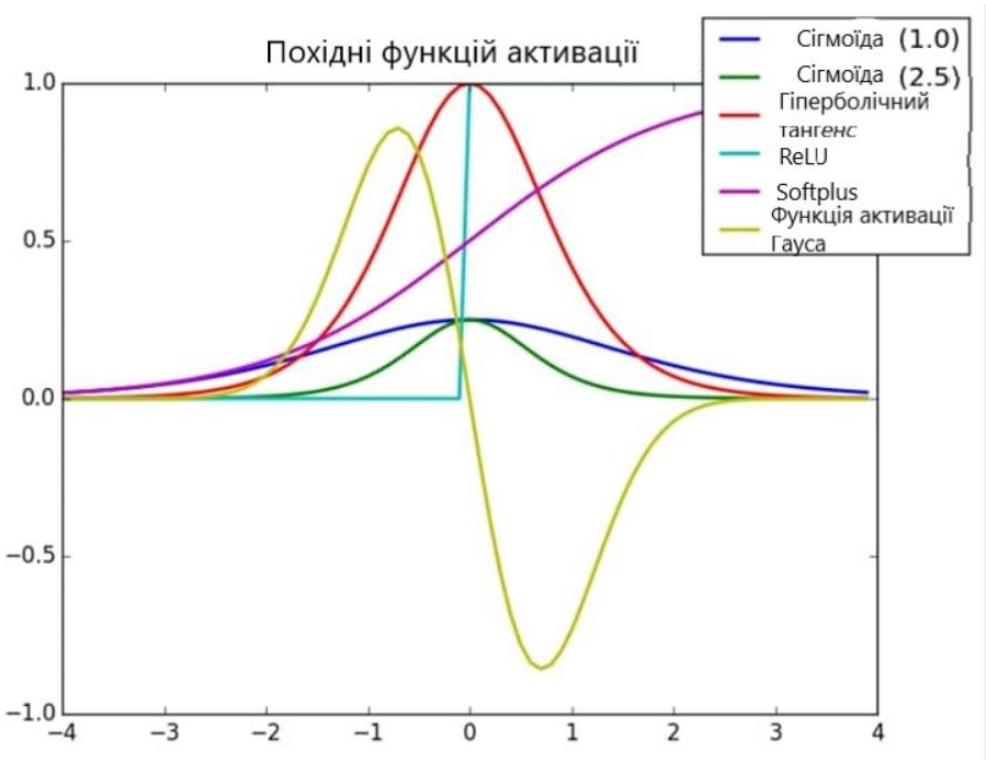


Рисунок 2.24 – Похідні деяких функцій активації

Під час тренування нейронної мережі відбувається процедура обчислення похідних функцій активації. Під час цього процесу може виникнути проблема зникаючих градієнтів.

Проблема зникаючих градієнтів полягає у наступному. Нехай у мережі використовується функція активації логістична сігмоїда. Похідна функції логістичної сігмоїди має суттєві значення в деякому околі нуля, а поза його межами значення збігаються до нуля. У такому випадку для нейронної мережі з багатьма шарами похідні будуть збігатися до нуля під час виконання процедури зворотнього розповсюдження похибки. Це о значає, що ваги мережі не будуть оновлюватися, а отже навчання не відбуватиметься [23].

Для подолання цієї пробеми застосовують функцію активації ReLU. Перевагою цієї функції є те, що її похідна постійна та не прямує до нуля при великих значеннях аргументу [23].

2.3.4 Тренування нейронної мережі

У процесі навчання мережа у довільному порядку продивляється навчальну вибірку. Порядок перегляду може бути послідовним, випадковим або іншим. Деякі мережі навчаються без учителя, продивляються вибірку лише один раз. Інші, а також мережі, що навчаються з учителем, продивляються вибірку багато разів, при цьому один повний прохід по вибірці називається епохою навчання.

Головною метою у процесі навчання є мінімізація похибки. Так як похибка – це функція вагів мережі, то необхідно мінімізувати похибку відносно вагів.

У випадку лінійної регресії мережа складається з одного нейрона з тотожньою функцією активації. Тоді процес навчання такої мережі має наступний вигляд [23]:

- Випадковим чином задають початкові значення вагів w.

- Складають функцію середньоквараточної похибки (функцію втрат) J для всіх елементів набору даних. Функція середньоквадратичної похибки

дорівнює середньому арифметичному значенню квадратів різниць отриманих значень та відомих значень.

$$J = \frac{1}{n}\sum_{i=0}^{n}(y^i - t^i)^2 = \frac{1}{n}\sum_{i=0}^{n}(x^i \square w - t^i)^2$$

- Оновлюють значення вагів w на основі похідних від функції втрат J за кожною змінною-вагою.

$$w \to w - \eta\nabla(J(w))$$

- Повторюють ці дії поки значення похибки не досягне певного значення.

Приклад зміни середньоквадратичної похибки у процесі навчання зображений на рисунку 2.25.

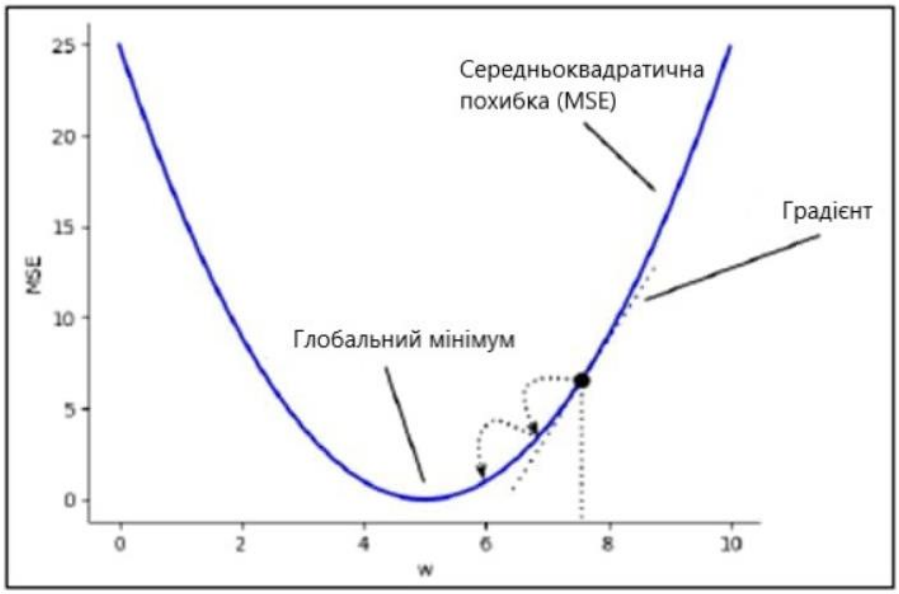


Рисунок 2.25 – Приклад зміни середньоквадратичної похибки

Задача мінімізації функції J полягає у тому, що потрібно знайти такі значення вагів w, при яких функція J досягне свого глобального мінімуму. Для цього необхідно обрахувати першу похідну (градієнт) функції J за змінною w.

У загальному випадку, коли є декілька вхідних значень та вагів, обраховують частинні похідні за кожною змінною w_j за наступною формулою [23]:

$$\vec{d} = \frac{\partial J(w)}{\partial w_j} = \frac{\partial \frac{1}{n}\sum_{i=0}^{n}(y^i - t^i)^2}{\partial w_j}$$

Для того, щоб рух відбувався в напрямку мінімуму, необхідно рухатися в протилежному напрямку до вектора $\vec{d}$ для кожного значення w_j.

Отже, маємо:

$$\frac{\partial J(w)}{\partial w_j} = \frac{\partial \frac{1}{n}\sum_{i=0}^{n}(y^i - t^i)^2}{\partial w_j} = \frac{1}{n}\sum_i \frac{\partial (y^i - t^i)^2}{\partial w_j} = \frac{2}{n}\sum_i \frac{\partial y^i}{\partial w_j}(y^i - t^i)$$

Якщо $y^i = x^i \square w$, тоді $\frac{\partial y^i}{\partial w_j} = x_j^i$, а також $\frac{\partial J(w)}{\partial w_j} = \frac{2}{n}\sum_i x_j^i (y^i - t^i)$.

Після того, як обраховані частинні похідні, онволюють значення вагів за наступною формулою:

$$w_j \to w_j - \eta \frac{\partial J(w)}{\partial w_j} = w_j - \eta \frac{2}{n}\sum_i x_j^i (y^i - t^i)$$

Тут параметр η є коефіцієнтом швидкості навчання.

Формула оновлення вагів у матричній формі має наступний вигляд:

$$w \to w - \eta \nabla (J(w)) = w - \eta \nabla (\frac{2}{n}\sum_i (y^i - t^i)^2)$$

Тут ∇ є оператором набла, який представляє вектор частинних похідних.

$$\nabla = (\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, ..., \frac{\partial}{\partial w_n})$$

Під час навчання нейронна мережа переглядає усі елементи набору даних, відповідно для кожного елементу обраховується похибка, яка вносить свою частку до загального значення похибки.

Часто трапляються задачі, для яких набори даних мають великі розміри. Процес оновлення вагів навіть для одного елементу даних вимагає певних

часових затрат. Відповідно процес оновлення вагів для великих наборів даних є надзвичайно повільним.

Для того, аби прискорити навчання, ваги оновлюють з певним кроком, лише після певної кількості проглянутих елементів даних. Цей тип оновлення вагів становить основу методу міні-пакетного градієнтного супску (mini-batch gragient descent) [23].

Також, однією з нетривіальних проблем оновлення вагів є проблема локальних мінімумів функції втрат.

У випадку логістичної регресії використовують функцію активації логістична сігмоїда, на відміну від лінійної регресії, де використовують тотожню функцію активації.

Нехай $\sigma(a)$ позначає логістичну сігмоїду, де a - значення активації нейрона x w. Для кожного елементу даних ймовірність того, що вихід належить класу за умови значень вагів має вигляд [23]:

$$P(t \mid x, w) = \begin{cases} \sigma(a), t = 0 \\ 1 - \sigma(a), t \neq 0 \end{cases}$$

Попереднє рівняння можна записати у вигляді:

$$P(t \mid x, w) = \sigma(a)^t (1 - \sigma(a))^{1-t}$$

Так як ймовірності $P(t^i \mid x^i, w)$ є незалежними для кожного елементу даних x^i , попереднє рівняння можна записати у наступному вигляді:

$$P(t \mid x, w) = \prod_i P(t^i \mid x^i, w) = \prod_i \sigma(a^i)^{t^i} (1 - \sigma(a^i))^{(1-t^i)}$$

Якщо взяти натурнальний логарифм від попереднього рівняння, то отримається наступна рівність:

$$\log(P(t \mid x, w)) = \log(\prod_i \sigma(a^i)^{t^i} (1 - \sigma(a^i))^{(1-t^i)}) =$$

$$\sum_i [t^i \log(\sigma(a^i)) + (1 - t^i) \log(1 - \sigma(a^i))]$$

Головним завданням є максимізація даного логарифму для того, щоб отримати найбільшу ймовірність передбачення правильного результату.

Для цього використовують алгоритм градієнтного спуску. За допомогою алгоритму градієнтного спуску мінімізують функцію втрат $J(w)$, яка задається наступним виразом:

$$J(w) = -\log(P(y \mid x, w))$$

Для мінімізації даної функції обраховують її частинні похідні за кожною змінною-вагою w_j :

$$\frac{\partial \log(P(t \mid x, w))}{\partial w_j} = \frac{\sum_i [t^i \log(\sigma(a^i)) + (1-t^i)\log(1-\sigma(a^i))]}{\partial w_j} =$$
$$\sum_i [t^i (1-\sigma(a^i)) x_j^i + (1-t^i)\sigma(a^i) x_j^i]$$

Наступним кроком оновлюють значення вагів у той самий спосіб, як і у випадку лінійної регресії.

Процедура оновлення вагів повторюється до того часу, доки функція втрат не досягне свого глобального мінімуму.

2.3.5 Метод зворотнього розповсюдження похибки

Розглянутий алгоритм оновлення похибки є справедливим лише для нейронних мереж з одним прихованим шаром. У випадку багатошарової нейронної мережі цю процедуру оновлення можна застосувати лише для вагів, які сполучають останній прихований шар з вихідним шаром [23].

Ідея оновлення вагів багатошарової нейронної мережі полягає у наступному. Обчислюють похибку на останньому прихованому шарі мережі, а потім оцінюють яке буде значення похибки на попередньому рівні. Тобто похибка буде поширюватися з останнього шару мережі до першого. Саме тому алгоритм, що здійснює цю процедуру має назву метод зворотнього поширення похибки [23].

Нехай J позначає функцію втрат, a - значення активації нейрона x w, y - вихід функції активації нейрона, $w_{i,j}$ - вага, що сполучає і-тий нейрон шару l з j-тим нейроном шару l+1 мережі.

Алгоритм починає свою роботу з останнього шару мережі. Обраховують частинні похідні функції втрат за змінними-вагами, які сполучають останній прихований шар з вихідним шаром [23].

$$\frac{\partial J}{\partial w_{i,j}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial w_{i,j}}$$

Так як $\frac{\partial a_j}{\partial w_{i,j}} = y_i$, то виконується наступна рівність:

$$\frac{\partial J}{\partial w_{i,j}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial a_j} y_i$$

Для попередніх шарів мережі справедлива та ж сама формула:

$$\frac{\partial J}{\partial w_{i,j}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial w_{i,j}}$$

Відомо, що $\frac{\partial a_j}{\partial w_{i,j}} = y_i$, а також $\frac{\partial y_j}{\partial a_j}$ є похідною функції активації (відома функція). Залишається знайти похідну $\frac{\partial J}{\partial y_j}$.

Отже, якщо є послідовність з трьох шарів $y_i \to y_j \to y_k$, то можна використати дві основні формули:

$$\frac{\partial J}{\partial w_{i,j}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial w_{i,j}}$$

$$\frac{\partial J}{\partial y_j} = \sum_k \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial y_j}$$

За допомгою цих формул знаходять похідні функції втрат на кожному шарі.

Якщо покласти $\delta_j = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial a_j}$, де являє δ_j собою похибкою нейрону y_j , то має місце наступна рівність:

$$\frac{\partial J}{\partial y_i} = \sum_j \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial y_i} = \sum_j \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial a_j}\frac{\partial a_j}{\partial y_i} = \sum_j \delta_j w_{i,j}$$

Похибку нейрону на попередньому шарі можна знайти за наступною формулою:

$$\delta_i = (\sum_j \delta_j w_{i,j})\frac{\partial y_i}{\partial a_i}$$

Тоді справедлива наступна формула:

$$\frac{\partial J}{\partial w_{i,j}} = \delta_j \frac{\partial a_j}{\partial w_{i,j}} = \delta_j y_i$$

Отже, правило оновлення вагів на кожному шарові має вигляд:

$$w_{i,j} \to w_{i,j} - \eta \delta_j y_i$$

2.4 Пошук об’єктів в зображенні за допомогою згорткової нейронної мережі

Використання нейронних мереж з повнозв'язними шарами для розв'язку задачі пошуку об'єктів в зображенні не дає гарних результатів. Причиною цього є те, що на вхід мережі подається забраження великих розмірів, як правило з трьома каналами (RGB), у результаті чого перший шар мережі містить сотні тисяч нейронів. Так як кожен нейрон попереднього шару пов’язаний з усіма нейронами наступного шару, кількість зв’язків у такій мережі є надзвичайно великою. Усе це призводить до перенавчання мережі та надмірного соживання пам’яті. Також мережа не має інформації стосовно просторовго розміщення пікселів зображення, що призводить до значного погіршення результату пошуку об’єктів [23].

Альтернативою є використання згорткових нейронних мереж, які мають такі особливості:

- З’єднуються нейрони, які відповідають лише сусіднім пікселям зображення. Таким чином, нейрони приймають на вхід дані лише від інших просторово близьких нейронів. Це призводить до зменшення кількості вагів, так як не усі нейрони взаємопов’язані між собою.

- Застосовується сумісне використання параметрів. Це призводить до ще більшого зменшення кількості вагів та допомагає подолати перенавчання.

Таким чином, згорткові нейронні мережі враховують той факт, що вхідні дані являють собою зображення. Тому і архітектура згорткової нейронної мережі влаштована відповідним чином.

До прикладу, якщо на вхід мережі подається зображення з трьома каналами (RGB), то, на відміну від звичайної мережі, нейрони згорткової мережі розміщені у трьох вимірах (ширина, висота, глибина). Також нейрони наступного шару зв’язані лише з частиною нейронів попереднього шару (частиною нейронів певної області попереднього шару) [35].

Порівняння архітектур звичайної та згорткової мереж зображене на рисунку 2.26.

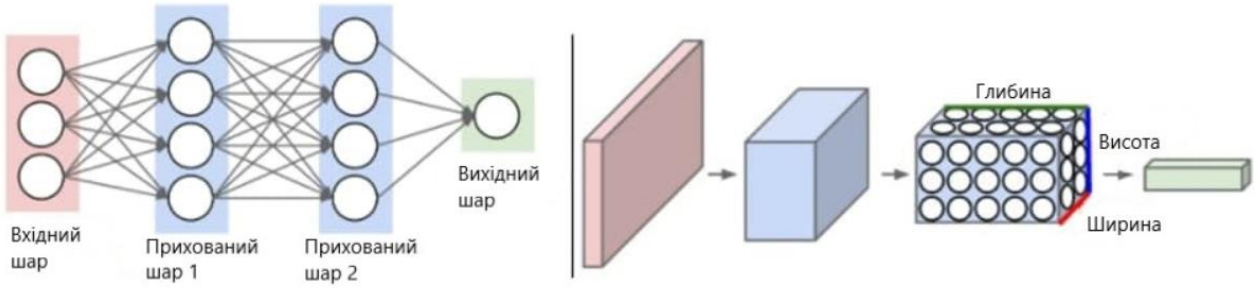


Рисунок 2.26 – Порівняння архітектур звичайної та згорткової мереж

Таким чином, найпростіша згорткова мережа – це послідовність шарів, кожен з яких перетворює один об’єкм активацій на інший за допомгою диференційованих функцій.

Робота згорткової мережі зазвичай інтерпритується як перехід від конкретних особливостей зображення до більш абстрактних деталей, а далі до

ще більш абстрактних деталей аж до виявлення понять високого рівня. При цьому мережа самоналаштовується та самостійно будує послідовність карт ознак, фільтруючи неважливі деталі та виділяючі суттєві.

Структура згорткової мережі – однонаправлена та принципово багатошарова [36].

У згорткових нейронних мережах використовують три головні типи шарів: згортковий шар (Convolutional Layer) , шар субдискретизації (Pooling Layer) та повнозв'язний шар (Fully Connected Layer). Також використовується шар активації (шар ReLU) [35].

Кожен шар може мати або не мати параметрів та додаткових гіперпараметрів.

У згортковій нейронній мережі у операції згортки застосовується лише обмежена матриця вагів невеликого розміру, яка «ковзає» по усьому шару, що оброблюється (на самому початку – по вхідному зображенню). У процесі «ковзання» після кожного зсуву матриці формується сигнал активації для нейрону наступного шару з аналогічною позицією. Таким чином, для різних нейронів вихідного шару застосовується одна й та ж сама матриця вагів, яку ще називають ядром згортки. Ядро згортки інтерпретується як графічне кодування деякої ознаки зображення. Тоді шар, що отриманий в результаті виконання операції згортки з матрицею вагів, містить інформацію про наявність ознаки у шарі, що оброблюється, та координати цієї ознаки. Таким чином формується карта ознак [36].

У згортковій нейронній мережі застосовують декілька наборів вагів, які використовуються для пошуків різного типу ознак на зображенні. При чому ядра згортки безпосередньо не задаються. Вони формуються у процесі навчання мережі методом зворотнього розповсюдження похибки.

Кожен набір вагів (кожне ядро) у результаті операції згортки формує свій зразок карти ознак, у результаті чого нейронна мережа стає багатоканальною.

Крім згорткового шару, у згортковій мережі застосовується шар субдискретизації. Операція субдискретизації забезпечує зменшення розмірності

софрмованих карт ознак. На даному шарі головним завданням є отримання інформації щодо наявності деякої ознаки, тоді як положення цієї ознаки шукається на подальших кроках. Тому серед сукупності сусідніх нейронів карти ознак вибирається максимальний, який буде належати карті ознак меншої розмірності. За рахунок цього прискорюються подальші обрахунки, а також мережа стає більш інваріантною до масштабу вхідного зображення.

Типова архітектура згорткової нейронної мережі складається з великої кількості шарів. Спочатку сигнал проходить початковий шар мережі, далі відбувається проходження серії згорткових шарів, у який чергуються операції згортки та субдискретизації. У результаті цього формується карта ознак. На кожному наступному шарові карта ознак зменшується у розмірах, але збільшується її кількість каналів. Після проходження декількох шарів карта ознак вироджується у вектор або у скаляр, але таких карт стає сотні. Остаточно сформовані карти ознак подаються на вхід до повнозв'язного шару, яких може бути декілька.

2.4.1 Згортковий шар

Згорктовий шар є основним та найбільш важливим шаром згорткової нейронної мережі. Він складається з набору фільтрів (ядер згортки), кожен з яких застосовується по усім областям вхідних даних. Фільтр визначається набором вагів, які змінюються у процесі навчання [36].

На рисунку 2.27 показаний приклад застосування фільтру.

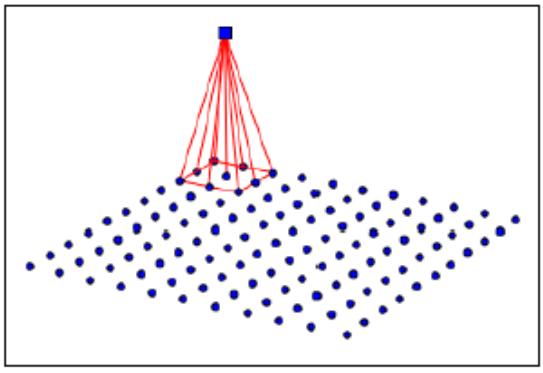


Рисунок 2.27 – Приклад застосування фільтру

Тут показаний двовимірний вхідний шар мережі, кожен нейрон якого являє собою інтенсивність кольору зображення з одним каналом та асоційований з однією вагою фільтра. Результатом застосування фільтру для його кожної позиції є зважена сума вхідних значень нейронів (значень активації нейронів). Тобто обраховується скалярний добуток вектору вагів та вектору значень активації нейронів [23].

Сукупність нейронів, значення активаціїї яких використовуються у процесі обрахунку зваженої суми називають рецептивним полем. Рецептивне поле є гіперпараметром та являє собою розмір фільтру [23].

Таким самим чином обраховуються значення активації нейронів наступного шару. Цей процес зображений на рисунку 2.28.

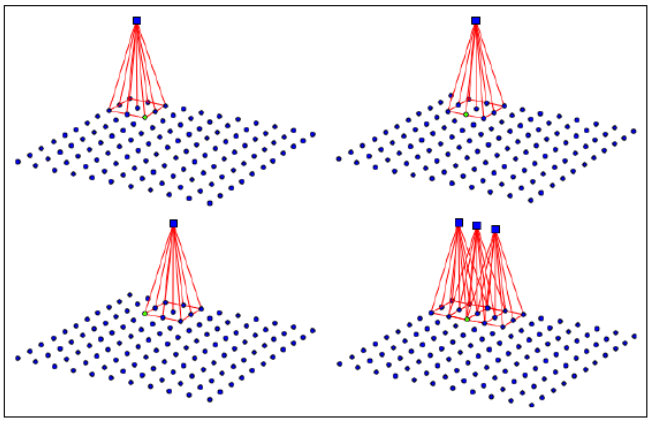


Рисунок 2.28 – Обрахунок значень активації нейронів

Важливою особливістю є те, що значення вагів фільтру не змінюються у процесі обрахунку операцій згортки для усього зображення. Тобто одні й ті самі значення вагів застосовуються для обрахунку усіх значень активації нейронів наступного шару, але для різних значень вхідних нейронів.

Цю особливість називають спільним використанням параметрів та застосовують для того, щоб [23]:

- Зменшити використання пам’яті та запобігти перенавчанню.

- Гарантувати, що фільтр зможе знайти положення шуканої ознаки на зображенні.

Така сама процедура відбувається і у випадку, коли на вхід мережі подається зображення з трьома каналами (RGB). Відповідна ситуація зображена на рисунку 2.29.

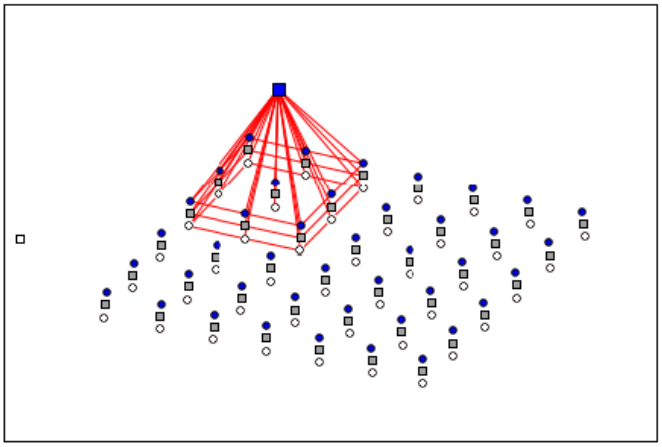


Рисунок 2.29 – Операція згортки для зображення з трьома каналами

У результаті операції згортки один об’єм активацій (вхідний) перетворюється на інший (вихідний). На кількість нейронів у вихідному об’ємі впливає глибина вхідного об’єму (depth), крок (stride) та доповнення нулями (zero-padding).

Глибина вхідного об’єму (depth) є гіперпараметром, що позначає кількість фільтрів (ядер згортки), що використовуються.

Крок (stride) – параметр, що визначає кількість пікселів, на які зсувається фільтр під час кожного зміщення. Найчастіше застосовують крок 1 та крок 2, тобто при кожному зсуві фільтр переміщується на 1 та 2 пікселі відповідно. Зазвичай крок є сталим для усіх вимірів вхідних даних. Головним ефектом більшого кроку є збільшення рецептивного поля вихідних нейронів Іншими словами, вихідний нейрон у такому разі «покриє» більшу площу вхідного шару, що дозволить йому визначити більш складні ознаки.

Приклад операції згортки для кроку 2 зображений на рисунку 2.30.

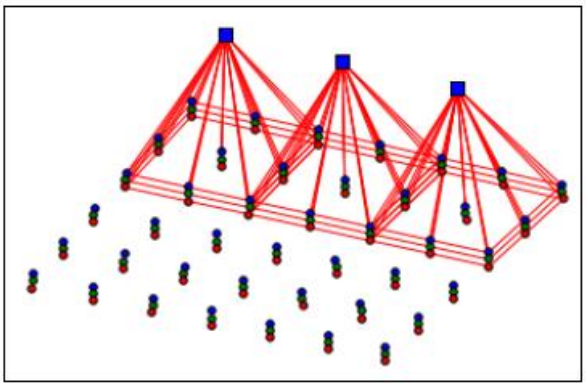


Рисунок 2.30 – Згортка з кроком 2

Доповнення нулями (zero-padding) також є гіперпараметром, який визначає кількість піселів нуьової інтенсивності, які додаються навколо границі зображення. Це дозволяє контролювати просторовий розмір вихідних об’ємів активацій (найчастіше використовується для збереження просторовго розміру вхідного об’єму, щоб висота та ширина вхідного і вихідного об’ємів була однаковою).

Приклад операції згортки з параметром доповнення нулями, який дорівнює 1, зображений на рисунку 2.31.

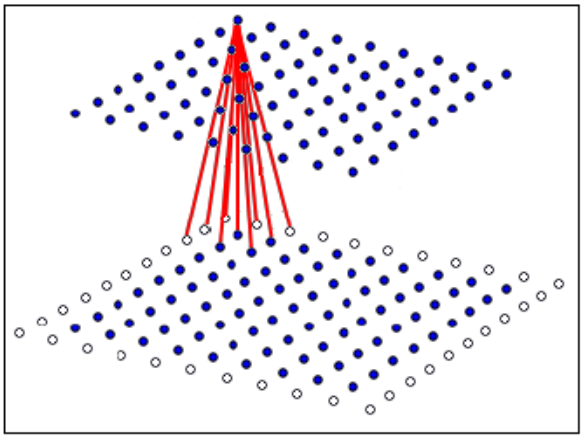


Рисунок 2.31 – Згортка з параметром доповнення нулями, який дорівнює 1

Тут білі нейрони є новими доданими нейронами. Вони не впливають на загальний результат згортки, адже пов’язані з іншими нейронами зв’язками з нульовими вагами.

Таким чином, розмір вихідного об’єму може бути обрахований як функція від розміру вхідного об’єму W.

Якщо F - розмір фільтру, S - крок, P - доповнення нулями, тоді розмір вихідного об’єму обчислюють за формулою [35]:

$$(W - F + 2P) / S + 1$$

Частинним випадком згортки є 1x1 згортка, тобто згортка з фільтром розміру 1x1. Такий фільтр не збільшує розмір рецептивного поля вихідних нейронів. Результатом його роботи буде точкове масштабування.

Такий тип згортки застосовують для зміни глибини між вхідними та вихідними об’ємами активацій. У загальному випадку є об’єм з глибиною D зрізів та M фільтрів для M вихідних зрізів. Кожен вихідний зріз сторюється шляхом застосування унікального фільтру по усім вхідним зрізам. Якщо використовується фільтр розміром 1x1 та D!=M, отримані зрізи будуть однакового розміру, але різної глибини об’єму. При цьому розмір рецептивного поля між входом і виходом не змінюється. Найбільш розповсюдженим варіантом використання є використання з метою зменшення вихідного об’єму.

2.4.2 Шар субдискретизації

Зазвичай у згорткових нейронних мережах між послідовними згортковими шарами періодично вставляють шари субдискретизації. Їх функція полягає у поступовому зменшенні просторового предстaлення даних, зменшенні кількості параметрів та кількості обрахунків у мережі, що дозволяє контролювати перенавчання [23].

Шар субдискретизації поділяє вхідний зріз на сітку, у якоій кожна комірка являє собою рецептивне поле деякого числа нейронів. Далі застосовується операція субдискретизації для кожної комірки сітки.

Шар субдискретизації не змінює глибину об’єму активацій, адже операція субдискретизації виконується незалежно для кожного зрізу.

Існує декілька типів шарів субдискретизації. Найбільш поширеним є шар субдискретизації з вибором максимального елементу (max pooling).

Операція max pooling вибирає нейрон з найбільшим значенням активації у кожному рецептивному полі (у кожній комірці сітки) і у подальшому використовує тільки це значення.

На рисунку 2.31 показана операція max pooling.

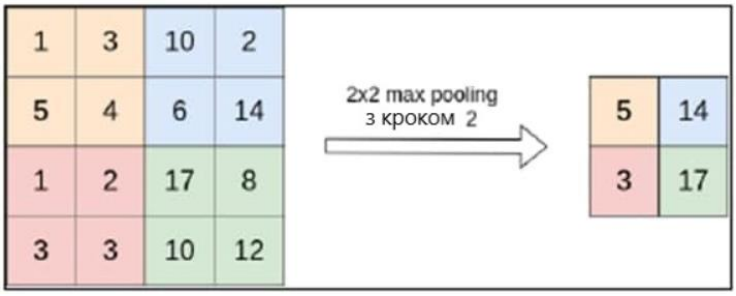
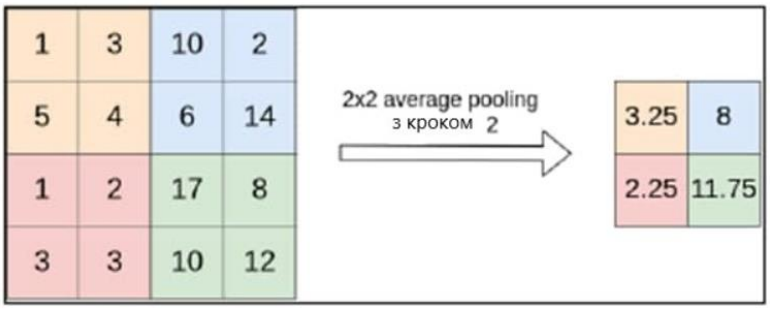


Рисунок 2.31 – Операція max pooling

Іншим типом шарів субдискретизації є шар субдискретизації з вибором середнього арифметичного значень активації нейронів (average pooling). Тобто виходом кожного рецептивного поля є середнє значення всіх активацій нейронів поля [23].

На рисунку 2.32 показаний приклад операції вибору середнього арифметичного значення.



Риснок 2.32 – Операція average pooling

Шар субдискретизації визначається двома параметрами:

- Крок (параметр, аналогічний до кроку у згортковому шарі).

- Рецептивне поле (параметр, еквівалентний розміру фільтра у згортковому шарі).

На практиці використовують лише дві комбінації цих параметрів: рецептивне поле 2 разом з кроком 2 та рецетивне поле 3 разом з кроком 2.

Якщо використовувати більші значення будь-якого з параметрів, то мережа втрачає багато корисної інформації.

На основі цих параметрів можна обрахувати вихідний розмір шару субдискретизації. Якщо I– розмір вхідного зрізу, F- розмір рецептивного поля, S- крок, тоді висота O_h та ширина O_w вихідного зрізу обчислюються за формулами [23]:

$$O_w = \frac{I_w - F_w}{S_w} + 1$$

$$O_h = \frac{I_h - F_h}{S_h} + 1$$

Характерною особливістю шару субдискретизації є те, що тут доповнення нулями не застосовується.

2.4.3 Основна структура згорткової нейронної мережі

Найбільш розповсюджена форма архітектрури згорткової нейронної мережі складається з блоків згортковий шар-шар ReLU, за якими йдуть шари субдискретизації. Цей шаблон повторюється до тих пір, поки вхідне зображення не буде просторово перетворене до зображення невеликого розміру. Тоді підключають повнозв'язні шари. Останній повнозв'язний шар містить вихідні дані, так як оцінки класів [35].

Таким чином, архітектура має вигляд:

$$INPUT \rightarrow \left[\left[CONV \rightarrow RELU\right]*N \rightarrow POOL?\right]*M \rightarrow$$
$$\left[FC \rightarrow RELU\right]*K \rightarrow FC$$

Тут знак * позначає повторення, знак ? – вибірковість операції.

Також $N \geq 0$ (як правило $N \leq 3$), $M \geq 0$, $K \geq 0$ (як правило $K < 3$).

На рисунку 2.33 зображений приклад базової архітектури згорткової нейронної мережі [23]:

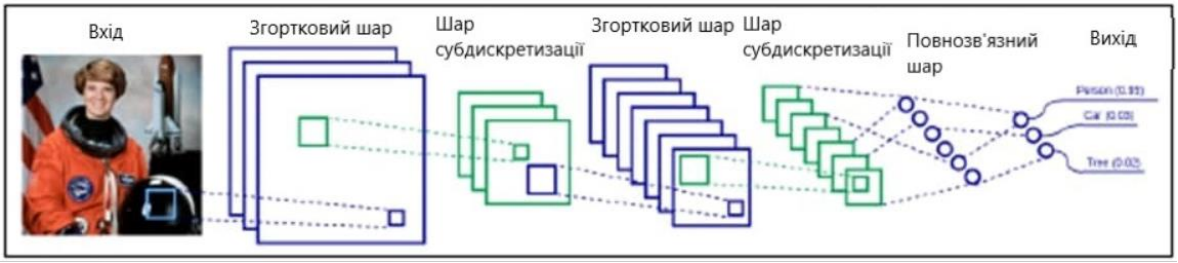


Рисунок 2.33 – Базова архітектура згорткової нейронної мережі

Найбільш поширеними архітектурами згорткових нейронних мереж є [35]:

- LeNet. Ця архітектура є першою успішно розробленою архітектурою згорткових мереж. Вона використовувалася для читання почтових індексів, цифр і. т. ін.

- AlexNet. Архітектура є дуже схожою до LeNet, але вона має більшу глибину, а також має послідовно з’єднані згорткові шари (без шару субдискретизації між ними).

- ZF Net. Ця архітектура є покращенням попередньої архітектури за рахунок налаштування гіперпараметрів. Наприклад, було збільшено розмір середніх згорткових шарів та зменшено крок фільтру на першому шарі.

- GoogLeNet. Особливістю архітектури є відносно мала кількість параметрів мережі. Також замість повнозв’язного шару у кінці мережі застосовують шар субдискретизації з вибором середнього арифметичного значення.

- VGGNet. Особливістю архітектури є її глибина та однорідна структура. Використовуються лише згортки з фільтрами 3x3 та операції субдискретизації з 2x2 рецептивним полем. Недоліком архітектури є надмірне використання пам’яті та параметрів.

- ResNet. Особливістю архітектури є наявність з’єднань швидкого доступу, а також інтенсивне використання пакетної нормалізації. У архітектурі відсутні повнозв’язні шари у кінці мережі.

2.4.4 Покращення результатів роботи згорткової мережі

Для того, аби покращити результат роботи згорткової мережі застосовують ряд процедур, таких як [23]:

- Попередня обробка даних (Data pre-processing).

- Регуляризація.

До методів регуляризації належать:

- Зменшення вагів (Weight decay).

- Виключення (Dropout).

- Аугментація даних (Data augmentation).

- Пакетна нормалізація (Batch normalization).

Для розв’язання задачі пошуку об’єктів в зображенні на вхід згорткової мережі подають зображення, що являє собою набір пікселів з інтенсивностями, які змінюються у діапазоні [0:255]. Але такий діапазон зміни значень не є оптимальним. Якщо, наприклад, на вхід мережі подається RGB зображення у якого інтенсивності пікселів одного каналу кольору є дуже великими у порівнянні з інтенсивностями пікселів інших каналів, то значення цього каналу будуть домінувати, зменшуючи вплив інших каналів. Це призводить до спотворення результатів, адже кожен канал є однаково важливим.

Для подолання цієї проблеми застосовують попередню обробку (нормалізацію) даних.

На практиці використовують два типи нормалізації:

- Масштабування ознак. Цей тип нормалізації перетворює інтенсивності кольорів у такий спосіб, щоб вони змінювалися у діапазоні [0,1].

Для цього інтенсивності перетворюють за формулою:

$$x = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Тут x - інтенсивність кольору, $x_{\min}$ - мінімальне значення інтенсивності кольору на зображенні, $x_{\max}$ - максимальне значення інтенсивності кольору на зображенні.

- Стандартна оцінка. Цей тип нормалізації перетворює інтенсивності кольорів за формулою:

$$x = \frac{x - \mu}{\sigma}$$

Тут x - інтенсивність кольору, $x_{\min}$ - мінімальне значення інтенсивності кольору на зображенні, μ - середнє значення інтенсивності кольору усього зобрадження, σ - стандартне відхилення інтенсивності кольору усього зображення.

Іншою розповсюдженою проблемою є перенавчання – центральна проблема машинного навчання. Для її подолання використовують методи, відомі як регуляризація [23].

Першим методом регуляризації є метод зменшення вагів. Якщо ваги мережі мають великі значення, то функція втрат зростає. Даний метод регуляризації полягає у зменшення вагів шляхом додавання доданка до функції втрат. Це допомагає уникнути перенавчання.

На практиці доданок додають у процесі оновлення вагів.

$$w \to w - \eta \nabla (J(w))$$
$$w \to w - \eta (\nabla (J(w)) - \lambda w)$$

Тут w - ваги, η - коефіцієнт швидкості навчання, J - функція втрат, λ - коефіцієнт зменшення вагів.

Другим методом регуляризації є метод виключення. Він застосовується до виходів деяких шарів мережі та полягає у випадковому та періодичному видаленню частини нейронів разом з їхніми вхідними та вихідними вагами.

Під час навчання на міні-пакеті даних кожен нейрон має ймовірність виключення. Це робиться для того, щоб кожен нейрон не спирався у великій мірі

на значення інших нейронів. Процедура виключення може застосовуватися після згорткового шару, шару субдискретизації чи повнозв’язного шару.

На рисунку 2.34 показана процедура виключення у повнозв’язному шарі.

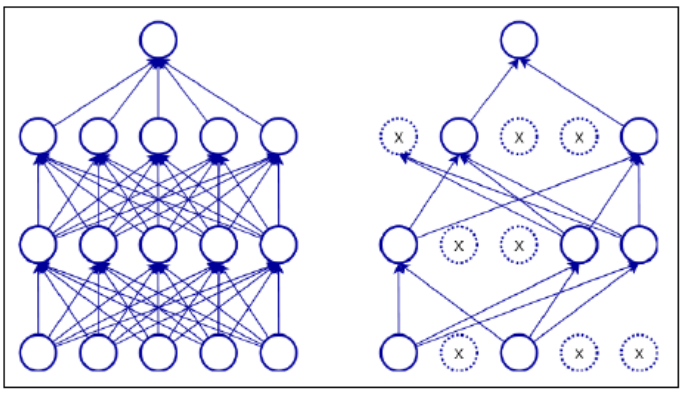


Рисунок 2.34 – Процедура виключення у повнозв’язному шарі

Третім та найбільш ефективним методом регуляризації є аугментація даних. Якщо набір даних є досить малим, то мережа може перенавчитися.

Для уникнення цього кількість даних збільшують шляхом аугментації наявних даних. Застосовують випадкову аугментацію зображень перед тим, як почати тренування мережі.

Найбільш поширеними варіантами аугментації зображень є:

- Поворот.
- Горизонтальне чи вертикальне віддзеркалення.
- Збільшення чи зменшення певних областей зображення.
- Нарізання.
- Зміщення.
- Зміна контрасту чи яскравості.

Приклад аугментації зображення показаний на рисунку 2.35.

Рисунок 2.35 – Приклад аугментації зображення

Третім методом регуляризації є пакетна нормалізація, який полягає у нормалізації вихідних даних проміжних шарів для кожного міні-пакету даних таким чином, щоб середнє значення активації було близьким донуля, а його стандартне відхилення було близьким до одиниці. Пакетна нормалізація використовується як для згорткових шарів, так і для повнозв’язних шарів. Мережі з пакетною нормалізацією навчаються швидше.

3 МОДЕЛЬ YOLOV3

YOLO – це детектор об’єктів (алгоритм для пошуку об’єктів в зображенні), який використовує ознаки, отримані у результаті навчання глибокої згорткової нейронної мережі [37].

Кожен детектор об’єктів у результаті роботи повертає список знайдених об’єктів з наступною інофрмацією про кожен об’єкт [23]:

- Клас об’єкту.

- Ймовірність (оцінка впевненості (confidence score)) у діапазоні [0,1], яка вказує на те, на скільки «впевненим» є детктор у тому, що об’єкт знаходиться в конкретній позиції.

- Координати прямокутної області на зображенні, де знаходиться об’єкт. Цей прямокутник ще називають обмежувальною рамкою (bounding box).

Таким чином, особливостями детектору YOLO є [23]:

- Використання згорткових шарів, залишкових з’єднань (residual connections) та пакетної нормалізації. Відсутність шарів субдискретизації. Використання трьох різних масштабів зображення для виявлення об’єктів.

- Прийняття мережею всього зображення на вхід та видача на виході обмежувальних рамок, класів об’єктів, оцінок впевненості для всіх об’єктів за один прохід зображення через мережу.

YOLO використовує лише згорткові шари, що робить мережу повністю згортковою. Мережа містить 75 згорткових шарів, залишкові з’єднання та шари збільшення масштабу (upsampling layers). Також використовуються згорткові шари з кроком 2 для зменшення масштабу карт ознак. Це допомагає запобігти втраті низькорівневих ознак [37].

Хоча YOLO є повністю згортковою мережею, що є інваріантною до розмірів вхідних зображень, на практиці доводиться приводити усі зображення до єдиного масштабу. Це надає можливість обробляти зображення пакетами, де усі пакети обробляються паралельно за допомогою GPU, що пришвидшує роботу алгоритму.

Як і у випадку з іншиими детекторами, у YOLO ознаки, отримані за допомогою згорткових шарів, передаються до класифікатору/регресору, який прогонозує положення об'єктів. Прогнозування у YOLO здійснюється шляхом використання 1x1 згорткового шару.

У результаті виконання таких процедур на виході отримується карта ознак. Так як використовувалися операції згортки 1x1, то розмір карти прогнозів буде такий самий як і розмір карти ознак. Отримана карта прогнозів складається з комірок, кожна з яких здатна прогнозувати фіксовану кількість обмежувальних прямокутників.

Таким чином, вхідне зображення розбивається на сітку комірок, де кількість комірок дорівнює розмірності фінальної карти ознак. Центр кожної комірки сітки трактується як центр області, в якій може знаходитися об'єкт. Об'єкт може знаходитися як всередині комірки (тоді його обмежувальна рамка буде меншою за комірку), так і охоплювати декілька комірок (тоді його обмежувальна рамка буде більшою). За допомогою якорів (anchor boxes) в кожній комірці сітки може бути знайдено декілька об'єктів, але кожен об'єкт асоціюється лише з однією коміркою. У випадку, якщо обмежувальна рамка об'єкту охоплює декілька комірок, то об'єкт асоціюється з тією коміркою, в якій розташований центр обмежувальної рамки [23].

Отже, вхідне зображення розбивається на сітку для того, аби визначити, яка комірка карти прогнозів відповідає прогнозу наявності об'єкта. У результаті роботи алгоритму на виході для кожної комірки отримують набір значень:

$$[b_x, b_y, b_w, b_h, p_c, c_1, \ldots, c_n]$$

Для визначення розмірів об'єкта було б зручно прогнозувати ширину та висоту обмежуальної рамки. Але на практиці це призводить до нестабільності градієнтів у процесі тренування мережі. Тому визначають зміщення відносно спеціальних рамок, які ще називають якорями (anchors).

YOLO має три якорі, що дозволяє прогнозувати три обмежувальні рамки для кожної комірки сітки [13].

На наступних формулах показано, як змінюються вихідні значення мережі для отримання прогнозів обмежувальних рамок [13]:

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

У попередніх формулах:

b_x b_y b_w b_h -координати центру, ширина та висота прогнозованиої рамки.

t_x t_y t_w t_h -значення, які формуються на виході мережі.

c_x c_y -координати верхнього лівого кута комірки.

p_w p_h -розміри якоря для обмежувальної рамки.

Так як YOLO не прогнозує абсолютні значення координат центрів обмежувальних рамок, то вихідні значення мережі подаються на вхід функції сігмоїди. Тобто вихідні значення змінюються у діапазоні [0,1]. Таким чином, прогнозуються зміщення відносно лівого верхнього кута комірки, які нормалізуються значеннями розмірності комірки карти ознак.

Приклад прогнозування розмірностей обмежувальної рамки зображений на рисунку 3.1.

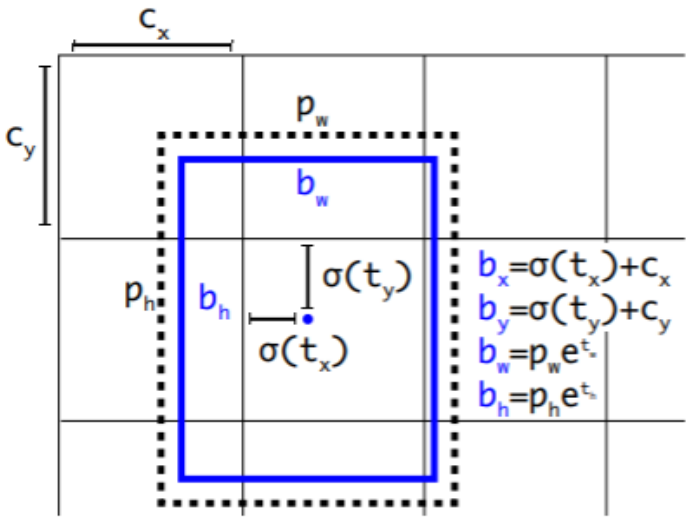


Рисунок 3.1 – Прогнозування розмірностей обмежувальної рамки

Оцінка впевненості (confidence score) p_c є ймовірністю того, що об'єкт розташований всередині обмежувальної рамки. Ця оцінка також подаєтья на вхід функції сігмоїди, щоб її значення змінювалися у діапазоні [0,1].

Інші значення, які отримують для кожної комірки – це ймовірності об'єктів $c_1,\ldots,c_n$, де n – кількість класів об'єктів.

YOLO проводить пошук об'єктів в зображенні у трьох масштабах. Якщо вхідне зображення має розмірність 416x416 пікселів, то пошук об'єктів здійснюється за масштабів 13x13, 26x26 та 52x52 [37].

Мережа здійснює субдискретизацію вхідного зображення (зменшення розмірності) до першого шару пошуку, де пошук здійснюється з використанням карт ознак шару з кроком 32. Далі шари збільшують розмірність вдвічі та об'єднуються з картами ознак попередніх шарів, які мають такі ж самі розмірності. На цьому етапі пошук об'єктів здійснюється за допомогою шару з кроком 16. Далі повторюється процес збільшення розмірності, а також фінальний пошук об'єктів здійснюється за допомогоюшару з кроком 8 [37].

У кожному масштабі кожна комірка сітки прогнозує 3 обмежувальні рамки, використовуючи для цього 3 якорі, у результаті чого загальна кількість якорів,що використовуються, стнаовить 9. Тобто якорі є різними для кожного масштабу.

Проводячи пошук об'єктів за трьох різних масштабів мережа здатна краще виявляти об'єкти малих розмірів.

Приклад пошуку об'єктів за масштабу 13x13 показаний на рисунку 3.2, приклад пошуку за масштабу 26x26 показаний на рисунку 3.3, приклад пошуку за масштабу 52x52 показаний на рисунку 3.4 [37].

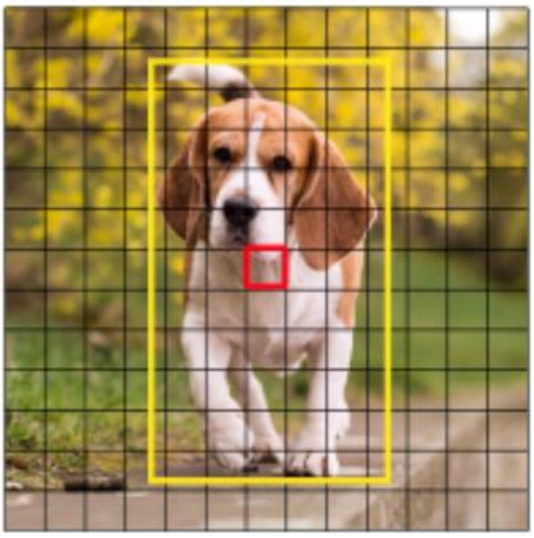


Рисунок 3.2 – Пошук об’єктів за масштабу 13x13

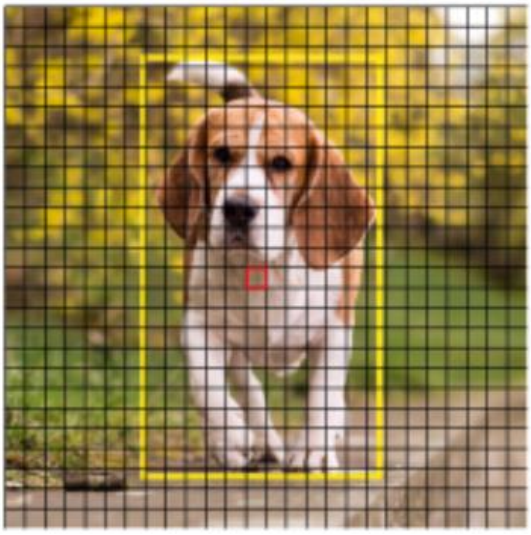


Рисунок 3.3 – Пошук об’єктів за масштабу 26x26

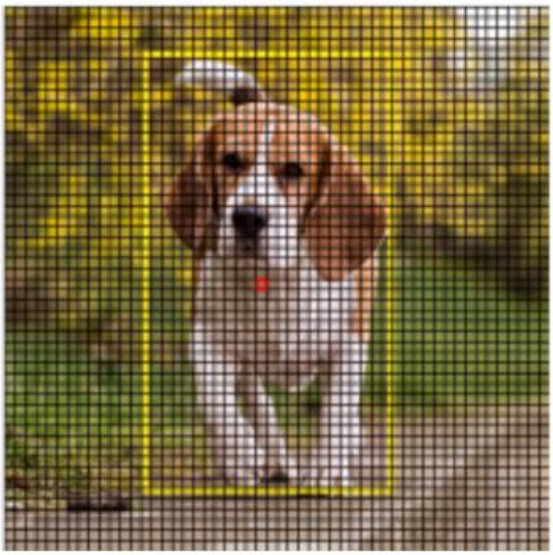


Рисунок 3.4 – Приклад пошуку об’єктів за масштабу 52x52

Таким чином, для зображення розмірністю 416x416 YOLO прогнозує ((52*52) + (26*26) + (13*13))*3 , що дорівнює 10647 обмежувальних рамок.

Для того, щоб виділити з усієї сукупності рамок ті, що позначають шукані об’єкти, використовують процедуру non-maximum suppression.

По-перше, відкидають ті обмежувальні рамки, у яких оцінка впевненості (confidence score) менша за деяке порогове значення (найчастше 0.6). По-друге, серед тих рамок, що залишилися, вибирають рамку з найбільшою оцінкою

впевненості. По-третє, відкидають ті обмежувальні рамки, для яких IoU >=0.5 з рамкою, яка була вибрана на другому кроці [23].

IoU (Intersection over Union) – це відношення площі перетину обмежувальних рамок до площі об’єднання обмежувальних рамок. Обрахунок IoU показаний на рисунку 3.5 [23].

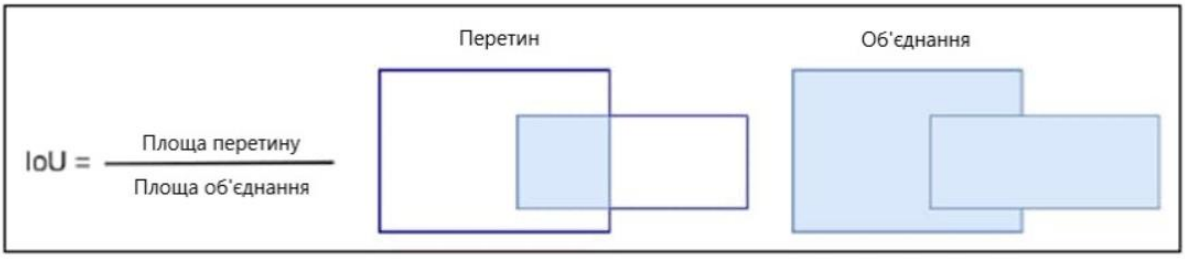


Рисунок 3.5 – Обрахунок IoU

Архітектура YOLO зображена на рисунку 3.6 [38].

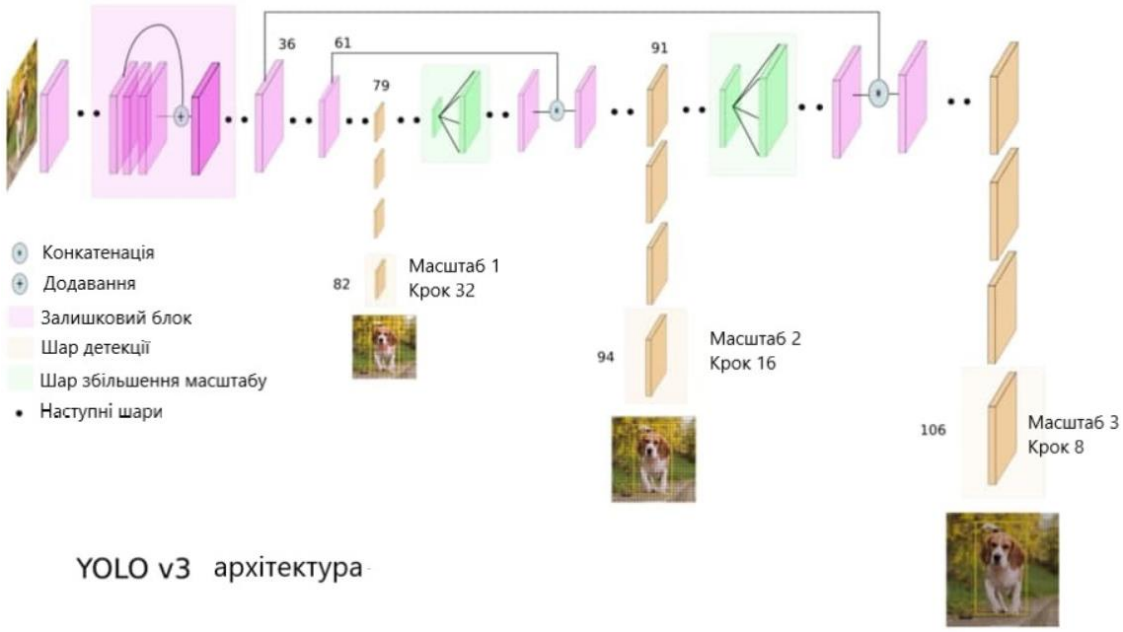


Рисунок 3.6 – Архітектура YOLO

Приклад результату роботи детектору YOLO показаний на рисунку 3.7 [38].

Рисунок 3.7 - Приклад результату роботи детектору YOLO

4 РОЗРОБКА ПРОГРАМНО-АЛГОРИТМІЧНОГО КОМПЛЕКСУ ДЛЯ ПРАКТИЧНОГО РОЗВ‘ЯЗКУ ПОСТАВЛЕНОЇ ЗАДАЧІ

Розробка програмно – алгоритмічного комплексу складається з таких основних кроків:

- Завантаження та трансформація даних.
- Створення користувацьких наборів даних.
- Створення моделі.
- Визначення функції похибки, оптимізатора та IOU метрики.
- Тренування моделі.
- Розгортання моделі.

У якості даних використаємо набір даних Wildfire Smoke Dataset [39]. Набір складається з 737 зображень розмірністю 640x480 пікселів. Усі зображення були розділені на три категорії: тренувальні дані, дані перевірки та тестові дані. Відповідно тестові дані складаються з 517 зображень, дані перевірки складаються з 148 зображень, а тестові дані – з 75 зображень. До кожної категорії даних додається csv-файл, у якому у кожному рядку (для кожного зображення) розміщена інформація у наступному порядку:

- назва зображення;
- ширина та висота зображення;
- клас об’єкту, що шукається на зображенні;
- положення обмежувальної рамки об’єкту (Координати верхнього лівого та нижнього правого кута рамки).

Вигляд частини csv-файлу для категорії тренувальних даних показаний на рисунку 4.1.

	A	B	C	D	E	F	G	H	I	J	K
1	filename,width,height,class,xmin,ymin,xmax,ymax										
2	ck0qd8gs6ko7j0721x25cv4o3_jpeg.rf.005f5707706e4d68702bbf8e0ce96a63.jpg,640,480,smoke,125,190,177,286										
3	ck0t40rhdz68s0a46ekx049a6_jpeg.rf.00403179fe5f0eb665e4af5845be125e.jpg,640,480,smoke,326,207,494,249										
4	ck0m0ch9ugnna07940o8x989j_jpeg.rf.0101cdb46a16b3fde020710836b4af0b.jpg,640,480,smoke,308,166,582,257										
5	ck0rr6bfa9b3w0721aw5unwdy_jpeg.rf.00982c053d66c090d55b3d775a722aff.jpg,640,480,smoke,241,204,310,244										
6	ck0uk75x5ysls0721e5a9j891_jpeg.rf.00d7fd8503e1e0e6a66294f3db79b346.jpg,640,480,smoke,523,208,619,288										
7	ck0m0f8x4hwsq0838adn9hkti_jpeg.rf.02c20dbd46efee307ec38486fc3f5da2.jpg,640,480,smoke,224,200,302,238										
8	ck0na7jmkg9gj0a46p7jofg58_jpeg.rf.023825dac5dc5fbd89bbd5d83f89a5a8.jpg,640,480,smoke,96,222,238,270										
9	ck0nf3wev47fm0794xfr1xuhs_jpeg.rf.025a886c2323bc0eb8b24cabfc681aab.jpg,640,480,smoke,220,203,257,237										
10	ck0knam319ybv0701clozevt3_jpeg.rf.01d252cb506d60495e9f965a4cf48418.jpg,640,480,smoke,482,296,525,320										
11	ck0oumds8r2ws0794yqsnls8v_jpeg.rf.012d2c43910a089d6fec492c2b914840.jpg,640,480,smoke,325,207,406,251										
12	ck0rr05k797je0721hrz4bw9e_jpeg.rf.031b5346c52c69e28fc9345ca7b28010.jpg,640,480,smoke,500,184,636,269										
13	ck0u07w91ulq009446ol1bta4_jpeg.rf.05a02ac3bf9a27f86cc6c315a72ff6c7.jpg,640,480,smoke,258,203,308,240										
14	ck0kd86qe8gf507018qw8usfe_jpeg.rf.04a3091cdb2674d667aa79b142f77db6.jpg,640,480,smoke,217,196,250,237										
15	ck0kmh9hhkg5e0a46we9ywaxl_jpeg.rf.05a2d3d0278ae39a0a626bfd173d6b94.jpg,640,480,smoke,473,198,513,243										
16	ck0k99r7bir3f0a460bctrlmy_jpeg.rf.0520c5989e9d6326d3beb346e7edc7c2.jpg,640,480,smoke,137,272,187,310										
17	ck0u0iymarljc079432rld6ur_jpeg.rf.049e444971d411f2bc15d14189f8ebc9.jpg,640,480,smoke,342,271,526,316										
18	ck0t5243y02ss0848d4w8z1bl_jpeg.rf.073eb5726a7f3fca2caa57b085c7b655.jpg,640,480,smoke,194,207,213,225										

Рисунок 4.1 – Вигляд частини csv-файлу для категорії тренувальних даних

Узагалі, існує три варіанти представлення обмежувальної рамки:

- $[x_0,y_0,w,h]$, де x_0, y_0 – координати верхнього лівого кута обмежувальної рамки, w, h – ширина та висота обмежувальної рамки;

- $[x_0,y_0, x_1,y_1]$, де x_0, y_0 – координати верхнього лівого кута обмежувальної рамки, x_1, y_1 – координати нижнього правого кута обмежувальної рамки;

- $[x_c,y_c,w,h]$, де x_c, y_c – координати центру обмежувальної рамки, w, h – ширина та висота обмежувальної рамки.

У заданому наборі даних обмежувальні рамки подаються у другому форматі. Перетворимо дані таким чином, щоб обмежувальні рамки подавалися у третьому форматі. На рисунку 4.2 показане зображення з обмежувальною рамкою та чотирма відповідними їй числами (третій формат представлення).

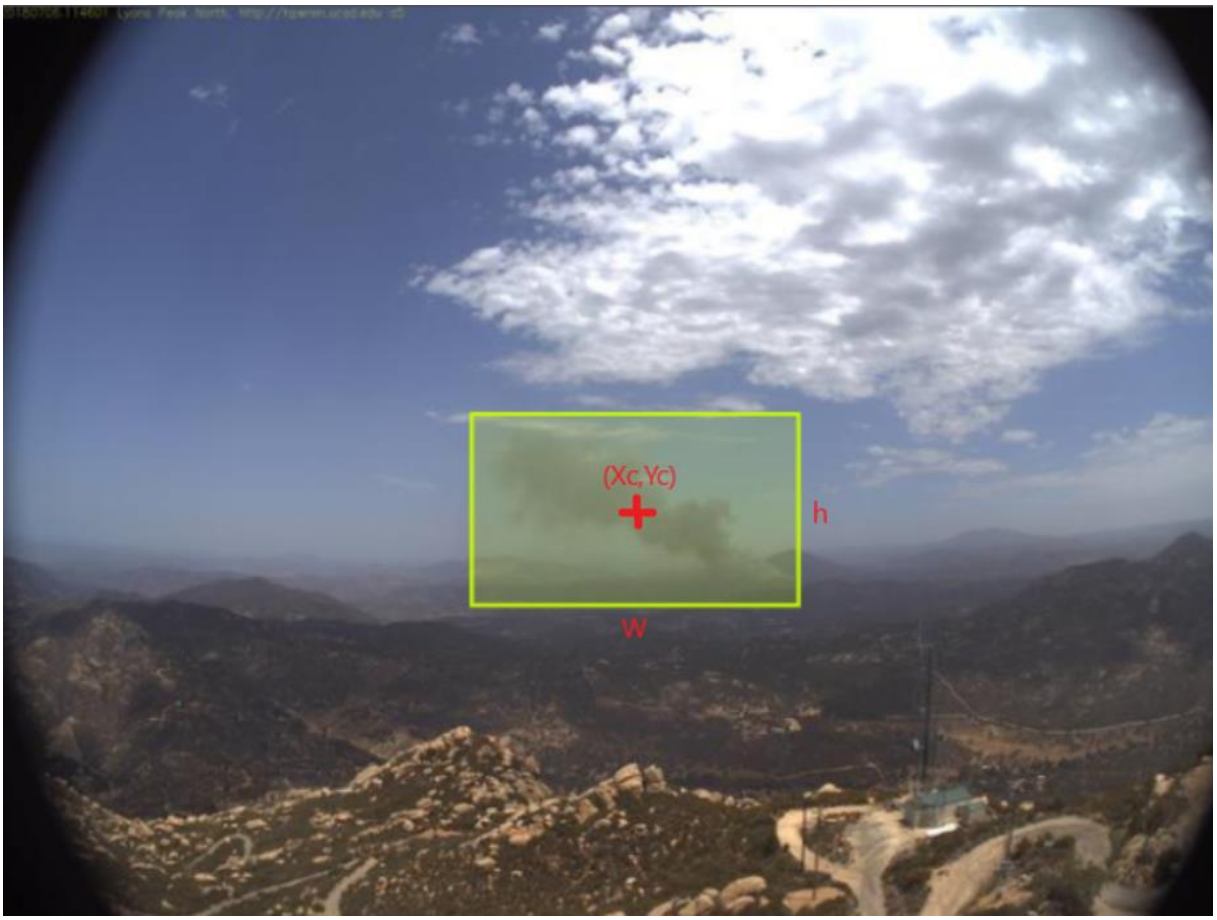


Рисунок 4.2 - Зображення з обмежувальною рамкою та чотирма відповідними їй числами (третій формат представлення)

Першим кроком створимо клас Smoke_dataset, який наслідується від батьківського класу Dataset [40], розміщеного у файлі torch.utils.data та який призначений для завантаження та обробки даних. У створеному класі перевизначимо методи __init__ та __getitem__. Метод __len__ повертає довжину набору даних та викликається за допомогою функції len. Метод __getitem__ повертає зображення та мітку (список з чотирьох чисел, які визначають обмежувальну рамку об’єкта).

Метод __init__ приймає чотири аргументи, два з яких є опціональними: self, path2data (шлях до даних), transform (функція, що здійснює перевторення данихж; опціональний аргумент), trans_params (словник, у якому розміщені параметри трансформації даних, опціональний аргумент).

У методі __init__ завантажимо дані за допомогою функції read_csv, яка знаходиться в пакеті pandas. Дані завантажуються у DataFrame, який можна

розглядати як спеціалізований словник. Далі відбувається перетворення формату представлення обмежувальних рамок.

У якості полів класу зберігається інофрмація про назви зображень (imgName), мітки (labels), індекси об’єкту DataFrame (ids), шляхи до зображень (fullPath2img), функція трансформації зображення (transform) та параметри трансформації (trans_params).

Код методу __init__ зображений на рисунку 4.3.

```
class Smoke_dataset(Dataset):
    def __init__(self,path2data, transform=None,trans_params=None):
        if trans_params["p_hflip"] > 0:
            path2dataf = os.path.join(path2data,"train")
            path2labels = os.path.join(path2dataf,"train_labels.csv")
        else:
            path2dataf = os.path.join(path2data,"valid")
            path2labels = os.path.join(path2dataf,"valid_labels.csv")
        labels_df = pd.read_csv(path2labels)
        x_min = labels_df["xmin"]
        y_min = labels_df["ymin"]
        x_max = labels_df["xmax"]
        y_max = labels_df["ymax"]
        labels_df["centerx"] = x_min+(x_max - x_min)/2
        labels_df["centery"] = y_min+(y_max - y_min)/2
        labels_df["w"] = (x_max - x_min)
        labels_df["h"] = (y_max - y_min)
        self.labels = labels_df[["centerx","centery","w","h"]].values
        self.imgName = labels_df["filename"]
        print(labels_df.head())

        self.ids = labels_df.index
        self.fullPath2img = [0]*len(self.ids)
        for id_ in self.ids:
            if trans_params["p_hflip"] > 0:
                self.fullPath2img[id_]=os.path.join(path2data,"train",self.imgName[id_])
            else:
                self.fullPath2img[id_]=os.path.join(path2data,"valid",self.imgName[id_])
        self.transform = transform
        self.trans_params = trans_params
```

Рисунок 4.3 – Код методу __init__

Код методу __len__ показаний на рисунку 4.4.

```
    def __len__(self):
        return len(self.labels)
```

Рисунок 4.4 – Код методу __len__

У методі __getitem__ відбувається зчитування зображення за допомогою функції open модуля Image з файлу PIL. Якщо при конструюванні об’єкта класу

Smoke_dataset у якості параметра була передана функція трансформації даних, то відбувається перетворення зображення та її мітки. У результаті метод __getitem__ повертає зображення та мітку.

Код методу __getitem__ зображений на рисунку 4.5.

```
def __getitem__(self,idx):
    image = Image.open(self.fullPath2img[idx])
    label = self.labels[idx]
    if self.transform:
        image,label = self.transform(image,label,self.trans_params)
    return image, label
```

Рисунок 4.5 - Код методу __getitem__

Другим кроком визначимо функцію трансформації даних (transformer) та параметри трансформації тренуальних даних (trans_params_train) і параметри трансформації даних перевірки (trans_params_val).

Аугментація та трансформація даних є важливим кроком під час тренування алгоритмів глибокого навчання, особливо у випадку малої кількості вхідних даних. Серед методів трансформації найбільш поширеними є горизонтальне та вертикальне віддзеркалення зображення, зсув зображення, зміна контрастності та яскравості зображення, гамма-перетворення інтенсивностей пікселів зображення, перетворення зображення на чорно-біле, а також поворот зображення на певний кут. Також зображення перетворюються таким чином, щоб вони мали розмірність 256x256 пікселів. Перетворючи зображення за необхідності змінюють відповідні їм мітки.

Код функцій, які змінюють масштаб зображень (resize_img_label), виконують горизонтальне (random_hflip) та вертикельне (random_vflip) віддзеркалення зображень, зсувають зображення (random_shift), перетворюють зображення на чорно-біле (grayscale) та повертають зображення на певний кут (rotation) показаний на рисунку 4.6.

Усі розглянуті перевторення відбуваються за допомогою функцій з пакету torchvision.transforms.functional. Так, зміна масштабу зображення відбувається за

допомгою функції resize, горизонтальне та вертикальне віддзеркалення здійснюються за допомогою функцій hflip та vflip відповідно, зсув зображення – за допомогою функції affine, перевтореня на чорно-біле – за допомогою функції to_grayscale, поворот на певний кут – функція rotate.

Зсув зображення по вертикалі та горизонталі відбувається на довжину не більше ніж на двадцять відсотків від загальної висоти та ширини зображення відповідно. Це обмеження задається параметром max_translate. Припускається, що поворот завжди здійснюється на кут у діапазоні (-30, 30) градусів. Формули зміни координат центрів обмежувальних рамок при повороті зображення взяті з лінійної алгебри [41]. Інші формули зміни міток (координат центрів обмежувальних рамок) є очевидними.

```python
def resize_img_label(image,label = (0.,0.),target_size = (256,256)):
    w_orig,h_orig = image.size
    w_target,h_target = target_size
    cx, cy = label
    image_new = TF.resize(image,target_size)
    label_new = cx/w_orig*w_target, cy/h_orig*h_target
    return image_new,label_new

def random_hflip(image,label):
    w,h = image.size
    x,y = label
    image = TF.hflip(image)
    label = w-x, y
    return image,label

def random_vflip(image,label):
    w,h = image.size
    x,y = label
    image = TF.vflip(image)
    label = x,w-y
    return image, label

def random_shift(image,label,max_translate=(0.2,0.2)):
    w,h = image.size
    max_t_w,max_t_h = max_translate
    cx,cy = label
    trans_coef = np.random.rand()*2 - 1
    w_t = int(trans_coef*max_t_w*w)
    h_t = int(trans_coef*max_t_h*h)
    image=TF.affine(image,translate=(w_t,h_t),shear=0,angle=0,scale=1)
    label = cx+w_t, cy+h_t
    return image, label

def grayscale(image,label):
    image = TF.to_grayscale(image,3)
    return image, label

def rotation(image,label):
    w,h = image.size
    angle = random.randrange(-30, 30)
    angle_l = -angle/57.3
    #angle = -30
    #angle_l = np.pi/6
    image = TF.rotate(image,angle)

    label_1 = (label[0]*w-w/2)*np.cos(angle_l)-(label[1]*h-h/2)*np.sin(angle_l)+w/2
    label_2 = (label[0]*w-w/2)*np.sin(angle_l)+(label[1]*h-h/2)*np.cos(angle_l)+h/2
    label[0] = label_1
    label[1] = label_2
    label[0] =label[0]/w
    label[1] =label[1]/h

    return image, label
```

Рисунок 4.6 – Код функцій, що здійснюють основні перетворення зображень та відповідних їм міток.

Усі наведені перетворення відбуваються з зображеннями з певною ймовірністю. Ці ймовірності задаються для кожного перетворення окремо та разом формують словники параметрів трансформації тренуальних даних

(trans_params_train) і параметрів трансформації даних перевірки (trans_params_val). Для даних перевірки трансформації не застосовуються, тому усі ймовірності нульові. Словники параметрів зображені на рисунку 4.7.

```
trans_params_train={
    "target_size":(256,256),
    "p_hflip":0.5,
    "p_vflip":0.5,
    "p_shift":0.5,
    "max_translate":(0.2,0.2),
    "p_brightness":0.5,
    "brightness_factor":0.2,
    "p_contrast":0.5,
    "contrast_factor":0.2,
    "p_gamma":0.5,
    "p_gray" : 0.5,
    "p_rotate":0.0,
    "gamma":0.2,
    "scale_label":True,
}
```

```
trans_params_val={
    "target_size":(256,256),
    "p_hflip":0.0,
    "p_vflip":0.0,
    "p_shift":0.0,
    "p_brightness":0.0,
    "p_contrast":0.0,
    "p_gamma":0.0,
    "p_gray" : 0.0,
    "p_rotate":0.0,
    "gamma":0.0,
    "scale_label":True,
}
```

Рисунок 4.7 – Словники параметрів трансформації

Також за допомогою функції scale_label перетворимо мітки кожного зображення таким чином, щоб відповідні їм числові значення змінювалися у діапазоні (0,1). Тобто за допомогою даної функції відбувається проектування значень на діапазон (0,1).

Код функції scale_label зображений на рисунку 4.8.

```
def scale_label(a,b):
    div = [ai/bi for ai,bi in zip(a,b)]
    return div
```

Рисунок 4.8 – Код функції scale_label

Таким чином, створюють два об’єкти класу Smoke_dataset - train_ds та val_ds. Код створення об’єктів показаний на рисунку 4.9.

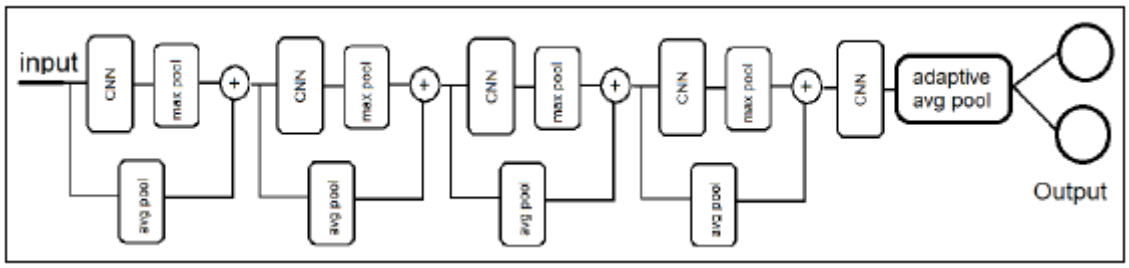
```
train_ds = Smoke_dataset(path2data,transformer,trans_params_train)
val_ds = Smoke_dataset(path2data,transformer,trans_params_val)
```

Рисунок 4.9 – Код створення об’єктів

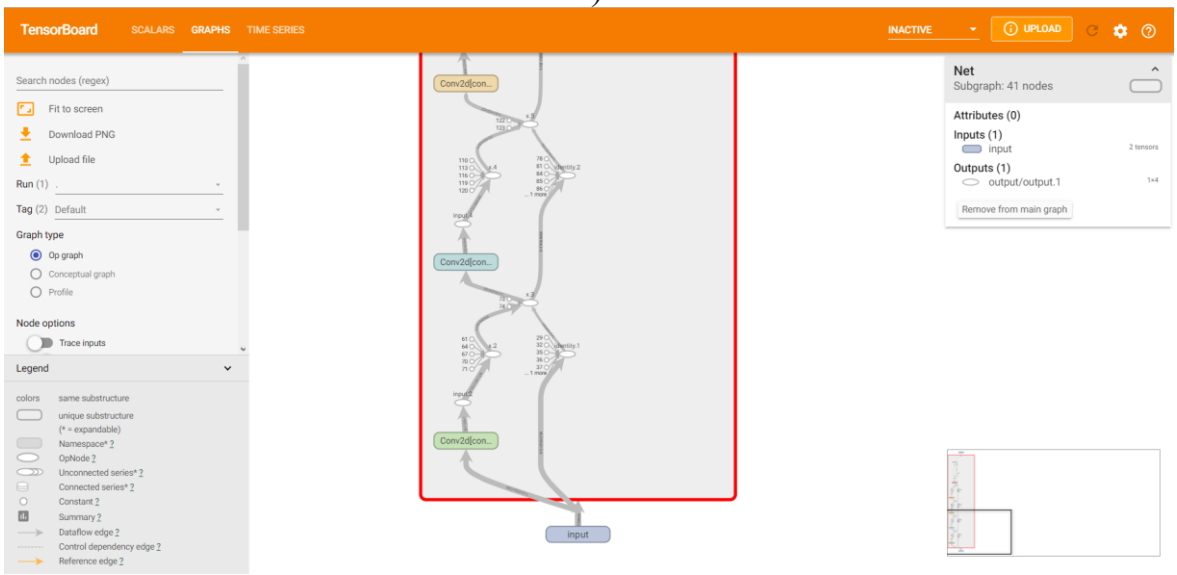
Наступним кроком використаємо клас DataLoader [40] з пакету torch.utils.data та створимо об’єкти-завантажувачі даних: train_dl, val_dl. Використовуючи ці об’єкти можна автоматично отримувати міні-пакети даних для обробки. Розміри міні-пакетів даних є гіперпараметрами. Встановимо розмір міні-пакету тренувальних даних у вісім зображень, а міні-пакету даних перевірки - шістнадцять зображень.

Для розв’язку поставленої задачі використаємо згорткову нейронну мережу.

Структура побудованої мережі зображена на рисунку 4.10.



а)



б)

Рисунок 4.10 – Структура побудованої згорткової нейронної мережі:
а – загальний вигляд моделі, б – детальний вигляд моделі, отриманий за допомогою набору інструментів Tensorboard

Модель складається з декількох згорткових шарів та шарів субдискретизації. Модель отримує на вхід RGB-зображення заданого масштабу, а на виході видає чотири числа – координати центру шуканого об’єкту та висота і ширина відповідної обмежувальної рамки.

Створимо клас Net, який відображає модель мережі. Даний клас наслідуєтсья від батьківського класу Module [42] з пакету torch.nn та містить два методи: __init__ та forward.

У методі __init__ визначаються складові частини моделі: 5 згорткових шарів та 1 повнозв’язний шар. Для кожного згорткового шару встановлений параметр padding=1 (доповнення нулями).

У метод forward визначаються зв’язки між шарами. У якості функції активації використовується функція relu з пакету torch.nn.functional. Також наявні 4 блоки skip-connections. skip-connections зв’язки застосовуються у мережах з великою кількістю шарів для подолання проблеми затухаючих градієнтів під час зворотнього розповсюдження похибки.

Параметри моделі задаються у словнику params_model. Відповідний код показаний на рисунку 4.11.

```
params_model={
    "input_shape":(3,256,256),
    "initial_filters": 32,
    "num_outputs": 4,
}

model = Net(params_model)
```

Рисунок 4.11 – Код, що визначає словник з параметрами моделі

Тут вважається, що початкова кількість фільтрів дорівнює 32.

Код класу Net показаний на рисунку 4.12.

```
class Net(nn.Module):
    def __init__(self,params):
        super(Net, self).__init__()
        C_in,H_in,W_in = params["input_shape"]
        init_f = params["initial_filters"]
        num_outputs=params["num_outputs"]
        self.conv1 = nn.Conv2d(C_in,init_f,kernel_size=3,stride=2,padding=1)
        self.conv2 = nn.Conv2d(init_f+C_in,2*init_f,kernel_size=3,stride=1,padding=1)
        self.conv3 = nn.Conv2d(3*init_f+C_in, 4*init_f,kernel_size=3,padding=1)
        self.conv4 = nn.Conv2d(7*init_f+C_in, 8*init_f,kernel_size=3,padding=1)
        self.conv5 = nn.Conv2d(15*init_f+C_in, 16*init_f,kernel_size=3,padding=1)
        self.fc1 = nn.Linear(16*init_f, num_outputs)
    def forward(self,x):
        identity = F.avg_pool2d(x,4,4)
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = torch.cat((x, identity), dim = 1)

        identity = F.avg_pool2d(x,2,2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = torch.cat((x,identity),dim = 1)

        identity = F.avg_pool2d(x, 2, 2)
        x = F.relu(self.conv3(x))
        x = F.max_pool2d(x, 2, 2)
        x = torch.cat((x, identity),dim = 1)

        identity = F.avg_pool2d(x, 2, 2)
        x = F.relu(self.conv4(x))
        x = F.max_pool2d(x,2, 2)
        x = torch.cat((x, identity),dim = 1)

        x = F.relu(self.conv5(x))
        x = F.adaptive_avg_pool2d(x,1)
        x = x.reshape(x.size(0), -1)

        x = self.fc1(x)
        return x
```

Рисунок 4.12 – Код класу Net

Наступним кроком дана модель переміщюється на CUDA пристрій. Структура створеної моделі показана на рисунку 4.13.

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(35, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(99, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(227, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(483, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=512, out_features=4, bias=True)
)
```

Рисунок 4.13 – Структура створеної моделі

Найбільш розповсюдженими функціями втрат для задач пошуку об’єктів є функція середньоквадратичної похибки (MSE loss) та функція згладжених втрат

(Smoothed loss). Функція згладжених втрат є менш чутливую до викидів присутніх у даних та запобігає швидке зростання градієнтів. Саме тому використаємо функію SmoothL1Loss з пакету torch.nn з параметром reduction="sum" для того, щоб функція повертала суму похибок міні-пакету даних. Код, який відповідає за визначення даної функції показаний на рисунку 4.14.

```
loss_func = nn.SmoothL1Loss(reduction="sum")
```

Рисунок 4.14 – Визначення функції втрат

Наступним кроком визначимо оптимізатор для автоматичного оновлення параметрів моделі. У якості оптимізатору вибиремо Adam-оптимізатор. Для цього використаємо функцію Adam [43] з пакету optim. Дана функція має два аргументи – параметри, що потребують оновлення у процесі навчання та глобальний параметр - швидкість навчання.

Код, який відповідає за визначення даної функції показаний на рисунку 4.15.

```
opt = optim.Adam(model.parameters(),lr = 1e-3)
```

Рисунок 4.15 – Визначення оптимізатора Adam

Також визначимо функцію (так звану scheduler-функцію), яка б зменшувала швидкість навчання в залежності від поточних обрахунків. Якщо значення, яке повертає функція метрики не змінюється впродовж деякого часу (час задається у вигляді кількості епох параметром patience scheduler-функції), то відбувається зменшення швидкості навчання (ступінь зменшення швидкості навчання задається параметром factor scheduler-функції).

Таким чином, використаємо функцію ReduceLROnPlateau [42] з пакету torch.optim.lr_scheduler.

Код, який відповідає за визначення даної функції показаний на рисунку 4.16.

```
lr_scheduler = ReduceLROnPlateau(opt,mode = 'min',factor = 0.5,patience = 20,verbose = 1)
```

Рисунок 4.16 – Визначення scheduler-функції ReduceLROnPlateau

Тут параметр opt - оптимізатор; mode = 'min' – означає зменшення швидкості навчання тоді, коли значення метрики більше не зменшується; factor =0.5 – ступінь зменшення швидкості навчання (тобто зменшення параметру відбувається вдвічі); patirnce = 20 – задає кількість епох з незмінним значенням метрики, після якого відбувається зменшення швидкості навчання; verbose = 1 – виводить повідомлення про зменшення швидкості навчання у процесі тренування мережі.

Наступним кроком визначимо функцію метрики. Для цього перш за все визначимо допоміжну функцію cxcy2bbox, яка б перетворювала третій формат представлення обмежувальної рамки на другий.

Код даної функції зображений на рисунку 4.17.

```
def cxcy2bbox(cxcy):
    return torch.cat((cxcy[:,:2] - cxcy[:,2:]/2,cxcy[:,:2]+cxcy[:,2:]/2),1)
```

Рисунок 4.17 – Код допоміжної функції cxcy2bbox

Функція метрики metrics_batch обраховує IOU для двох наборів вхідних обмежувальних рамок з міні-пакетів. Один з вхідних наборів обмежувальних рамок визначається в процесі навчання мережі, інший є набором міток (надається для кожного зображення у наборі даних). Для обрахунку IOU використаємо функцію box_iou [44] з пакету torchvision.ops.

Допоміжна функція була використана тому, що більш зручно обраховувати IOU двох обмежувальних рамок тоді, коли вони представлені у другому форматі.

Код функції метрики показаний на рисунку 4.18.

```
import torchvision
def metrics_batch(output, target):
    output = cxcy2bbox(output)
    target = cxcy2bbox(target)
    iou = torchvision.ops.box_iou(output,target)
    return torch.diagonal(iou,0).sum().item()
```

Рисунок 4.18 – Код функції метрики

Для того, аби почати тренуання моделі, створимо функцію train_val та дві допоміжні функції loss_epoch та loss_batch. Функція train_val приймає два параметри – модель model та параметри моделі params_train (що являє собою словник параметрів).

Відповідний код показаний на рисунку 4.19.

```
path2models = "D:\dataSmoke\models"
if not os.path.exists(path2models):
    os.mkdir(path2models)

params_train={
    "num_epochs":500,
    "optimizer":opt,
    "loss_func":loss_func,
    "train_dl":train_dl,
    "val_dl":val_dl,
    "sanity_check":False,
    "lr_scheduler":lr_scheduler,
    "path2weights":path2models+"weights_smoothl7.pt",
}
model,loss_hist,metric_hist=train_val(model,params_train)
```

Рисунок 4.19 – Запуск процесу тренування мережі

Тут словник params_train містить такі параметри: "num_epochs" – кількість епох; "optimizer" – оптимізатор; "loss_func" – функція втрат; "train_dl" – об’єкт-завантажувач тренувальних даних; "val_dl" – об’єкт-завантажувач даних перевірки; "sanity_check" – параметр, що використовується для поверхневої перевірки коректності роботи моделі (якщо "sanity_check" має значення True, то процес тренування відбувається лише для одного міні-пакету даних); "lr_scheduler" – scheduler-функція; "path2weights" – шлях до файлу, у якому зберігаються натреновані в процесі навчання ваги. Попередньо перевіряється чи існує директорія, що міститиме майбутній файл з вагами. Якщо дана директорія не існує, то вона автоматично створюється програмою.

Основна функція, що забезпечує навчання мережі – функція train_val. На початку функції відбувається виділення параметрів тренування зі словника параметрів. Далі створюємо два словники, у яких в процесі тренування

зберігатимемо інформацію про похибку та точність для тренувальних даних та даних перевірки. Під час тренування необхідно зберігати найкращі параметри моделі. Для цього використовуєтсья змінна best_model_wts. Для того, аби відслідковувати поведінку моделі під час навчання, зберігають найкраще значення похибки, яке зберігається у змінній best_loss. Для першої епохи тренування немає попереднього значення похибки, тому best_loss ініціалізується велкиким числом ('inf'). Далі починається цикл, який має таку кількість ітерацій, яка вказана у параметрі "num_epochs" зі словника тренувальних параметрів.

У кожній ітерації циклу відбувається наступне:

- Виводиться поточне значення швидкості навчання.

- Модель переходить у режим тренування за допомогою функції model.train().

- Відбувається тренування мережі шляхом обрахунку значення похибки та оновлення вагів оптимізатором всередині допоміжної функції loss_epoch.

- Модель переходить у режим тестування за допомогою функції model.eval().

- Обчислюється похибка на даних перевірки. У процесі цього градієнти не обчислюються. Після кожної обчисленої похибки порівнюють поточне значення похибки з найкращим значенням похибки (збергіається у змінній best_loss). Якщо отримане значення є кращим за попереднє, то зберігають параметри моделі. Далі обчислену похибку подають на вхід scheduler-функції. Якщо похибка залишається незмнною впродовж 20 епох (20 ітерацій циклу), то швидкість навчання зменшується вдвічі.

Код функції train_val показаний на рисунку 4.20.

Допоміжна функція loss_epoch приймає п’ять аргументів: модель model, функцію втрат loss_func, дані dataset_dl, праметр sanity_check (за замовчуванням False) та опційний параметр opt (оптимізатор). У даній функції наявний цикл, який працює з міні-пакетами даних. (кількість ітерацій циклу дорівнює кількості міні-пакетів). З кожного міні-пакету даних виділяються зображення та мітки. Зображення пропускаються через мережу, у результаті чого на виході отримують

спрогнозовані параметри обмежувальних рамок. Спрогнозовані параметри разом з мітками (точними значеннями параметрів, заданими у наборі даних) подаються на вхід іншої допоміжної функції loss_batch.

Допоміжна функція повертає два значення: похибка та значення, обчислене функцією метрики. Ці значення отримуються на кожній ітерації циклу (для кожного міні-пакету). У результаті функція loss_epoch повератє середнє арифметичне значення похибки та середнє арифметичне значення функції метрики.

Код функції loss_epoch показаний на рисунку 4.21.

```python
def train_val(model,params):
    num_epochs = params["num_epochs"]
    loss_func = params["loss_func"]
    opt = params["optimizer"]
    train_dl=params["train_dl"]
    val_dl = params["val_dl"]
    sanity_check=params["sanity_check"]
    lr_scheduler = params["lr_scheduler"]
    path2weights = params["path2weights"]

    loss_history = {
        "train": [],
        "val":[],
    }
    metric_history={
        "train":[],
        "val":[],
    }
    best_model_wts = copy.deepcopy(model.state_dict())
    best_loss = float('inf')
    for epoch in range(num_epochs):
        current_lr = get_lr(opt)
        print('Epoch {}/{}, current lr = {}'.format(epoch,num_epochs-1,current_lr))
        model.train()
        train_loss,train_metric = loss_epoch(model,loss_func,train_dl,sanity_check,opt)
        loss_history["train"].append(train_loss)
        metric_history["train"].append(train_metric)
        writer.add_scalar("training loss",train_loss,epoch)
        writer.add_scalar("training accuracy",train_metric,epoch)
        model.eval()
        with torch.no_grad():
            val_loss,val_metric = loss_epoch(model,loss_func,val_dl,sanity_check)
            loss_history["val"].append(val_loss)
            metric_history["val"].append(val_metric)
            if val_loss < best_loss:
                best_loss = val_loss
                best_model_wts = copy.deepcopy(model.state_dict())
                torch.save(model.state_dict(),path2weights)
                print("Copied best model weights!")
            lr_scheduler.step(val_loss)
            if current_lr != get_lr(opt):
                print("Loading best model weights!")
                model.load_state_dict(best_model_wts)
            print("train loss:%.6f,accuracy:%.2f"%(train_loss,100*train_metric))
            print("val loss:%.6f,accuracy:%.2f"%(val_loss,100*val_metric))
            print("-"*10)
    model.load_state_dict(best_model_wts)
    return model, loss_history, metric_history
```

Рисунок 4.20 – Код функції train_val

```
def loss_epoch(model,loss_func,dataset_dl,sanity_check=False,opt=None):
    running_loss = 0.0
    running_metric = 0.0
    len_data=len(dataset_dl.dataset)

    for xb,yb in dataset_dl:
        yb = torch.stack(yb,1)
        yb = yb.type(torch.float32).to(device)
        output = model(xb.to(device))
        loss_b,metric_b = loss_batch(loss_func,output,yb,opt)
        running_loss+=loss_b
        if metric_b is not None:
            running_metric+=metric_b
        if sanity_check is True:
            break
    loss = running_loss/float(len_data)
    metric = running_metric/float(len_data)
    return loss, metric
```

Рисунок 4.21 – Код функції loss_epoch

Допоміжна функція loss_batch приймає чотири аргументи: функція втрат loss_func; спрогнозовані параметри обмежувальних рамок output; мітки target, оптимізтор opt (опційний параметр). Дана функція обраховує похибку за допомогою функції loss_func та значення точностіза допомогою функції метрики metrics_batch. Також у функції здійснюється процедура обрахунку градієнтів та оновлення вагів шляхом виконання рядків коду loss.backward() та opt.step().

Рядок коду opt.zero_grad() занулює градієнти, обчислені на попередньому кроці, адже інакше відбувалося б її сумування.

Повний код даної функції показаний на рисунку 4.22.

```
def loss_batch(loss_func,output,target,opt = None):
    loss = loss_func(output,target)
    with torch.no_grad():
        metric_b = metrics_batch(output,target)
    if opt is not None:
        opt.zero_grad()
        loss.backward()
        opt.step()
    return loss.item(),metric_b
```

Рисунок 4.22 – Код допоміжної функції loss_batch

Приклад перших епох навчання зображений на рисунку 4.23.

На рисунку 4.23 видно, що з кожною наступною епохою навчання похибка тренувальних даних та похибка даних перевірки зменшуються. Також відповідні значення точності потроху зростають. Це є індикатором того, що навчання проходить успішно.

```
Epoch 0/499, current lr = 0.001
Copied best model weights!
train loss:0.105098,accuracy:3.08
val loss:0.047463,accuracy:6.63
----------
Epoch 1/499, current lr = 0.001
train loss:0.056138,accuracy:4.87
val loss:0.047909,accuracy:7.05
----------
Epoch 2/499, current lr = 0.001
Copied best model weights!
train loss:0.056165,accuracy:4.95
val loss:0.043882,accuracy:4.55
----------
Epoch 3/499, current lr = 0.001
train loss:0.052649,accuracy:5.25
val loss:0.045202,accuracy:2.85
----------
Epoch 4/499, current lr = 0.001
```

Рисунок 4.23 – Процес навчання мережі

Після закінчення процесу навчання виведемо графіки зміни похибок та значень точності впродовж зміни епох. Дані графіки зображені на рисунках 4.24 та 4.25.

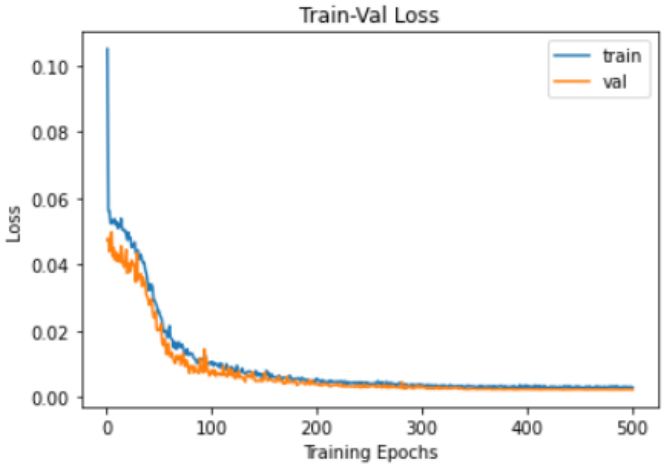


Рисунок 4.24 – Графік зміни похибок

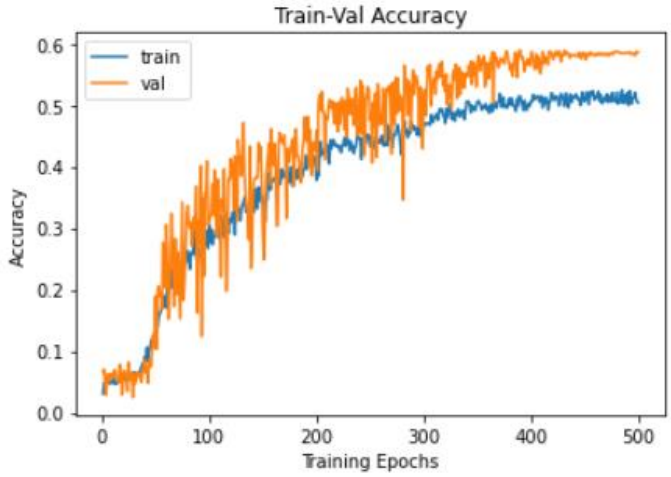


Рисунок 4.25 – Графік зміни значень точності

Завантажимо натреновані ваги до моделі нейронної мережі. Відповідний код показаний на рисунку 4.26.

```
path2weights="D:\dataSmoke\modelsweights_smoothl7.pt"
model.load_state_dict(torch.load(path2weights))
```

Рисунок 4.26 – Завантаження натренованих вагів

Для оцінки роботи мережі виберемо три зображення з набору даних перевірки та за допомогою функції show_tensor_2labels зобразимо дані зображення разом із обмежувальними рамками (заданою рамкою та рамкою спрогнозованою мережею).

Код функції show_tensor_2labels показаний на рисунку 4.27. Три вибрані зображення разом із рамками показані на рисунку 4.28.

```
def show_tensor_2labels(img,label1=None,label2=None,w_h=(100,100)):
    if label1 is not None:
        w1 = label1[2]*img.shape[1:][0]
        h1 = label1[3]*img.shape[1:][1]
        label1 = [label1[0],label1[1]]
        label1=rescale_label(label1,img.shape[1:])
    if label2 is not None:
        w2 = label2[2]*img.shape[1:][0]
        h2 = label2[3]*img.shape[1:][1]
        label2 = [label2[0],label2[1]]
        label2=rescale_label(label2,img.shape[1:])
    img=to_pil_image(img)

    if label1 is not None:
        cx,cy=label1
        draw=ImageDraw.Draw(img)
        draw.rectangle(((cx-w1/2,cy-h1/2),(cx+w1/2,cy+h1/2)),outline="green",width=2)
    if label2 is not None:
        cx,cy = label2
        draw.rectangle(((cx-w2/2,cy-h2/2),(cx+w2/2,cy+h2/2)),outline="red",width=2)

    plt.imshow(np.asarray(img))
```

Рисунок 4.27 – Код функції show_tensor_2labels

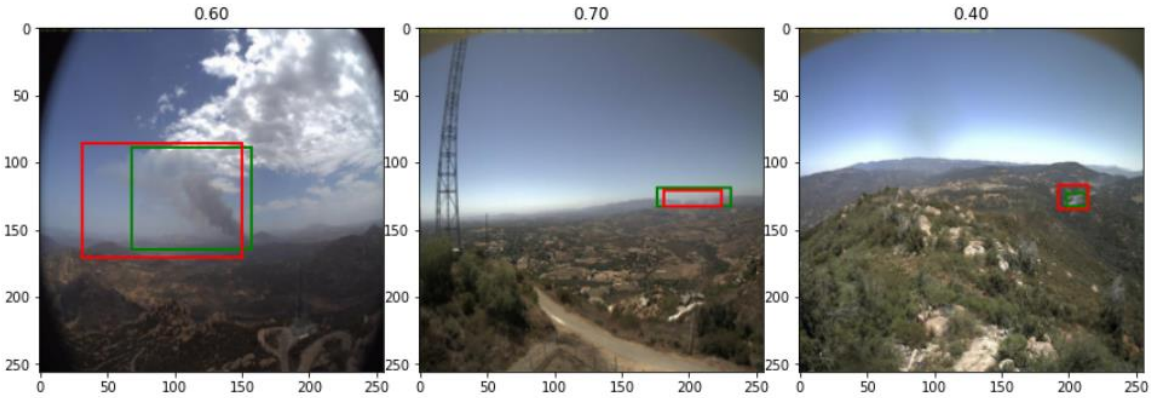
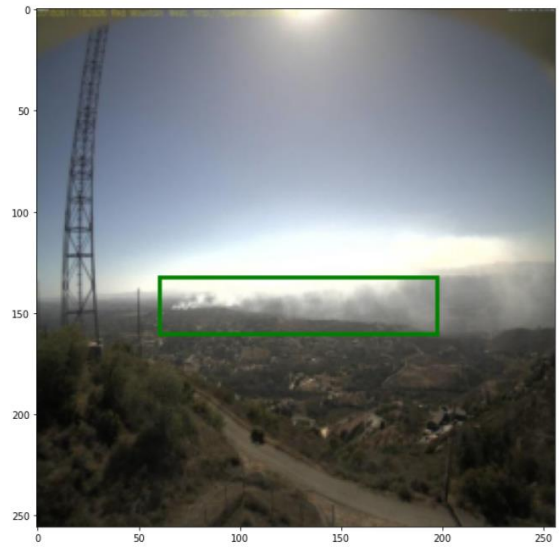
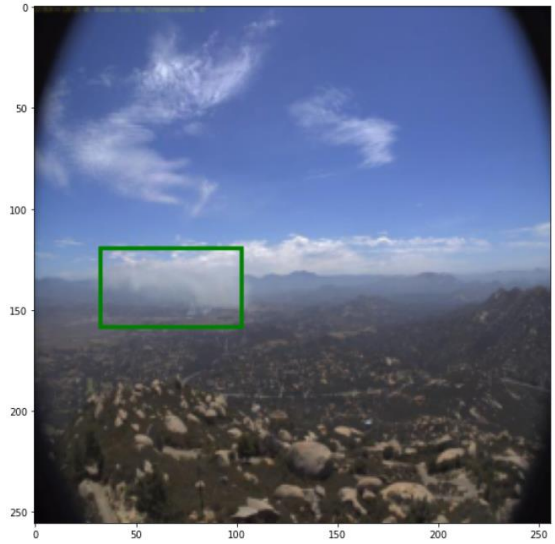


Рисунок 4.28 – Три вибрані зображення разом із рамками

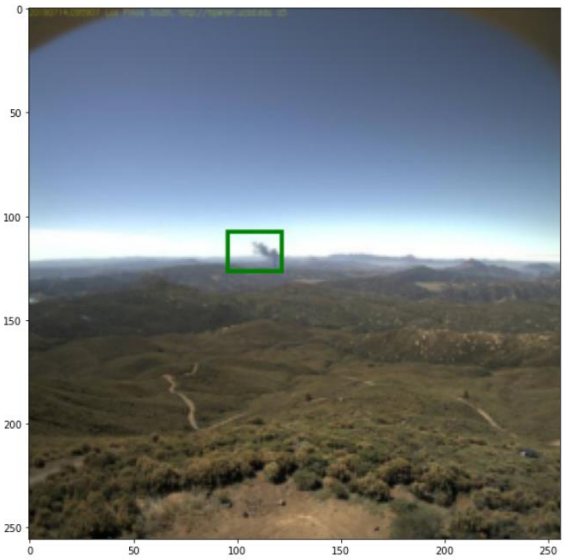
Перевіримо точність роботи програми на тестових даних. Вибрані зображення показані на рисунку 4.29.



а)



б)



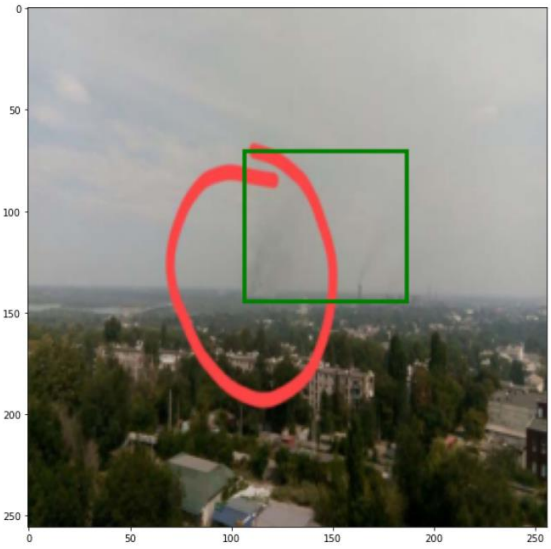
в)

г)

Рисунок 4.29 – Приклад роботи програми на тестових даних: а – перше зображення, б – друге зображення, в – третє зображення, г – четверте зображення

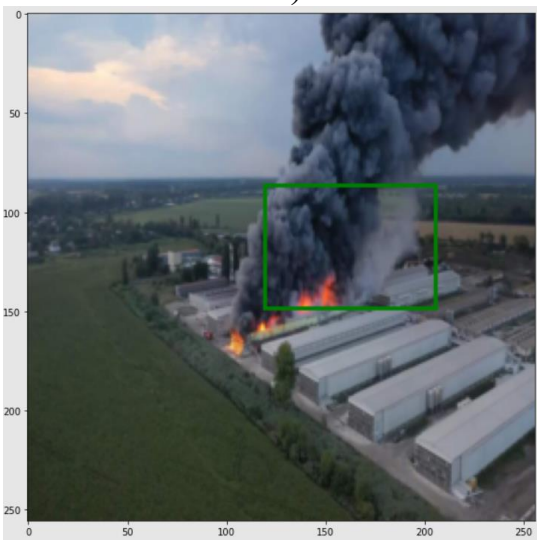
Також візьмемо сторонні зображення для перевірки точності.

Дані зображення разом із прогнозованими обмежувальними рамками показані на рисунку 4.30.

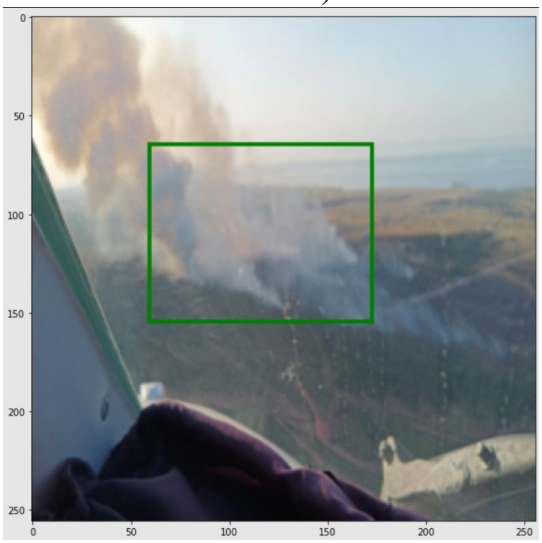


а)

б)



в)



г)

д)

е)

Рисунок 4.30 – Перевірка точності роботи програми на сторонніх зображеннях: а –1-ше зображення, б –2-ге зображення, в –3-тє зображення, г – 4-те зображення, д – 5-те зображення, е – 6-те зображення

ВИСНОВКИ

Наразі технології комп’ютерного зору перебувають на стадії бурхливого розвитку. Їх застосування дозволяє зробити життя та бізнес простішими, дешевшими та безпечнішими. За оцінками експертів цей ринок у найближчі роки буде рухатися лише у напрямку зростання, що і дозволяє розвиватися відповідним технологіям як в сторону продуктивності, так і у сторону якості.

Безумовно, ключову роль серед цих технологій займає технологія Object Detection (пошук об’єктів в зображенні). Саме тому ця робота присвячена саме темі пошуку об’єктів в зображенні.

Відповідно, у роботі розглянуто різні методи та алгоритми пошуку об’єктів в зображенні. Показано, що вся множина методів поділяється на дві категорії:

- методи, що виникли у так званий традиційний період розвитку технології пошуку об’єктів в зображенні, та не використовують машинне навчання;
- методи, що виникли у новий період розвитку, та грунтуються на машинному та глибокому навчанні.

І хоча методи першої категорії є досить точними та ефективними, їх не можна узагальнити для розв’язання широкого класу задач. Власне, у звязку з чим особливої уваги та детального розгляду потребують методи другої категорії, які на даний момент становлять основу технології пошуку об’єктів в зображенні.

Таким чином, ця робота зосереджена на дослідженні методів нейронних мереж розв’язання задачі пошуку об’єктів в зображенні. Було детально розглянуто основні поняття, будову, способи навчання та різні архітектури нейронних мереж. Обгрунтовано, що найбільш доцільним для розв’язання задачі пошуку об’єктів в зображенні є викристання згорткових нейронних мереж, а також архітектур мереж, побудованих на їх основі. Одночасно розглянуто будову та принцип роботи одного з найефективніших сучасних детекторів – детектору YOLO.

З метою розв’язання задачі пошуку об’єктів в зображенні за допомогою нейронних мереж створено програмний продукт з використанням

найсучасніших засобів розробки, що включає модель згорткової нейронної мережі для пошуку одного об’єкту вибраного класу на зображенні (single-object detection).

Під час написання програми були виконані такі кроки:

1) Здійснено приведення вхідних даних до необхідного формату.
2) Здійснено процедуру аугментації даних.
3) Створено модель згорткової нейронної мережі з використанням інструменту Tensorboard, що допомогло перевірити та зрозуміти структуру та продуктивність моделі.
4) Здійснено процедуру навчання мережі з використанням scheduler-функції, що мало суттєвий вплив на якість отриманих результатів.
5) Протетовано результуючу модель на різних типах даних.

Незважаючи на те, що отримані результати є досить точними, можливе покращення точності роботи програми шляхом збільшення кількості вхідних даних, зміни архітектури моделі, впровадження різних технік попередньої обробки даних та регуляризації.

Проте, ця робота має велике практичне значення, адже розроблений програмний продукт може бути використаний для розв’язання складних задач комп’ютерного зору. Програма може бути використана для пошуку одного класу об’єктів на зображенні. Наприклад, метою створеної програми є пошук осередків вогню шляхом виявлення диму. Зазначений програмний продукт може бути використаний для пошуку об’єкту будь-якого класу на вхідному зображенні. Важливою умовою є те, що вхідне зображення має містити лише один об’єкт даного класу. У такому випадку необхідно лише натренувати модель на нових вхідних даних.

Розглянута тема пошуку об’єктів в зображенні за допомогою нейронних мереж потребує подальших досліджень. Для подолання обмеження «одне зображення – один об’єкт» варто працювати над практичною реалізацією детектору YOLO. Одночасно варто розширити тему досліджень, доопрацювати програмний продукт таким чином, щоб за його допомогою було можливе

вирішення задачі пошуку об’єктів на відео, чи навіть з камери у режимі реального часу. У такому випадку використання програмного продукту допоможе у вирішенні найбільш складних та практичних задач комп’ютерного зору.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Shivang Agarwal. Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks [Електронний ресурс] / Shivang Agarwal, Jean Ogier Du Terrail, Frédéric Jurie. – 2019. – Режим доступу до ресурсу: https://hal.archives-ouvertes.fr/hal-01869779v2/document
2. SSD vs. YOLO for Detection of Outdoor Urban Advertising Panels under Multiple Variabilities [Електронний ресурс] / Ángel Morera, Ángel Sánchez, A. Belén Moreno та ін.]. – 2020. – Режим доступу до ресурсу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7472390/.
3. Компьютерное зрение: технологии, рынок, перспективы [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.tadviser.ru/index.php/Статья:Компьютерное_зрение:_технологии,_рынок,_перспективы.
4. Object detection [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Object_detection#:~:text=Object%20detection%20is%20a%20computer,in%20digital%20images%20and%20videos.
5. Компьютерное зрение [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. - Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Компьютерное_зрение
6. Viola–Jones object detection framework [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Viola–Jones_object_detection_framework.
7. https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
8. P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9):1627–1645, Sept 2010.
9. Ross Girshick. Rich feature hierarchies for accurate object detection and semantic segmentation [Електронний ресурс] / Ross Girshick, Jeff Donahue,

Trevor Darrell // 2014 – Режим доступу до ресурсу: https://www.cv-foundation.org/openaccess/content_cvpr_2014/papers/Girshick_Rich_Feature_Hierarchies_2014_CVPR_paper.pdf.

10. Kaiming He. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition [Електронний ресурс] / Kaiming He, Xiangyu Zhang, Shaoqing Ren. – 2015. – Режим доступу до ресурсу: https://arxiv.org/pdf/1406.4729v4.pdf.
11. Rohith Gandhi. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms [Електронний ресурс] / Rohith Gandhi. – 2018. – Режим доступу до ресурсу: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
12. Shaoqing Ren. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [Електронний ресурс] / Shaoqing Ren, Kaiming He, Ross Girshick. – 2016. – Режим доступу до ресурсу: https://arxiv.org/pdf/1506.01497v3.pdf.
13. Joseph Redmon. YOLOv3: An Incremental Improvement [Електронний ресурс] / Joseph Redmon, Ali Farhadi – Режим доступу до ресурсу: https://pjreddie.com/media/files/papers/YOLOv3.pdf.
14. Wei Liu. SSD: Single Shot MultiBox Detector [Електронний ресурс] / Wei Liu, Dragomir Anguelov, Dumitru Erhan. – 2016. – Режим доступу до ресурсу: https://arxiv.org/pdf/1512.02325.pdf.
15. Tsung-Yi Lin. Feature Pyramid Networks for Object Detection [Електронний ресурс] / Tsung-Yi Lin, Piotr Dollar, Ross Girshick. – 2017. – Режим доступу до ресурсу: https://arxiv.org/pdf/1612.03144.pdf.
16. Борисов Е. О задаче поиска объекта на изображении. Часть 1: Базовые методы. [Електронний ресурс] / Евгений Борисов. – 2017. – Режим доступу до ресурсу: http://mechanoid.su/cv-image-detector.html#bor-imgbase.
17. PRIORITY DIRECTIONS OF SCIENCE AND TECHNOLOGY DEVELOPMENT: The 5 th International scientific and practical conference ―Priority directions of

science and technology development‖ (January 24-26, 2021) SPC ―Sciconf.com.ua‖, Kyiv, Ukraine. 2021. 1798 p. [Електронний ресурс] . – Режим доступу до ресурсу: https://sci-conf.com.ua/wp-content/uploads/2021/01/PRIORITY-DIRECTIONS-OF-SCIENCE-AND-TECHNOLOGY-DEVELOPMENT-24-26.01.21.pdf

18. Sebastian Raschka. STAT 479: Machine Learning Lecture Notes [Електронний ресурс] / Sebastian Raschka. – 2018. – Режим доступу до ресурсу: https://sebastianraschka.com/pdf/lecture-notes/stat479fs18/01_ml-overview_notes.pdf.
19. Машинное обучение [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Машинное_обучение
20. S. Raschka, V. Mirjalili. Python Machine Learning: Packt Publishing Ltd., 2019.742p
21. I. Goodfellow, Y. Bengio, A. Courville. Deep Learning. URL: https://www.deeplearningbook.org
22. Линейная регрессия [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Линейная_регрессия
23. I. Vasilev, D. Slater, G. Spacagna, P. Roelants, V. Zocca. Python Deep Learning: Packt Publishing Ltd., 2019.367p
24. Метод опорных векторов [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Метод_опорных_векторов
25. Наивный байесовский классификатор [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Наивный_байесовский_классификатор
26. Теорема Байеса [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Теорема_Байеса

27. k-means clustering [Електронний ресурс]: Вікіпедія. Вільна енциклопедія. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/K-means_clustering
28. Обучение с подкреплением [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Обучение_с_подкреплением
29. Q-обучение [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Q-обучение
30. Шолле Франсуа Ш78 Глубокое обучение на Python. — СПб.: Питер, 2018. — 400 с.: ил. — (Серия «Биб- лиотека программиста»).
31. Глубокое обучение [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Глубокое_обучение
32. Нейронная сеть [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Нейронная_сеть
33. Персептроны [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://neuralnet.info/chapter/персептроны/.
34. SAGAR SHARMA. Activation Functions in Neural Networks [Електронний ресурс] / SAGAR SHARMA. – 2017. – Режим доступу до ресурсу: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.
35. Convolutional Neural Networks (CNNs / ConvNets) [Електронний ресурс] – Режим доступу до ресурсу: https://cs231n.github.io/convolutional-networks/#architectures.
36. Сверточная нейронная сеть [Электронный ресурс]: Википедия. Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Свёрточная_нейронная_сеть
37. Ayoosh Kathuria. How to implement a YOLO (v3) object detector from scratch in PyTorch: Part 1 [Електронний ресурс] / Ayoosh Kathuria. –

2018. – Режим доступу до ресурсу: https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/.

38. Deep Learning — как это работает? Часть 4 [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://habr.com/ru/post/513444/.
39. Wildfire Smoke Dataset. – URL: https://public.roboflow.com/object-detection/wildfire-smoke/1
40. Torch.utils.data. – URL: https://pytorch.org/docs/stable/data.html
41. Лінійна алгебра та аналітична геометрія: Навч. Посібник / В. В. Булдигін, І. В. Алєксєєва, В. О. Гайдей, О. О. Диховичний, Н. Р. Коновалова, Л. Б. Федорова; за ред. проф. В. В. Булдигіна. – К. :ТвіМС, 2011. – 224 с.
42. Torch.nn. – URL: https://pytorch.org/docs/stable/nn.html
43. Torch.optim. – URL: https://pytorch.org/docs/stable/optim.html
44. Torchvision.ops. – URL: https://pytorch.org/vision/stable/ops.html