МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ
ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«___»_____2022 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»

Тема: «Нейронні мережі для розпізнавання об'єктів»

Виконавець: Володимир ПОТАПЕНКО

Керівник: к.т.н., доцент Наталія КІРХАР

Нормоконтролер: к.т.н., доцент Олена ТОЛСТІКОВА

КИЇВ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *кібербезпеки, комп'ютерної та програмної інженерії*
Кафедра *Комп'ютерних інформаційних технологій*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ:
завідувач кафедри КІТ
_____Аліна САВЧЕНКО
(підпис)
«____»_____2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
Потапенка Володимира Сергійовича
(ПІБ випускника)

1. Тема роботи: «Нейронні мережі для розпізнавання об'єктів» затверджена наказом ректора № 1774/ст від 28.09.2022р.

2. Термін виконання роботи: з 26 вересня 2022 року по 27 листопада 2022 року.

3. Вихідні дані до роботи: програмний модуль розпізнавання зображень на основі нейронної мережі *YOLOv*3.

4. Зміст пояснювальної записки: 1. Аналіз проблематики області дослідження. 2. Огляд нейронних мереж та способи їх навчання. 3. Аналіз рішень для розробки системи розпізнавання зображень. 4. Розробка програмного модуля розпізнавання зображень.

5. Перелік обов'язкового ілюстративного матеріалу: 1. Схема проектування нейронної мережі *YOLO*. 2. Графік відношення точності розпізнавання та часу висновку архітектур *CNN*. 3. Схема принципу роботи нейронної мережі *YOLO*.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Аналіз літератури за темою кваліфікаційної роботи	26.09.2022 – 29.09.2022	
2.	Аналіз проблематики області дослідження	01.10.2022 – 05.10.2022	
3.	Провести огляд аналогів програм розпізнавання зображень	06.10.2022 – 10.10.2022	
4.	Огляд нейронних мереж та способів їх навчання	12.09.2022 – 16.09.2022	
5.	Аналіз архітектурних рішень для програмного модуля розпізнавання зображень	18.10.2022 – 20.10.2022	
5.	Розробка програмного модуля розпізнавання зображень	24.10.2022 – 28.10.2022	
6.	Провести тестування програмного модуля розпізнавання зображень	04.11.2022 – 08.11.2022	
8.	Оформити пояснювальну записку	10.11.2022 – 13.11.2022	
9.	Підготувати графічний матеріал	20.11.2022	

7. Дата видачі завдання 26.09.2022р.

Керівник кваліфікаційної роботи Наталія КІРХАР

_____
(підпис керівника)

Завдання прийняв до виконання Володимир ПОТАПЕНКО

_____
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи на тему «Нейронні мережі для розпізнавання об’єктів» містить: 88 сторінок, 38 рисунків, 1 таблиця, 25 інформаційних джерел.

Предмет дослідження – особливості існуючих алгоритмів і методів вирішення завдань комп'ютерного зору, їх доречність та актуальність розглянутої задачі.

Об’єкт дослідження – методи та підходи вирішення завдань комп'ютерного зору.

Тема дослідження – програмний модуль розпізнавання зображень на основі нейронної мережі.

Мета кваліфікаційної роботи – розробити програмний модуль розпізнавання зображень на основі нейронної мережі. Отримати навички розробки програмного забезпечення для розпізнавання зображень на основі нейронних мереж за допомогою програмного середовища *PyCharm*. Дослідити існуючи програмні додатки та сформувати вимоги до власного програмного продукту.

Методи дослідження – теоретично ознайомитися з існуючими програмними аналогами, застосувати можливості бібліотеки *OpenCV* та мови програмування *Python* для створення програмного модуля розпізнавання зображень на основі нейронної мережі.

Вихідні дані кваліфікаційної роботи – на *Python* розроблено програмний модуль розпізнавання зображень на основі нейронної мережі *YOLOv*3. Результати програми пропонуються для використання в роботизованих комплексах, гнучкому автоматизованому виробництві, дронах, безпілотних транспортних засобах та системах безпеки.

НЕЙРОННА МЕРЕЖА, *YOLO, CNN, OPENCV, PYCHARM COMMUNITY, COCO, YOLOV*3, *MAP, ICP*.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Нейронні мережі	–	набір алгоритмів, розроблених на основі спостереження за мозком людини, які служать для розпізнавання шаблонів.
YOLO (You Only Look Once)	–	архітектура згорткових нейромереж, яка накладає на зображення сітку, розділяючи його на осередки.
CNN (Convolutional Neural Network)	–	клас глибинних штучних нейронних мереж прямого поширення.
OpenCV (Open Source Computer Vision Library)	–	бібліотека функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом.
PyCharm Community	–	інтегроване середовище розробки для мови програмування *Python*.
COCO	–	великий набір зображень, призначений для виявлення об'єктів, сегментації, виявлення ключових точок людини, сегментації матеріалів та генерації підписів.
YOLO (You Only Look Once)	–	надзвичайно швидка і точна новіша реалізація архітектури згорткових нейромереж *YOLO*.
mAP (Mean Average Precision)	–	загальноприйнятий спосіб оцінювання змагань з виявлення об'єктів.
ІСР *(Інтегроване середовище розробки)*	–	комплексне програмне рішення для розробки програмного забезпечення.

ВСТУП

З розвитком комп’ютерних технологій багато завдань, що виникають у процесі життя, можна вирішувати, полегшувати, прискорювати та покращуючи якість результатів, таких як робота різноманітних систем життєзабезпечення, взаємодія людини з комп’ютером, поява роботизованих систем. Слід, однак, зазначити, що через недосконаліть розроблених алгоритмів розпізнавання наразі неможливо забезпечити задовільні результати в деяких завданнях (розпізнавання об’єктів, що швидко рухаються, рукописний текст).

Розпізнавання образів - це науковий напрям, що включає розробку принципів і побудову систем, спрямованих на визначення атрибуції даного об'єкту до одного із заздалегідь виділених класів об'єктів.

Завдання пошуку зображення за зразком є частиною (підзадачею) більш загальної задачі розпізнавання образів. При несистематичному і ненаправленному пошуку «схожих» об'єктів з безлічі об'єктів, їх можна перераховувати безкінечно довго і не прийти до завершення із заданою вірогідністю. В окремих випадках об'єкти характеризуються такими ідентифікаційними параметрами (ознаками), як форма, колір, положення, рухливість, по відмінним рисам, їх комбінації. Залежно від цих факторів об'єкти піддаються класифікації. Часто стоїть не глобальне завдання класифікації всіх оточуючих об'єктів, а необхідність виділити в вхідному відео-потоці об'єкти певного роду.

Завдання розпізнавання - це інформаційні завдання, що складаються з двох етапів: перетворення вихідних даних до виду, зручного для розпізнавання; власне розпізнавання (вказівка приналежності об'єкта певного класу).

У цих завданнях можна вводити поняття аналогії або подібності об'єктів і формулювати правила, на підставі яких об'єкт зараховується в один і той же клас або в різні класи. Також можна оперувати набором прецедентів-прикладів, класифікація яких відома і які у вигляді формалізованих описів можуть бути пред'явлені алгоритмом розпізнавання для настройки на

завдання в процесі навчання. Для цих завдань важко будувати формальні теорії і застосовувати класичні математичні методи (часто недоступна інформація для точної математичної моделі або виграш від використання моделі та математичних методів непорівнянний з витратами).

Виділяють такі типи завдань розпізнавання:

1) Задача розпізнавання - віднесення пред'явленого об'єкта за його опису до одного із заданих класів (навчання).

2) Задача автоматичної класифікації - розбиття множини об'єктів, ситуацій, явищ за їх описами на систему непересічних класів (таксономія, кластерний аналіз, самонавчання).

3) Задача вибору інформативного набору ознак при розпізнаванні.

4) Задача приведення вихідних даних до виду, зручного для розпізнавання.

5) Динамічне розпізнавання і динамічна класифікація - завдання 1 і 2 для динамічних об'єктів.

6) Задача прогнозування - попередній тип, в якому рішення повинне ставитися до деякого моменту в майбутньому.

У процесі біологічної еволюції багато тварин за допомогою зорового й слухового апарата вирішили задачу розпізнавання образів досить добре. Створення штучних систем розпізнавання образів залишається складною теоретичною й технічною проблемою. Необхідність у такому розпізнаванні виникає в самих різних областях - від військової справи й систем безпеки до оцифровування різних аналогових сигналів.

Традиційно задачі розпізнавання образів включають у коло задач штучного інтелекту.

Розпізнавання образів, науковий напрям, пов'язаний з розробкою принципів і побудовою систем, призначених для визначення приналежності даного об'єкту до одного із заздалегідь виділених класів. Під об’єктами в розпізнаванні образів розуміються різні об’єкти, явища, процеси, ситуації та сигнали. Кожен об’єкт описується набором основних ознак (символів, атрибутів) $X=(x_1,...,x_i,...,x_n)$, де i-та координата вектора X визначає i-ту ознаку,

а додаткова ознака *S* вказує на приналежність об'єкта до певного класу (зображення). Набір попередньо класифікованих об’єктів, де відомі ознаки *X* і *S*, використовуються для ідентифікації регулярних зв’язків між значеннями цих ознак, звідси й назва навчальних зразків. Ті об'єкти, характеристика *S* яких невідома, утворюють контрольну вибірку. Окремі об'єкти навчальної та контрольної вибірок називаються реалізаціями. Однією з основних задач розпізнавання образів є вибір правила (вирішальної функції) *D*, згідно з яким за значенням реалізації керування *X* визначається його приналежність до одного з шаблонів, а «найбільш обґрунтована» значення представляє характеристику *S* цього *X*.

Успіх вирішення завдань розпізнавання образів значною мірою залежить від того, наскільки правильно підібрані ознаки *X*.

Початковий набір функцій зазвичай дуже великий. У той же час прийнятні правила повинні базуватися на використанні кількох особливостей, які є найважливішими для відмінності одного методу від іншого. Тому в задачі медичної діагностики важливо визначити, які симптоми та їх поєднання (синдроми) слід використовувати при діагностиці того чи іншого захворювання. Тому проблема виділення інформативних ознак є важливою частиною задачі розпізнавання образів.

Проблема розпізнавання образів тісно пов'язана із завданням попередньої класифікації або класифікації.

В основному завданні розпізнавання образів функція рішення *D* будується з використанням канонічного зв’язку між ознаками *X* і *S*, знайденими на навчальних зразках, разом із деякими додатковими попередніми припущеннями, такими як:

Характеристики *X* реалізації зображення — це випадкові вибірки з генеральної сукупності з нормальним розподілом, реалізації методу «компактно» розташовані (у певному сенсі), ознаки в наборі *X* є незалежними тощо.

У сфері розпізнавання образів активно використовуються ідеї та результати багатьох інших. Наукові галузі – математика, кібернетика, психологія та ін.

З розвитком електронного обладнання в 1960-х роках широко використовувалися системи автоматичної ідентифікації. Ідентифікація системи зазвичай означає поєднання засобів, призначених для вирішення вищезазначених завдань. Методи розпізнавання образів використовуються в процесі машинної діагностики різних захворювань, прогнозування корисних копалин в геології, аналізі економічних і соціальних процесів, психології, криміналістиці, лінгвістиці, океанографії, хімії, ядерної та космічної фізики, автоматизованих системах управління та ін. є виправданим майже скрізь, де потрібно мати справу з класифікацією експериментальних даних.

Об’єктом дослідження є автоматичний метод розпізнавання зображень на основі нейронної мережі.

Тема дослідження – програмний модуль розпізнавання зображень на основі нейронної мережі.

Мета роботи – розробка програмного забезпечення для розпізнавання зображень на основі нейронних мереж за допомогою *IDE* спільноти *PyCharm*. Реалізовувати у модулі такі функції: детектування образів на завантажених графічних зображень; детектування образів на завантажених відео файлах; детектування образів на зображенні, отриманому з веб-камери; збереження результатів; можливість переключитись на інші версії нейромережі *YOLO* (*YOLOv*2, *TinyYOLOv*2, *YOLOv*3, *TinyYOLOv*3).

Методом дослідження є теоретичне ознайомлення з існуючими програмними аналогами, застосування можливостей бібліотеки *OpenCV* та мови програмування *Python* для створення програмних модулів розпізнавання зображень на основі нейронної мережі.

Практичне значення отриманих результатів полягає в подальшому розвитку можливостей використання створених засобів у робототехнічних комплексах, гнучких автоматизованих виробництвах, безпілотних літальних апаратах, безпілотних апаратах та системах безпеки.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМ У ГАЛУЗІ ДОСЛІДЖЕНЬ

1.1. Сучасний програмний комплекс для розпізнавання образів

Сферу комп'ютерного зору можна охарактеризувати як молоду і різноманітну. І хоча є більш ранні роботи, можна сперечатися, що цю проблему почали глибоко вивчати лише наприкінці 1970-х років, коли комп’ютери змогли обробляти великі масиви даних, такі як зображення. Однак ці дослідження зазвичай починаються з інших областей, тому не існує стандартної формули для проблем комп’ютерного зору. Крім того, і що більш важливо, не існує стандартного формулювання того, як вирішити проблеми комп’ютерного зору. Натомість існує кілька підходів до вирішення різноманітних чітко визначених проблем комп’ютерного зору, які часто стосуються конкретних завдань і рідко поширюються на широкі програми [1].

Багато методів і застосувань все ще знаходяться на стадії фундаментальних досліджень, але все більше і більше методів використовуються в комерційних продуктах, і вони часто є частиною складних систем, які можуть вирішувати складні проблеми (наприклад, у сфері медицини), зображення або вимірювання під час виробництво та контроль якості). У більшості практичних застосувань комп’ютерного зору комп’ютери попередньо запрограмовані для вирішення конкретних проблем, але підходи, засновані на знаннях, стають все більш поширеними.

Комп’ютерний зір — це галузь досліджень, спрямована на те, щоб допомогти комп’ютерам побачити проблеми. Це мультидисциплінарна галузь, яку можна загалом назвати підгалуззю штучного інтелекту та машинного навчання, яка може передбачати використання спеціалізованих методів і використання загальних алгоритмів навчання [2].

У більшості випадків, коли людина сприймає явища навколишнього світу, вона категоризує їх, тобто поділяє ці явища (предмети, ситуації) на групи схожих явищ (абсолютно схожих, не ідентичних). Чомусь виникає необхідність включити в групу явища або предмети, які в чомусь схожі, але можуть істотно відрізнятися один від одного.

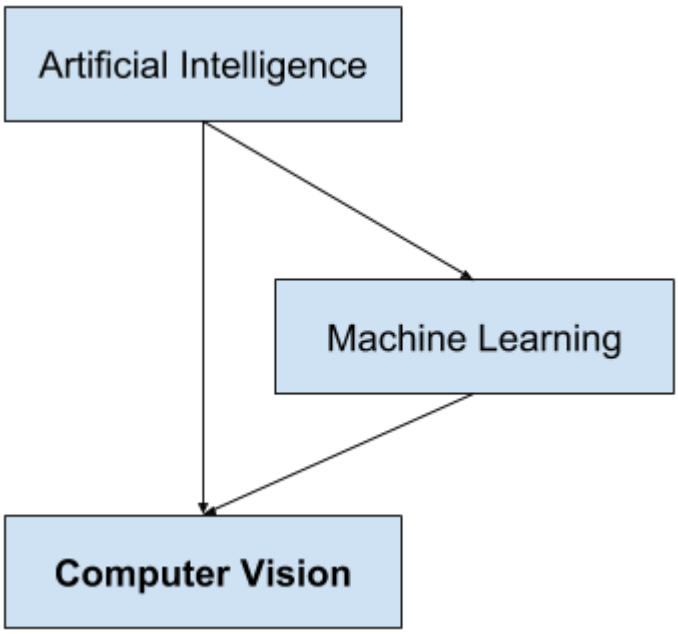

Рис. 1.1. Огляд зв’язку між штучним інтелектом і комп’ютерним зором

Для оптичного розпізнавання зображень можна застосовувати методи сортування зовнішнього вигляду об’єктів під різними кутами, масштабами та зміщеннями. Для літер, шрифтів, властивостей шрифту потрібно відсортувати.

Другий спосіб - знайти контур предмета і перевірити його властивості (зв'язність, наявність кутів).

Інший підхід полягає у використанні штучних нейронних мереж (багатошарові перцептрони, мережі квантування, карти Кохонена, рекурентні мережі). Такий підхід вимагає або великої кількості прикладів завдань на розпізнавання (з правильними відповідями), або спеціальної структури нейронної мережі, яка враховує деталі цього завдання.

Реалізація системи комп'ютерного зору багато в чому залежить від сфери її застосування. Деякі системи є автономними та вирішують конкретні проблеми виявлення та вимірювання, тоді як інші системи є підсистемами більших систем, які, наприклад, можуть містити підсистеми для керування механічними маніпуляторами, планування, інформаційні бази даних, людино-машинні інтерфейси тощо.

Реалізація системи комп’ютерного зору також залежить від того, чи є її функція наперед визначеною, чи деякі її частини можна вивчати та змінювати під час роботи. Однак багато систем комп’ютерного зору мають деякі спільні характеристики.Отримання зображень: цифрові зображення отримують з одного або кількох датчиків зображення, включаючи датчики дальності, радари, ультразвукові камери тощо, на додаток до різних типів фоточутливих камер. Залежно від типу датчика отримані дані можуть являти собою звичайні 2*D*-зображення, 3*D*-зображення або послідовності зображень. Значення пікселів зазвичай відповідають інтенсивності світла в одній або кількох спектральних смугах (кольорові або сірі зображення), але можуть бути пов’язані з різними фізичними вимірюваннями, такими як глибина, поглинання або відображення звукових чи електромагнітних хвиль або ядерний магнітний резонанс.

Попередня обробка: перед застосуванням методів комп’ютерного зору до відеоданих для отримання певної інформації, відеодані мають бути оброблені відповідно до певних вимог відповідно до використовуваного методу. Наприклад:

– перевибірка, щоб переконатися, що система координат зображення правильна;

– шумозаглушення для усунення спотворень, внесених датчиком;

– збільште контраст, щоб можна було виявити необхідну інформацію;

– збільште, щоб краще розрізняти структуру на зображенні;

– вилучення деталей: вилучення деталей зображення різної складності з відеоданих. Типовими прикладами таких деталей є: лінії та межі.

Місцеві об’єкти інтересу, такі як кути, плями або точки: складніші деталі можуть належати до структури, форми чи руху.

Виявлення або сегментація: на певному етапі обробки рішення про те, які точки чи області зображення важливі для подальшої обробки. Приклади:

– вибрати набір цікавих для нас точок;

– сегментуйте одну або кілька областей зображення, які містять характерні об’єкти.

Розширена обробка: на цьому етапі вхідними даними зазвичай є невеликий набір даних, наприклад набір точок або область на зображенні, якою вважається об’єкт. Приклади:

– перевірте, чи дані відповідають умовам, які залежать від методу та застосування;

– оцінка характерних параметрів, таких як розташування або розмір об'єкта;

Класифікуйте знайдені предмети за різними категоріями. Відомі такі програмні продукти для розпізнавання зображень:

– *CuneiForm* - це інструмент оптичного розпізнавання символів, розроблений російською компанією *Cognitive Technologies*. Програма перетворює файли зображень, отримані зі сканера або іншим способом, у текст;

– *FineReader* - програма оптичного розпізнавання символів, розроблена російською компанією *ABBYY*;

– *Readiris* — програма для конвертації документів у різні формати. Сканер або МФУ дозволяє сканувати та розпізнавати текст за допомогою цієї утиліти.

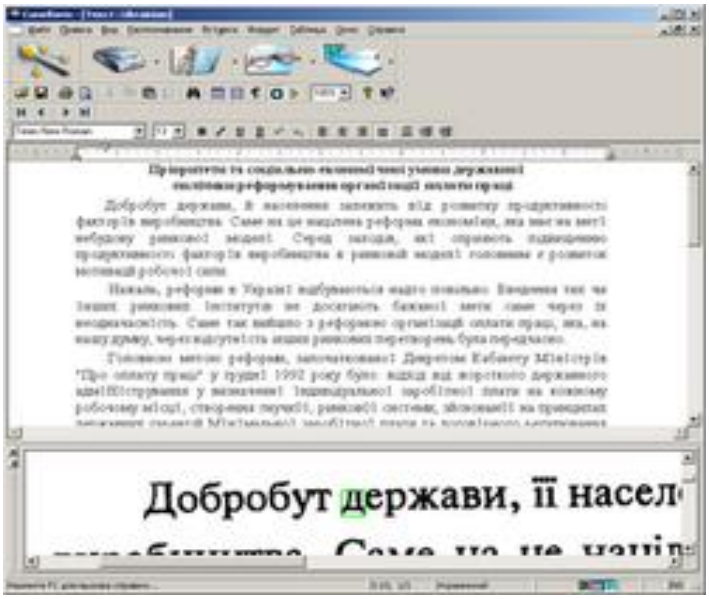

Рис. 1.2. Головне вікно програми *CueiForm*

CuneiForm — це система, що не залежить від шрифтів (*OmniFont*). Алгоритми, вбудовані в *CuneiForm*, базуються на правилах написання літер на основі їх топології та не вимагають встановлення певних критеріїв чи навчання. Розпізнається будь-який шрифт - книги, газети, журнали, роздруківки з лазерних і матричних принтерів, текст з друкарських машинок і т.д. Рукописний текст і декоративні шрифти (готика, рукописний стиль) не розпізнаються. У *CuneiForm* є спеціальні налаштування для розпізнавання тексту з матричних принтерів і факсів 200*x*100 *DPI*. *CuneiForm* зберігає форматування тексту та розпізнає складні форми довільної структури.

Програма розпізнає англійську, болгарську, голландську, датську, естонську, іспанську, італійську, латвійську, литовську, німецьку, польську, португальську, російську, румунську, сербську, словенську, турецьку. Текст англійською, угорською, українською, французькою, хорватською, чеською, шведською, російською та англійською.

Рис. 1.3. Головне вікно програми *FineReader*

FineReader швидко й точно розпізнає відскановані або сфотографовані документи та перетворює їх у редаговані електронні формати або *PDF*-файли з можливістю пошуку. Швидкий режим збільшує швидкість на 40% без шкоди для точності під час розпізнавання високоякісних документів. Для чорно-білих документів також можна використовувати чорно-білий режим розпізнавання, а швидкість роботи збільшується на 30%. *FineReader* зберігає оригінальну структуру багатосторінкових документів, включаючи розташування тексту, таблиць, колонтитулів, коментарів, номерів сторінок, змісту, змісту тощо. *FineReader* забезпечує миттєвий доступ до відсканованих сторінок документа, незалежно від їх розміру. *FineReader* може розпізнавати документи будь-якою комбінацією з 190 мов. Програма перетворює зображення документів і *PDF*-файли, отримані зі сканера (без текстового шару), у формат, придатний для збереження в електронних архівах і доступний для пошуку: *PDF* з текстовим шаром або *PDF/A*. Програма підтримує декілька форматів для збереження документів, які потрібні вам на роботі. Ви можете зберегти результати розпізнавання у файл або миттєво надіслати їх до таких програм, як *Microsoft Word, Excel, PowerPoint, OpenOffice Writer* тощо. Програма підтримує збереження електронних книг у найпопулярніших форматах, що допоможе швидко

робити електронні копії для портативних пристроїв (електронних книг, планшетів, смартфонів тощо).

Readiris — це продукт для розпізнавання тексту з функціями для окремих користувачів і фрілансерів. *Readiris* економить час, усуваючи необхідність повторно друкувати інформацію в необхідних документах. *Readiris* автоматично розпізнає скановані копії зображень, тексту, файлів *PDF* або документів і перетворює їх у цифровий формат, який можна редагувати.

1.2. Застосування нейронної мережі в розпізнаванні образів

1.2.1. Визначення нейронної мережі

Нейронні мережі — це набір алгоритмів, розроблених на основі спостережень людського мозку для розпізнавання закономірностей. Вони інтерпретують сенсорні дані за допомогою машинного сприйняття, маркування або групування необроблених даних. Шаблони, які вони розпізнають, є числовими й містяться у векторах, у які мають бути перетворені всі дані реального світу, чи то зображення, звуки, текст чи часові ряди. На рис. 1.4 зображено принципову схему нейрона [3].

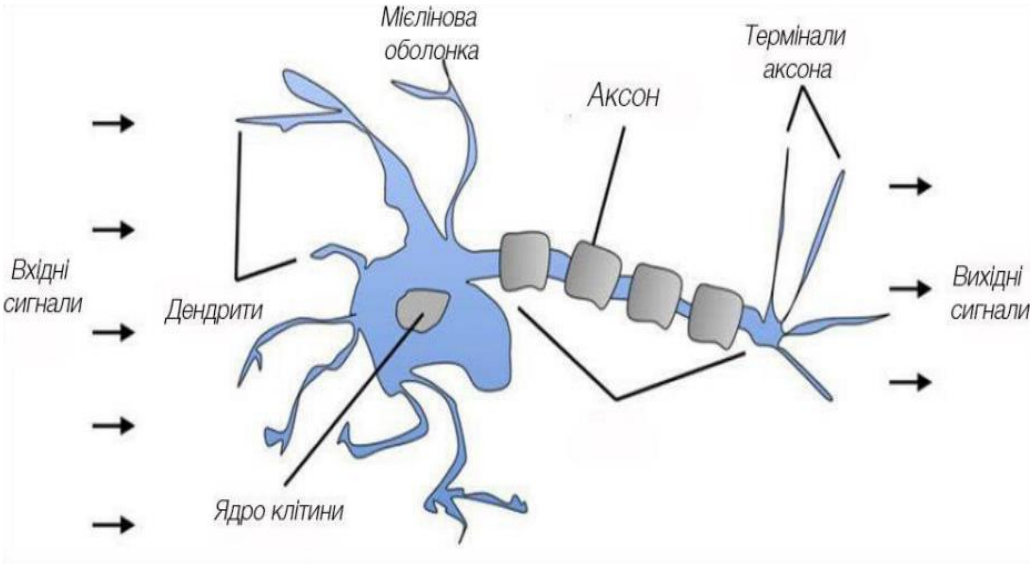

Рис. 1.4. Будова нейрона

Нейронні мережі допомагають нам кластеризувати та класифікувати дані. Їх можна розглядати як рівні кластеризації та класифікації поверх даних, які ви зберігаєте та керуєте ними. Вони допомагають групувати немарковані дані за подібністю між вхідними даними програми та класифікувати їх, якщо вони мають позначений навчальний набір даних. Нейронні мережі також можуть витягувати функції для використання в інших алгоритмах кластеризації та класифікації; таким чином, глибокі нейронні мережі можна вважати компонентами більших програм машинного навчання, включаючи навчання з підкріпленням, класифікацію та регресійні алгоритми [3].

Усі завдання класифікації покладаються на мічені набори даних, що означає, що люди повинні передавати свої знання в набір даних, щоб нейронна мережа могла дізнатися про зв’язок між мітками та даними. Це називається навчання під наглядом:

– виявляти обличчя, ідентифікувати людей на зображеннях, визначати вирази обличчя (злий, щасливий);

– впізнавати об’єкти на фотографіях (знаки зупинки, пішоходів, розмітку смуг);

– розпізнавати жести у відео;

– виявляти голоси, визначати, хто говорить, транскрибувати розмови в текст і визначати почуття в голосах;

– класифікуйте текст як спам (в електронних листах) або шахрайський (у страхових претензіях).

Будь-яку мітку, яку може згенерувати людина, будь-який результат, який вас цікавить і пов’язаний з даними, можна використовувати для навчання нейронної мережі. Завдяки класифікації глибоке навчання може встановлювати зв’язки між пікселями на зображенні та іменами людей. Це можна назвати статичним прогнозуванням. Так само глибоке навчання здатне встановлювати кореляції між поточними та майбутніми подіями за

наявності достатньої кількості даних. Між минулим і майбутнім можна здійснити регресію. Майбутня подія в певному сенсі схожа на маркер.

– вихід з ладу обладнання (центр обробки даних, виробництво, транспортування);

– розлади здоров'я (інсульт, інфаркт на основі життєво важливих статистичних даних і носимих даних);

– замовлення клієнтів (прогнозування ймовірності відходу клієнта на основі мережевої активності та метаданих);

Важка праця робітників.Іншим основним завданням нейронних мереж є завдання кластеризації. Кластеризація або групування – це ідентифікація подібності. Для глибокого навчання не потрібні мітки для виявлення подібності. Навчання без міток називається навчанням без нагляду. Дані без міток становлять основну частину даних у світі. Закон машинного навчання полягає в тому, що чим більше даних може працювати алгоритм, тим точнішим він буде. Таким чином, неконтрольоване навчання має потенціал для створення високоточних моделей. Пошук: Порівняйте документи, зображення чи звуки з елементами, які зовні схожі. Виявлення аномалій: розгортання виявлення подібності – виявлення аномалій або незвичної поведінки. У багатьох випадках ненормальна поведінка тісно пов’язана з чимось, що потрібно виявити та запобігти, наприклад шахрайству.

1.2.2. Елементи нейронних мереж

Глибоке навчання — це назва, яка використовується для складних нейронних мереж, тобто мереж, що складаються з кількох шарів. Шари складаються з вузлів. Вузол — це місце, де відбуваються обчислення, імітуючи нейрон у людському мозку, який спрацьовує, коли його достатньо стимулюють. Вузол поєднує вхідні дані з набором коефіцієнтів або ваг, які можуть збільшувати або зменшувати ці вхідні дані, тим самим призначаючи вхідним даним значення, що відповідає завданню, яке алгоритм намагається вивчити. Наприклад, шляхом вибору класифікації, які вхідні дані є найбільш корисними. Ці вхідні зважені продукти підсумовуються, а потім

пропускаються через так звану функцію активації вузла, щоб визначити, чи має сигнал поширюватися далі в мережі, щоб вплинути на кінцевий результат, і в якій мірі. Якщо сигнал проходить, нейрон активовано. На рис. 1.4 зображено принципову схему нейрона.

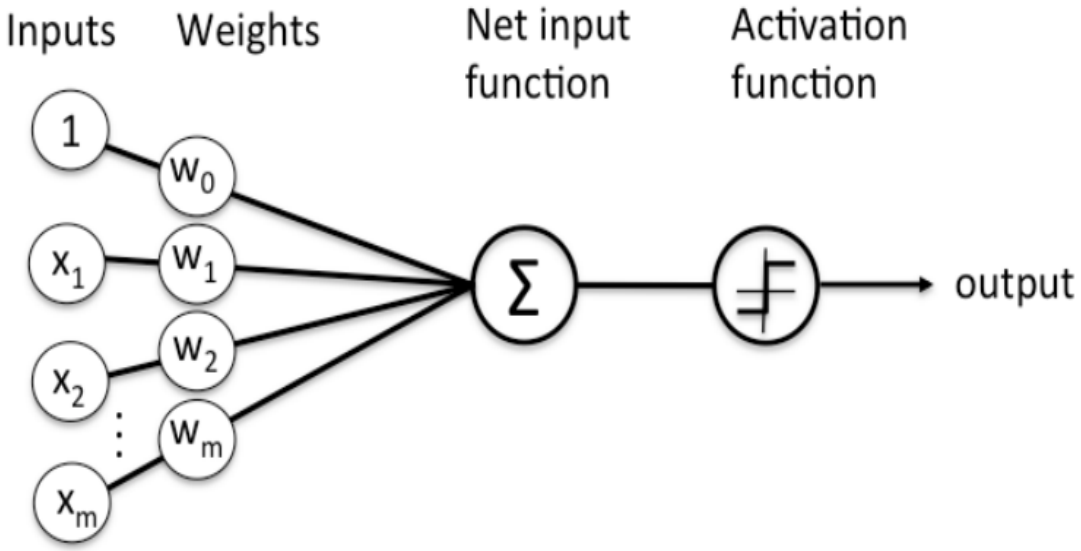

Рис. 1.5. Зображення діаграми вузла

Вузловий рівень — це серія нейронних перемикачів, які вмикаються або вимикаються, коли вхід надходить через мережу. Вихідні дані кожного шару також є вхідними даними для наступного рівня, починаючи з оригінального вхідного рівня, який отримав дані на рис. 1.5 показаний приклад простої нейронної мережі

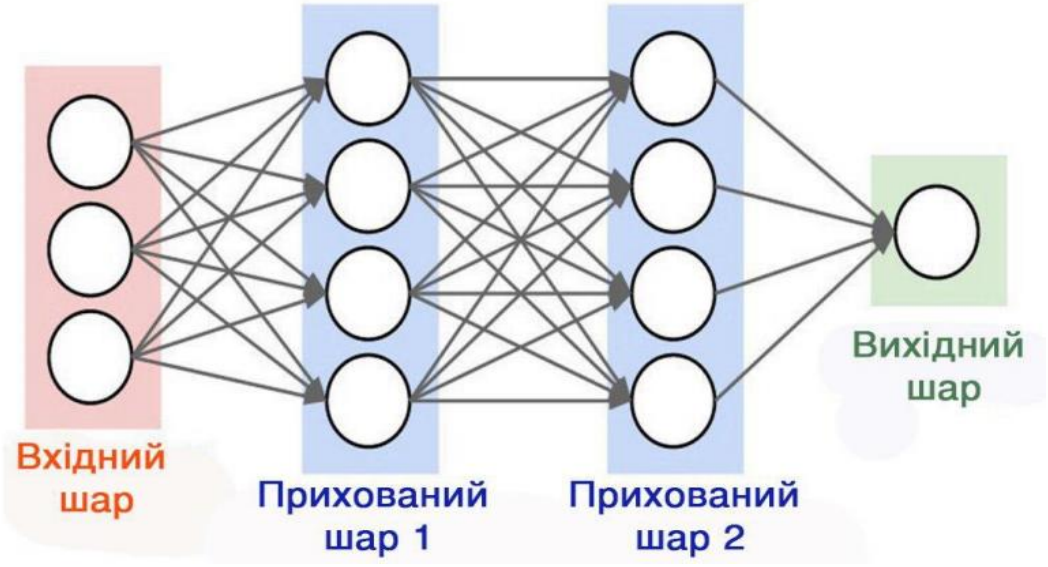

Рис. 1.6. Простий приклад нейронної мережі

Поєднання регульованих вагових коефіцієнтів моделі з вхідними функціями – це те, як ми призначаємо значення цим функціям на основі того, як нейронна мережа класифікує та кластеризує.

1.2.3. Основні поняття глибоких нейронних мереж

Мережі глибокого навчання відрізняються від більш поширених одношарових нейронних мереж глибиною, тобто кількістю шарів вузлів, через які мають проходити дані під час багатоетапного процесу розпізнавання образів.

Ранні версії нейронних мереж були поверхневими, складалися з вхідного рівня та вихідного рівня з принаймні одним прихованим шаром між ними. Більш ніж три рівні (включно з вхідним і вихідним) називають глибоким навчанням, що означає, що глибина — це термін, який означає більше одного прихованого шару.

Глибоке навчання (також відоме як глибоке структуроване навчання, ієрархічне навчання або глибоке машинне навчання) — це розділ машинного навчання, заснований на наборі алгоритмів, які намагаються змоделювати високорівневі абстракції даних. У простому прикладі ви можете мати два набори нейронів: один, який отримує вхідні сигнали, і інший, який надсилає вихідні сигнали. Коли вхідний рівень отримує вхідні дані, він передає

змінену версію вхідних даних на наступний рівень. У глибоких мережах існує багато рівнів між входом і виходом (ці шари не складаються з нейронів, але було б корисно думати про це саме так), що дозволяє алгоритму використовувати кілька рівнів обробки, що складаються з кількох лінійних і нелінійні перетворення

У мережі глибокого навчання кожен рівень вузлів навчається окремому набору функцій на основі результатів попереднього рівня. Чим глибше ви заходите в нейронну мережу, тим складніші функції можуть розпізнавати ваші вузли під час агрегування та реорганізації функцій попереднього рівня.

Це дозволяє мережам глибокого навчання обробляти дуже великі набори даних із мільярдами параметрів, що передаються через нелінійні функції. На рис. 1.6 показано дедалі складнішу та абстрактнішу ієрархію функцій [4].

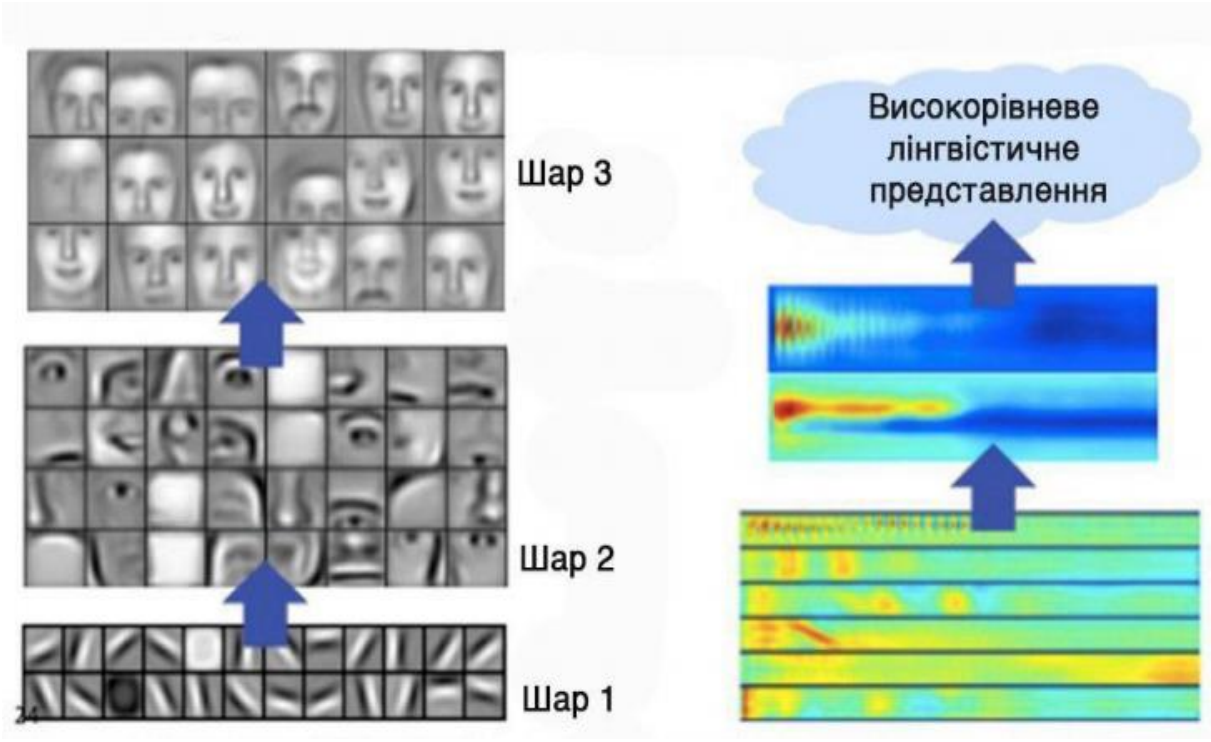

Рис. 1.7. Ієрархія функцій

По-перше, ці нейронні мережі здатні виявляти приховані структури в неструктурованих даних, які становлять переважну більшість даних у світі. Іншими словами, неструктуровані дані - це необроблені носії, тобто

зображення, текст, відео та аудіозаписи. Таким чином, однією з проблем, яку найкраще може вирішити глибоке навчання, є обробка та кластеризація глобальних, немаркованих носіїв, а також виявлення подібностей і аномалій у даних.

Наприклад, глибоке навчання може взяти мільйон зображень і згрупувати їх на основі їх схожості: кіт в одному кутку, криголам в іншому. Це основа того, що називається розумним фотоальбомом.

Глибоке навчання може кластирувати необроблений текст, наприклад електронні листи чи новинні статті. Електронні листи, повні гнівних скарг, можуть накопичуватися в кутку вектора простір, тоді як звіти від задоволених клієнтів або спам-сайтів можуть бути згруповані на інших сайтах. Це основа для різноманітних фільтрів повідомлень, які можна використовувати для управління взаємовідносинами з клієнтами. Те саме стосується голосових повідомлень.

З часом дані можуть бути згруповані навколо нормальної та ненормальної поведінки. Якщо дані часових рядів генеруються смартфонами, вони дадуть зрозуміти здоров’я та звички користувачів; якщо вони генеруються автомобільними деталями, їх можна використовувати для запобігання катастрофічним збоям.

На відміну від більшості традиційних алгоритмів машинного навчання, мережі глибокого навчання виконують автоматичне виділення ознак без втручання людини. Враховуючи те, що виділення функцій — це завдання, на виконання якого у команди вчених потрібні роки, глибоке навчання — це спосіб обійти обмежену точку зору експертів. Це збільшує потужність невеликих наукових груп, які за своєю суттю не мають масштабу.

Під час навчання кожен рівень вузлів у глибокій мережі автоматично вивчає функції, багаторазово намагаючись реконструювати вхідні дані, з яких були взяті зразки, намагаючись мінімізувати різницю між припущеннями мережі та розподілом ймовірностей самого вхідного сигналу [4].

У процесі нейронні мережі навчаються виявляти кореляції між певними релевантними функціями та найкращими результатами – вони встановлюють зв’язок між сигналами ознак і тим, що ці характеристики представляють, повністю реконструйовані чи позначені дані.

Мережа глибокого навчання закінчується вихідним рівнем, який є класифікатором, який призначає ймовірності конкретним результатам або міткам. Ви можете назвати це прогнозними даними. Наприклад, на основі необроблених даних зображення мережа глибокого навчання може визначити, що з імовірністю 90% вхідні дані представляють людину.

1.3. Метод розпізнавання образів

Для сприйняття зображення необхідно вибрати набір релевантних інформативних ознак і, вимірявши їх значення, сформувати векторну реалізацію зображення. Тим часом на практиці зазвичай потрібні додаткові процедури для попередньої обробки, фільтрації та відновлення зображень.

Попередня інформація об’єкта розпізнавання необхідна для формування алфавіту категорій розпізнавання, словника ознак і матриці навчання та відіграє важливу роль у процесі вирішення проблеми. Крім того, для встановлення високонадійних правил прийняття рішень на етапі формування вхідного математичного опису системи розпізнавання необхідний так званий інтелектуальний аналіз, щоб гарантувати статистичну стабільність і однорідність реалізації зображення [5].

Методи ідентифікації в основному поділяються на дві категорії:

1) Метод використання класифікованої освітньої матриці для формулювання вирішальних правил (метод навчання з учителем).

2) Автоматичні методи класифікації, здатні обробляти несекретні дані (методи навчання без викладача).

У кожній із цих основних груп методи далі поділяються на підгрупи відповідно до попередньої інформації, необхідної для їх застосування.

Розглянемо узагальнену постановку задачі синтезу інформації для навчальної системи розпізнавання образів у рамках геометричних методів [6].

Нехай дано алфавіт класів розпізнавання

$\{Xmo|m = 1. .. M\}$, де M – ступінь літери.

На етапі навчання необхідно:

1) Побудувати R-розбиття простору ознак $\Omega|N|$ якимось оптимальним способом (тут і далі в інформативному сенсі) щодо ідентифікації класів.

2) Геометричні параметри R^* оптимально розподіляються на класи розпізнавання відповідно до побудованого простору ознак, а безпомилкові правила прийняття рішень будуються відповідно до навчальної матриці.

На етапі перевірки, тобто безпосереднього розпізнавання, приймається високонадійне рішення щодо того, чи належить розпізнана реалізація зображення до одного з класів даного алфавіту.

Таким чином, у топологічному сенсі ідентифікаційний клас ідентифікує область у просторі ознак, елементи якої еквівалентно пов’язані.

Щоб вирішити, чи належить розпізнане зображення до одного з класів даного алфавіту, необхідне детерміноване правило. Їх можна побудувати за допомогою математичних формул, геометричних об’єктів, мовних структур тощо.

Розглянемо етап встановлення вирішальних правил у рамках геометричних методів, що характеризується найбільшою узагальненістю та наочністю:

1) Вимірювання ідентифікаційних ознак.

2) Нормалізація значень ознак ідентифікації, включаючи приведення значень до форми, яку легко обробляти на комп’ютері, формування векторної реалізації зображення кожного разу, коли спостерігають за об’єктом.

3) Формування матриці освіти.

4) Прийнятна трансформація матриці навчання в субперцептивному просторі.

5) Розділити простір ознак на класи розпізнавання якимось оптимальним чином.

6) На основі геометричних параметрів простір ознак оптимально розділено на категорії розпізнавання для встановлення вирішальних правил.

Тому для побудови вирішальних правил (або класифікаторів) необхідно оптимізувати функціональні параметри системи розпізнавання. Параметри продуктивності — це характеристики, які ідентифікують систему, що впливає на її функціональну ефективність.

У системі «від навчання до розпізнавання» параметри, з якими вона працює, часто називають параметрами навчання.

Розглянемо кілька прикладів формування вирішальних правил. нехай $X1$ і $X2$ є двома типами розпізнавання, розподіл реалізації яких показано на наступному рисунку:

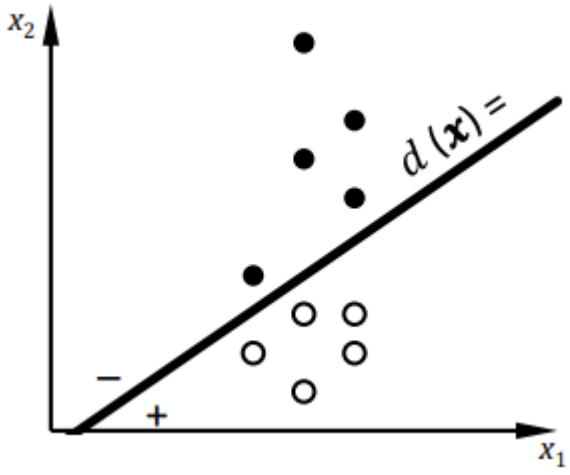

Рис. 1.8. Реалізація розподілу двох типів розпізнавання

Кожна реалізація є вектором ознак розпізнавання x = (x1, x2)$\boldsymbol{T}$, заданим у просторі ознак x1 − x2. На рисунку 1.7 реалізації, що належать до класу $X1$, позначено світлими кружочками, а які належать до класу $X2$ – темними.

Реалізації показаних на рис. 1.7 класів розпізнавання можна розділити прямою: d(x) = w0 + w1x1 + w2x2 = 0.

При цьому значення коефіцієнтів $w0$, $w1$ та $w2$ визначено так, що для всіх реалізацій класу $X1$ виконується нерівність $d(\text{x}) > 0$, а для всіх реалізацій класу $X2$ – $d(\text{x}) < 0$. Крім того, для будь-якої реалізації x, для якої апріорно відомо, що вона належить одному з класів $X1$ або $X2$, можна обчислити значення $d(\text{x})$ і прийняти рішення $\text{x} \in X1$, якщо $d(\text{x}) > 0$, або $\text{x} \in X2$, якщо $d(\text{x}) < 0$. Сформовану в такий спосіб роздільну функцію $d(\text{x})$ називають лінійним вирішальним правилом класу $X1$.

Для N-вимірного випадку $\text{x} = (\text{x}1, \text{x}2, \dots, \text{x}N)\boldsymbol{T}$ пряма перетворюється на гіперплощину:

$$d(\text{x}) = w0 + w1\text{x}1 + w2\text{x}2 + \cdots + wN\text{x}N = w0 + w\,\text{T}\text{x} = 0,$$

де $w = (w1, w2, \dots, wN)$ – вектор коефіцієнтів, до значень яких ставляться вимоги.

Зазвичай вектор-реалізація x розширюється до $\text{x} = (1, x1, x2, \dots, xN)\boldsymbol{T}$,

а вектор коефіцієнтів – до $w = (w0, w1, w2, \dots, wN)$.

Тоді лінійні вирішальні правила набирають вигляду:

$$d(\text{x}) = w\,\text{T}\text{x} = 0.$$

1.4. Критерії якості розпізнавання образів

Якість класифікації об'єктів багато в чому залежить від вибору міри їх близькості (подібності).

Міра близькості - це величина, яка має граничне значення і може збільшуватися в міру зменшення відстані між реалізаціями ідентифікованих класів. Розглянемо основний спосіб визначення близькості між об’єктами за кількісними шкалами вимірювання [7].

Найпоширенішою мірою близькості в теорії розпізнавання образів є евклідова відстань:

$$d_E(x_m, x_l) = \sqrt{\sum_{i=1}^{N} (x_m^i - x_l^i)^2},$$

де xm - реалізація класу xm; hl - реалізація класу hl; xmi - i-та ознака розпізнавання Xmo; hli - ознака розпізнавання i-го класу Xli; N - кількість ознак; N - кількість ознак кількість категорій розпізнавання.

Ця відстань відповідає інтуїтивному уявленню про геометричну близькість двох точок у багатовимірному просторі ознак і є особливо ефективним у розділенні векторів реалізації ідентифікованих класів при обґрунтуванні припущення про компактність їх розподілів. Ця метрика широко використовується в методах кластерного аналізу на основі детермінованих критеріїв відстані.

Щоб надати більшої ваги реалізаціям ідентифікованих класів, які знаходяться далеко один від одного, використовуйте квадрат евклідової відстані

$$d_E^2(x_m,\ x_l) = \sum_{i=1}^{N} (x_m^i - x_l^i)^2.$$

Ця міра дозволяє збільшити внески віддалених реалізацій, оскільки вони зведені в квадрат. Щоб зменшити вплив великих відмінностей, використовується лінійна (манхеттенська) міра наближення:

$$d(x_m, x_l) = \sum_{i=1}^{N} | x_m^i - x_l^i |,\ m,l = \overline{1,M}.$$

У випадках, пов'язаних з багатокритеріальною оцінкою (наприклад, якість продукції), для порівняння об'єктів доцільно використовувати модуль відхилення.

Використовуйте узагальнену дистанцію потужності у випадках, коли вам потрібно зменшити або збільшити ваги реалізації зображень, які сильно відрізняються:

$$d\ (x_m,\ x_l) = \sqrt[r]{\sum_{i=1}^{N} (x_m^i - x_l^i)^p}\ ,$$

де r — параметр, відповідальний за прогресивне зважування великих відстаней між об'єктами; p — параметр, відповідальний за прогресивне зважування відмінностей через окремі координати.

Якщо обидва аргументи у виразі дорівнюють 2, відстань дорівнює евклідовій відстані.

На практиці існує припущення, що зображення досягають нечіткої компактності, тому що попередні класи перетинаються і мають нечіткі (нечіткі) межі. Таким чином, застосування вищезгаданого детермінованого критерію відстані до завдань класифікації не дозволяє чітко поділити простір ознак на класи розпізнавання в цілому.

ВИСНОВКИ ДО РОЗДІЛУ 1

У цьому розділі аналізується сучасне портфоліо програмного забезпечення для розпізнавання образів, наприклад: *CuneiForm, FineReader, Readiris*. Описано методи розпізнавання образів. Дано визначення нейронних мереж і глибоких нейронних мереж з урахуванням елементів нейронних мереж.

Розділені на дві групи методів розпізнавання образів: методи, які використовують матриці навчання класифікації для генерації правил прийняття рішень, і методи автоматичної класифікації, здатні обробляти некласифіковані дані (методи неконтрольованого навчання), також аналізуються критерії якості для розпізнавання образів.

Відповідно до поставлених цілей поставлено в роботі наступні завдання: аналіз сучасних програмних комплексів розпізнавання образів; проаналізувати типи нейронних мереж, які можна використовувати в програмних модулях розпізнавання зображень; проаналізувати архітектуру згорткової нейронної мережі та вибрати програмний модуль для розпізнавання зображень; розробити програмний модуль розпізнавання зображень на основі нейронної мережі; перевірити роботу програмного модуля.

РОЗДІЛ 2
ОГЛЯД НЕЙРОННИХ МЕРЕЖ І МЕТОДІВ ЇХ НАВЧАННЯ

2.1. Типи нейронних мереж

Різні типи нейронних мереж використовують різні принципи при визначенні власних правил. Існує багато типів штучних нейронних мереж, кожна з яких має свої унікальні переваги.

Нейронні мережі прямого розповсюдження є одними з найпростіших типів штучних нейронних мереж. У нейронній мережі прямого розповсюдження дані проходять через різні вхідні вузли, поки не досягнуть вихідного вузла. Іншими словами, дані поширюються від першого рівня лише в одному напрямку, поки не досягнуть вихідного вузла. Цей тип також описується як хвиля, що поширюється, і зазвичай реалізується за допомогою функції активації класифікації.На відміну від більш складних типів нейронних мереж, він не має зворотного поширення, і дані переміщуються лише в одному напрямку. Нейронна мережа прямого розповсюдження може мати один шар або один прихований шар.

Ці нейронні мережі використовуються в таких технологіях, як розпізнавання обличчя та комп’ютерний зір. Це тому, що цільові класи в цих програмах важко класифікувати. Проста нейронна мережа прямого розповсюдження може обробляти шумові дані. Нейронні мережі прямого поширення також прості в обслуговуванні [8].

Багатошарові перцептрони мають три або більше шарів. Він використовується для класифікації даних, які не можна розділити лінійно. Це повністю пов’язана штучна нейронна мережа. Це пояснюється тим, що кожен вузол одного шару з’єднаний з кожним вузлом наступного рівня.

Багатошарові персептрони використовують нелінійні функції активації (переважно гіперболічний тангенс або логістичні функції).

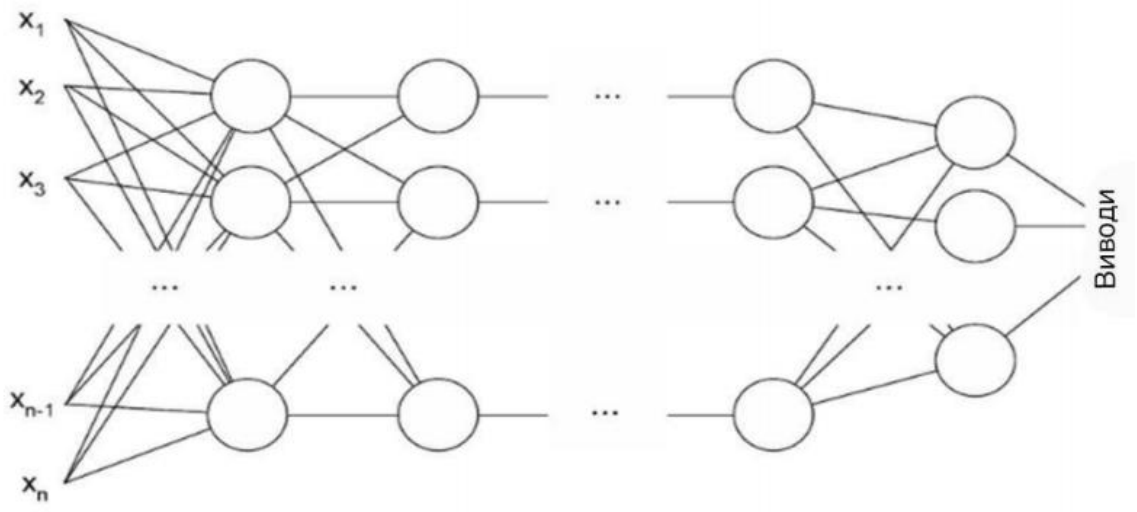

Рис. 2.1. Багатошаровий перцептрон

Цей тип нейронної мережі широко використовується в технології розпізнавання мови та машинного перекладу. Багатошарові персептрони або нейронні мережі прямого зв’язку з двома або більше рівнями мають більшу потужність обробки, а також можуть обробляти нелінійні структури.

Згорткові нейронні мережі. Згорткові нейронні мережі (*CNN*) використовують варіанти багатошарових персептронів. *CNN* містять один або більше згорткових шарів. Ці шари можуть бути повністю з'єднані між собою або об'єднані.

Згортковий рівень застосовує операцію згортки до вхідних даних перед передачею результату наступному шару. За допомогою цієї операції згортки мережа може бути глибшою, але з набагато меншими параметрами.

Завдяки цій можливості згорткові нейронні мережі показали дуже ефективні результати в розпізнаванні зображень і відео, обробці природної мови та системах рекомендацій [8].

Згорткові нейронні мережі також показали відмінні результати в семантичному аналізі. Вони також використовуються для обробки сигналів і класифікації зображень. *CNN* також використовуються для аналізу та розпізнавання зображень у сільському господарстві, де погодні характеристики отримують із супутників типу *LSAT* для прогнозування росту та врожайності земельної ділянки. На рис. 2.2 показано, як виглядає згорточна нейронна мережа.

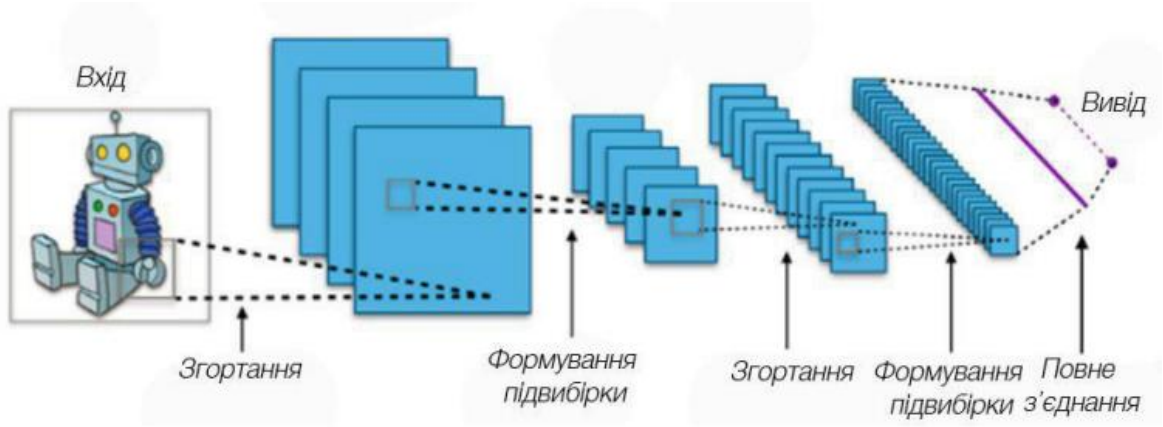

Рис. 2.2. Згорткові нейронні мережі

Рекурентні нейронні мережі мають багато різних мереж, які працюють незалежно та виконують завдання. Різні мережі насправді не спілкуються одна з одною і не передають сигнали під час обчислень. Вони працюють самостійно, щоб досягти результату.

Тому, розбиваючи його на незалежні компоненти, великі та складні обчислювальні процеси можна завершити швидше. Швидкість обчислень збільшується, оскільки мережі не взаємодіють і навіть не підключаються одна до одної. На рис. 2.3 зображено принципову схему модульної нейронної мережі.

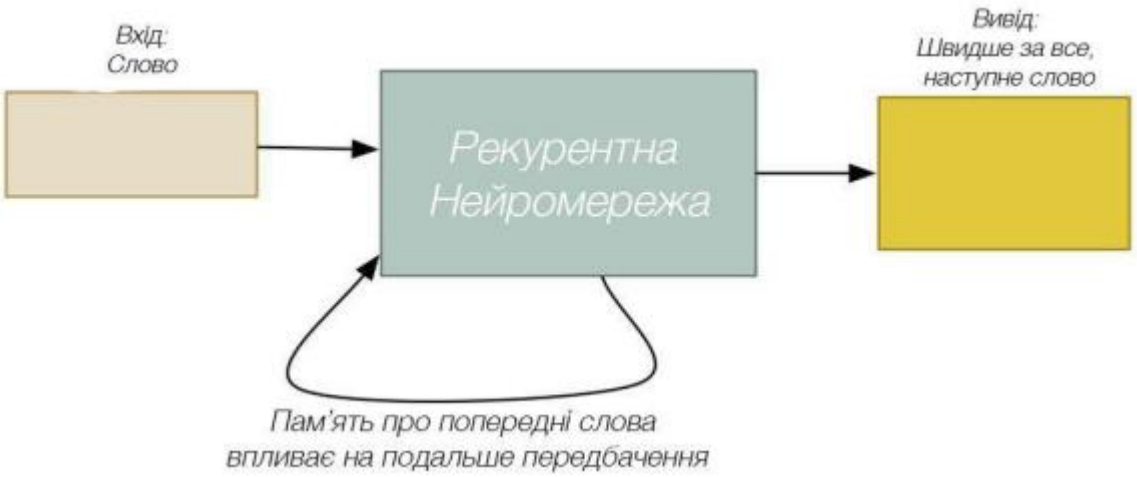

Рис. 2.3. Рекурентні нейронні мережі

Модульні нейронні мережі мають багато різних мереж, які працюють незалежно та виконують завдання. Різні мережі насправді не спілкуються одна з одною і не передають сигнали під час обчислень. Вони працюють самостійно, щоб досягти результату.

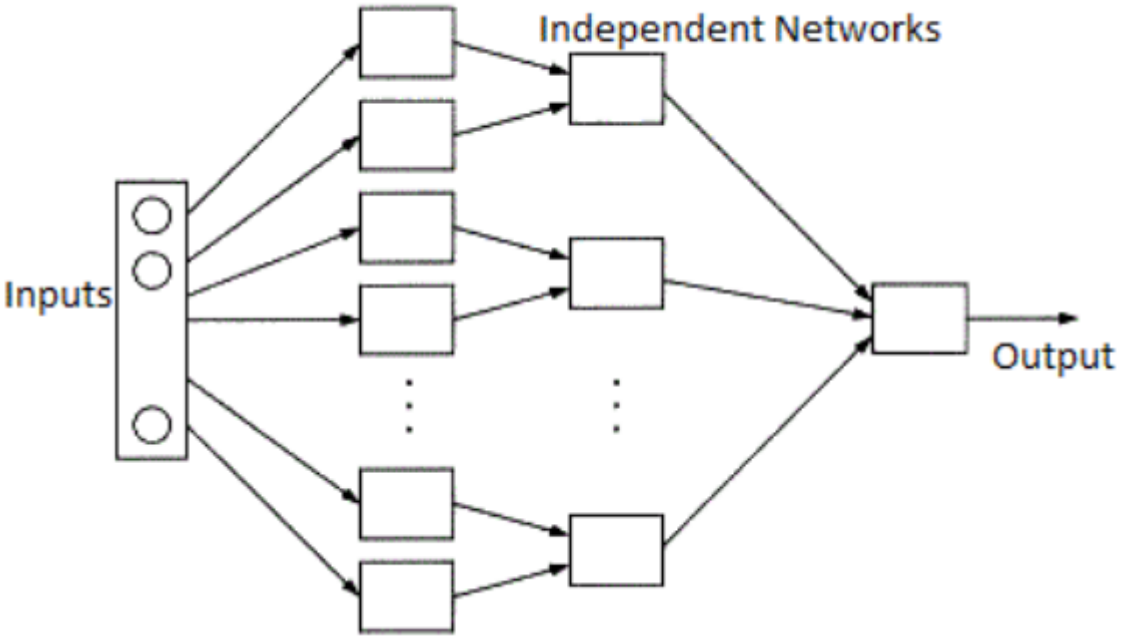

Рис. 2.4. Модульні нейронні мережі

Тому, розбиваючи його на незалежні компоненти, великі та складні обчислювальні процеси можна завершити швидше. Швидкість обчислень

збільшується, оскільки мережі не взаємодіють і навіть не підключаються одна до одної. На рис. 2.4 показано візуальне представлення модульної нейронної мережі.

2.2. Вивчення нейронних мереж

Як і будь-яка інша модель, нейронні мережі намагаються робити хороші прогнози. У нас є набір вхідних даних і набір цільових значень — ми намагаємося отримати прогнози, які якнайточніше відповідають цим цільовим значенням. На рис. 2.5 показана проста модель частини мережі.

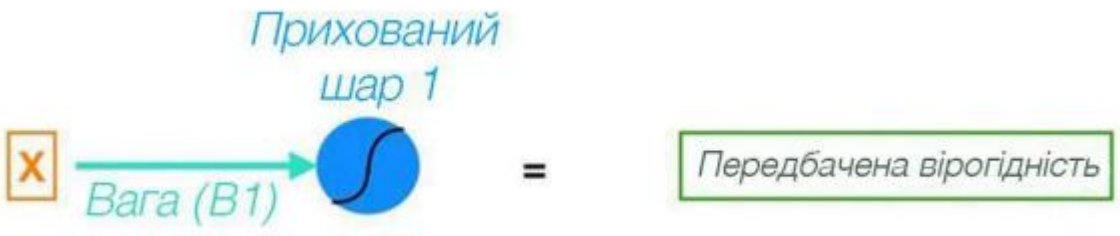

Рис. 2.5. Логістичні операції здійснюються через умову

Це одна логістична регресія, виражена через нейронну мережу (ми надаємо модель лише з однією змінною *X*). Щоб побачити, як вони пов’язані, ми можемо переписати рівняння логістичної регресії, використовуючи коди кольорів нейронної мережі. На рис. 2.6 показано рівняння для цієї операції.

$$Sigmoid(B_1 * X + B_0) = \text{Передбачена вірогідність}$$

Рис. 2.6. Рівняння логістичної регресії

Розглянемо кожен елемент:

– *X* — це наші вхідні дані, єдина функція, яку ми надаємо в моделі для обчислення прогнозів;

– B_1 – приблизний параметр нахилу логістичної регресії;

– B_0 — це зміщення, дуже схоже на член перехоплення в регресії. Ключова відмінність полягає в тому, що в нейронній мережі кожен нейрон має власний термін зміщення.

– блакитний нейрон також включає функцію активації сигмоподібної кишки (представлену кривою всередині синього кола). Сигмоїдна функція використовується для переходу від логарифма до ймовірності;

– отримуємо прогнозовані ймовірності, застосовуючи сигмоїдну функцію до значень ($B1 * X + B0$).

З цього прикладу можна зробити висновок, що надпроста нейронна мережа складається з наступних компонентів:з'єднання (хоча на практиці зазвичай існує багато з'єднань, кожне з яких має власну вагу в певному нейроні), вага якого перетворює ваш вхід (за допомогою $B1$) і передає його нейрону.

Нейрон, включаючи зсувний термін (B_0) і функцію активації.

Хоча ці два об'єкти є основними будівельними блоками нейронних мереж, складніші нейронні мережі — це лише моделі з більшою кількістю прихованих шарів, що означає більше нейронів і більше зв'язків між нейронами. І ця більш складна мережа зв'язків дозволяє нейронній мережі давати прогнозовану відповідь. на рис. На рис. 2.7 зображено принципову схему нейронної мережі.

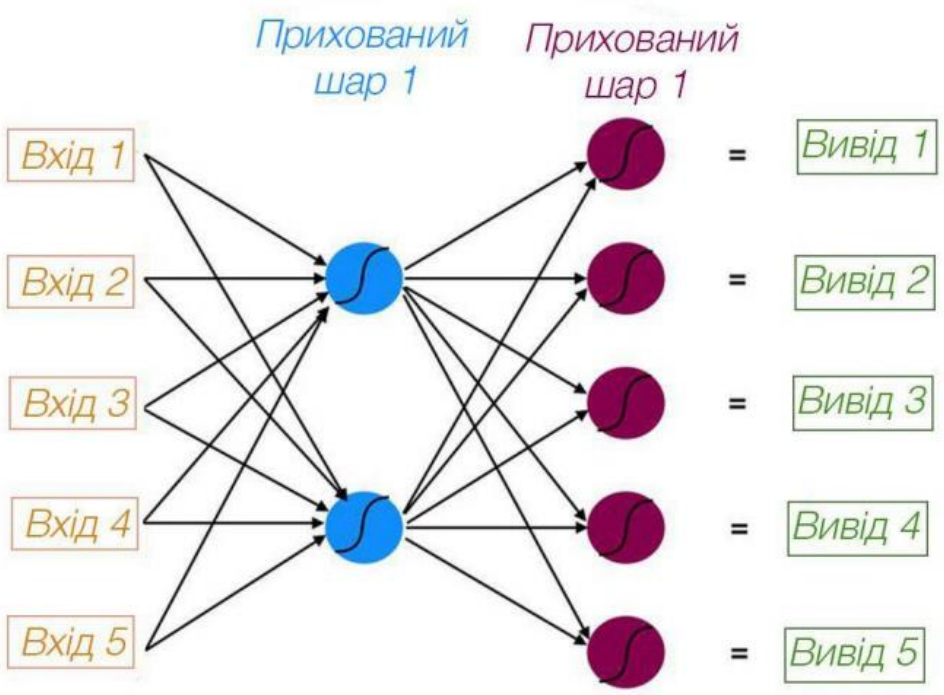

Рис. 2.7. Принципова схема складної нейронної мережі

Перший прихований шар складається з двох нейронів. Щоб підключити всі п’ять входів до нейронів у прихованому шарі 1, нам потрібно десять з’єднань. На рис. 2.7 показано лише зв’язок між входом 1 і прихованим шаром 1.

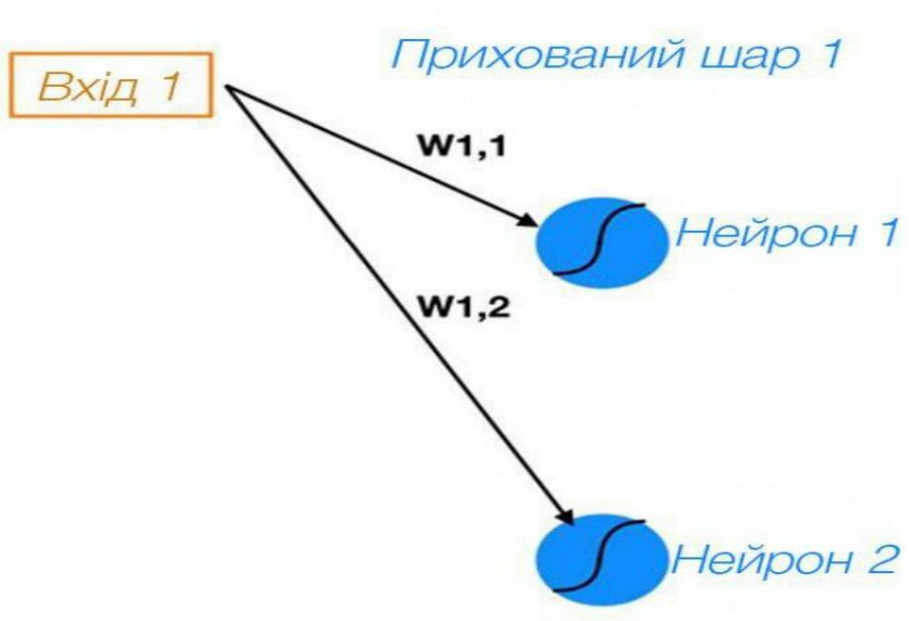

Рис. 2.8. Зв'язок між входом 1 і прихованим шаром

Символи вагових коефіцієнтів, які існують у зв’язках – *W*1,1 представляє ваги, які існують у зв’язку між входом 1 і нейроном 1, а *W*1,2 представляє ваги, які існують у зв’язку між входом 1 і нейроном 2. Загальне позначення *W a , b* позначає вагу зв’язку між входом *a* (або нейроном *a*) і нейроном *b*.

Тепер давайте обчислимо результат (який називається активацією) кожного нейрона в прихованому шарі 1. Для цього ми використовуємо таку формулу (*W* для ваг і *B* для вхідних даних).

$$Z1 = W1 * In1 + W2 * In2 + W3 * In3 + W4 * In4 + W5 * In5 + Bias\_Neuron1$$

Щоб узагальнити цей розрахунок за допомогою матричної математики, запишемо формулу на рис. 2.9.

$$\begin{bmatrix} W1,1 & W2,1 & W3,1 & W4,1 & W5,1 \\ W1,2 & W2,2 & W3,2 & W4,2 & W5,2 \end{bmatrix} x \begin{bmatrix} X1 \\ X2 \\ X3 \\ X4 \\ X5 \end{bmatrix} + \begin{bmatrix} Bias1 \\ Bias2 \end{bmatrix} = \begin{bmatrix} Z1 \\ Z2 \end{bmatrix}$$

Рис. 2.9. Представляє залежності між вузлами у формі матриці

Для будь-якого рівня нейронної мережі попередній рівень має глибину *m* елементів, а поточний – *n* елементів, підсумовуючи наступним чином:

[*W*] * [*X*] + [зміщення] = [*Z*]

Якщо [*W*] — матриця ваг розміром *n* × *m* (зв’язки між попереднім шаром і поточним шаром), то [*X*] — це матриця вихідних входів або активацій попереднього рівня розміром *m* × 1, а [*Bias*] — Матриця *n* × 1. Матриця зміщення нейронів [*Z*] є проміжною вихідною матрицею *n* × 1. Коли ми маємо [*Z*], ми можемо застосувати функцію активації (у нашому випадку

сигмоід) до кожного елемента [Z], який дає нам нейронні виходи (активації) поточного рівня. На рис. 2.8 показано кожен із цих елементів.

Багаторазово обчислюючи [Z] і застосовуючи функцію активації для кожного наступного шару, ми можемо перейти від початку до результату. Цей процес називається прямим поширенням. Тепер, коли ми знаємо, як обчислюються результати, настав час почати оцінювати якість результатів і тренувати нашу нейронну мережу.

У традиційній лінійній або логістичній регресії ми шукаємо коефіцієнти (B_0, B_1, B_2 тощо), які мінімізують функцію витрат. Для нейронних мереж ми робимо те ж саме, але в більшому масштабі та складніше.

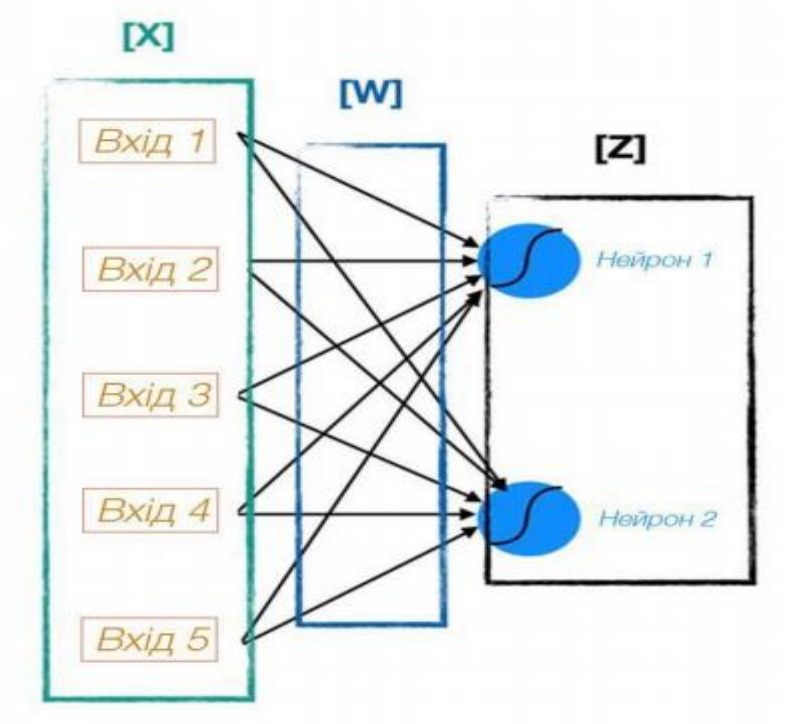

Рис. 2.10. Візуалізація [W], [X] і [Z]

У традиційній регресії ми можемо змінити будь-яке конкретне значення параметра, не впливаючи на інші бета-коефіцієнти. Застосовуючи невеликі окремі зміни до кожної бета-версії та вимірюючи її вплив на функцію витрат, ми можемо відносно легко визначити, у якому напрямку нам потрібно рухатися, зменшити і в кінцевому підсумку знизити витрати.

У нейронній мережі зміни ваги будь-якого з’єднання (або зміщення нейрона) мають зворотний вплив на всі інші нейрони та їх активацію в наступних шарах. Це тому, що кожен нейрон у нейронній мережі діє як власна маленька модель.

Таким чином, кожен прихований рівень нейронної мережі в основному є групою моделей (кожен окремий нейрон у цьому шарі діє як власна модель), вихідні дані яких передаються до більшої кількості моделей нижче (кожен наступний прихований шар нейронної мережі містить більше нейронів) [4].

Щоб навчити нашу нейронну мережу, ми будемо використовувати середньоквадратичну помилку (СКП) як функцію вартості:

СКП = *SUM* [(прогноз - фактичний)^2] * (1 / кількість_спостережень)

СКП – це середнє значення помилок, але з особливою особливістю: порівнюючи наші помилки прогнозу перед усередненням, ми відхиляємо прогнози, які набагато гірші, ніж ті, які незначно відхиляються. Функція витрат для лінійної регресії дуже подібна до функції логістичної регресії.

Щоб використовувати градієнтний спуск, вам потрібно знати градієнт нашої функції витрат, яка є вектором, спрямованим у найкрутішому напрямку (щоб отримати результат, вам потрібно продовжувати брати від’ємний градієнт, щоб знайти мінімум).

Градієнт функції — це вектор, елементами якого є часткові похідні за кожним параметром. Наприклад, якби ми намагалися мінімізувати функцію витрат $C(B0,B1)$ лише з двома змінними $B0$ та $B1$, градієнт був би:

Градієнт C (B_0, B_1) = [[dC / dB_0], [dC / dB_1]]

Таким чином, кожен елемент градієнта повідомляє нам, як зміниться функція вартості, якщо ми застосуємо невелику зміну до цього конкретного параметра, щоб ми знали, що налаштувати та скільки. На рис. 2.11 показано ілюстрацію знаходження мінімального значення. Підводячи підсумок, ми можемо досягти мінімальних вимог, виконавши такі кроки:

– розрахувати градієнт нашого поточного положення;

– змініть кожен параметр на величину, пропорційну його градієнтному елементу, і в протилежному напрямку його градієнтного елемента.

– наприклад, якщо наша функція витрат має додатну, але малу часткову похідну щодо $B0$, і від'ємну та велику часткову похідну щодо $B1$, тоді ми хочемо зменшити $B0$ на невелику величину та збільшити $B1$ на велику величину, тоді ми зменшуємо нашу функцію витрат;

– повторно обчисліть градієнт із нещодавно налаштованими значеннями параметрів і повторюйте попередні кроки, доки не буде досягнуто мінімальне значення.

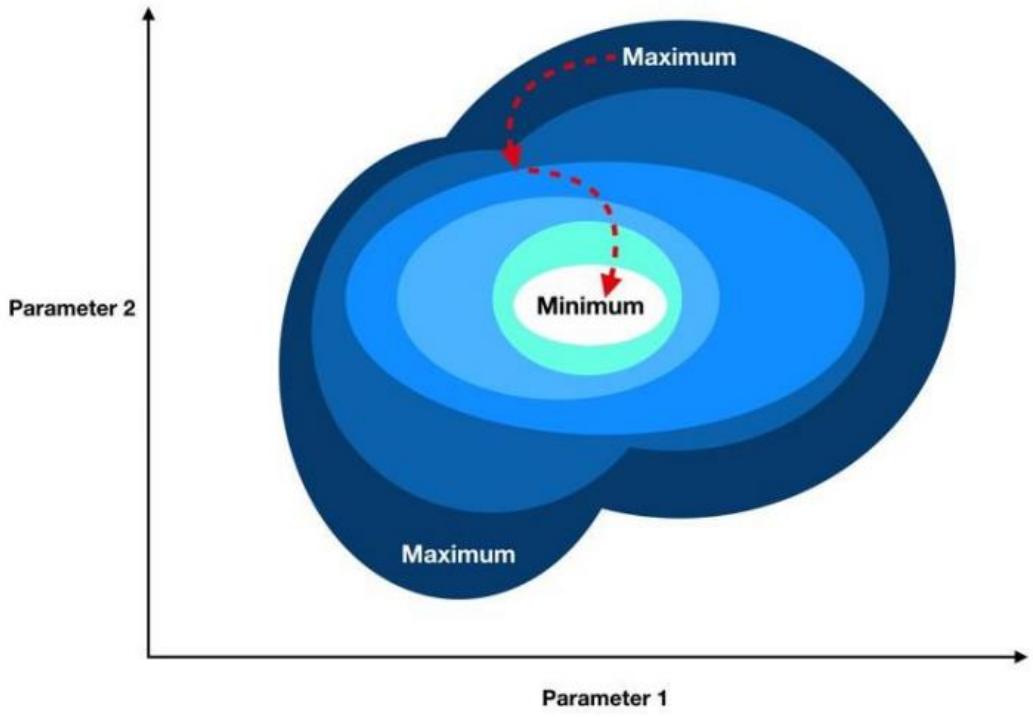

Рис. 2.11. Ілюстрація градієнтного спуску

Попереднє розповсюдження — це процес просування нейронною мережею (від вхідних даних до кінцевих результатів або прогнозів). Зворотне поширення - це зворотне поширення через нейронну мережу.

На рис. 2.12 показана проста нейронна мережа, яка поширюється вперед від входу до виходу. Процес можна підсумувати в наступних кроках:

– вхідні дані подаються на синій шар нейронів і модифікуються ваговими коефіцієнтами, зміщеннями та сигмою в кожному нейроні для отримання активацій. Наприклад: *activation*_1 = *sigmoid* (*offset*_1 + *W*1 * *input*_1);

– активація 1 і активація 2 із синього шару надходять до пурпурового нейрона, який використовує їх для отримання остаточної вихідної активації.

Метою зворотного поширення є обчислення активації кожного нейрона кожного наступного прихованого шару, поки ми не досягнемо результату.

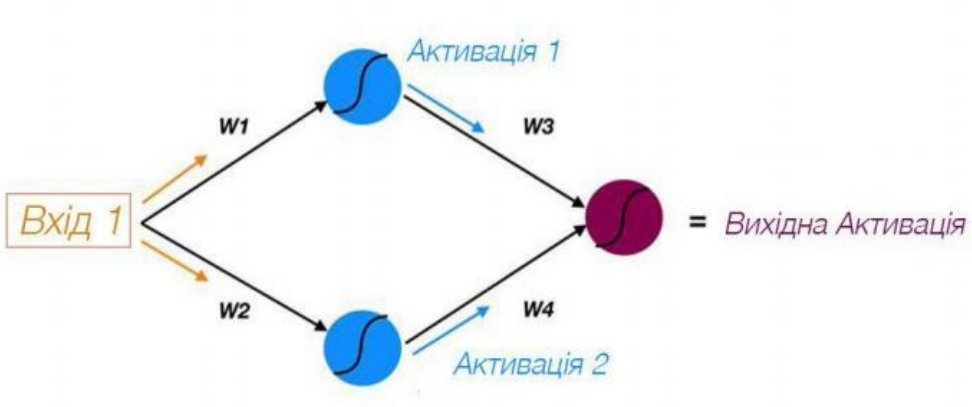

Рис. 2.12. Розподіл нейронної мережі

На рис. 2.13 показано вихідну активацію, яка використовується для прогнозування та як кінцеве джерело помилки в моделі. Ця помилка потім переміщується назад у модель, використовуючи ті самі вагові коефіцієнти та зв’язки, що використовуються для передачі сигналу (тому замість ініціювання 1 ми тепер використовуємо *error*1 – помилку, приписувану верхньому синьому нейрону).

Метою прямого розповсюдження є обчислення активації нейронів шар за шаром, поки ми не досягнемо результату.

Таким чином, помилка, приписувана кожному нейрону, обчислюється, починаючи від шару, найближчого до виходу, аж до початку шарів нашої моделі, що називається зворотним поширенням.

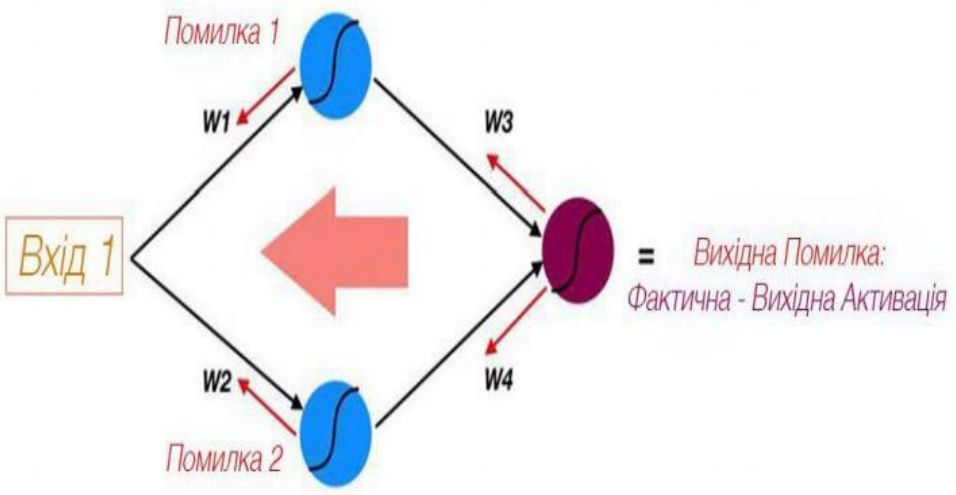

Рис. 2.13. Зворотне поширення в нейронних мережах

Двома будівельними блоками нейронної мережі є з'єднання, які передають сигнали певним нейронам (із ваговими коефіцієнтами для кожного з'єднання), і самі нейрони (зі зміщеннями). Ці повні ваги та зміщення мережі також є ваговими коефіцієнтами, які ми коригуємо, щоб змінити прогнози, які робить модель.

Величина помилки конкретного нейрона (відносно помилки всіх інших нейронів) пропорційна ефекту результату цього нейрона (також називається активацією) на нашу функцію витрат.

Таким чином, помилка кожного нейрона являє собою часткову похідну функції вартості по відношенню до вхідних даних цього нейрона. Це інтуїтивно зрозуміло: якщо певний нейрон має набагато більшу помилку, ніж усі інші нейрони, тоді коригування ваг і зміщень нашого нейрона матиме більший вплив на загальну помилку нашої моделі, ніж стимулювання будь-якого іншого нейрона.

А часткові похідні щодо кожної ваги та переміщення є незалежними елементами вектора градієнта, які складають нашу функцію витрат. Таким чином, зворотне поширення дозволяє нам обчислити помилку, приписану кожному нейрону, що, у свою чергу, дозволяє нам обчислити часткові похідні та, зрештою, градієнти, тож ми можемо використовувати градієнтний спад.

ВИСНОВКИ ДО РОЗДІЛУ 2

Різні типи нейронних мереж використовують різні принципи при визначенні власних правил. Існує багато типів штучних нейронних мереж, кожна зі своїми унікальними методами та схемами. У цьому розділі представлено такі типи нейронних мереж: нейронна мережа прямого поширення; багатошарові молекули; згорточна нейронна мережа; рекурентна нейронна мережа; модульні нейронні мережі.

Те, як вихід кожного шару стає входом наступного рівня, залежить від ваги, призначеної цьому конкретному посиланню, що залежить від функції вартості та оптимізатора. Кожен повний прохід і аналіз називають епохою нейронної мережі. Після кожної епохи витрати аналізуються, щоб побачити, де можна покращити модель. Потім функція оптимізації змінює внутрішні механізми мережі, такі як ваги та зміщення, на основі інформації, наданої функцією витрат, доки функція витрат не буде мінімізована. Універсальність багатьох взаємопов'язаних моделей методу та здатність процесу зворотного розповсюдження ефективно й оптимально встановлювати ваги та поведінку кожної моделі дозволяє нейронним мережам надійно вчитися на даних так, як не можуть багато інших алгоритмів.

Виходячи з розглянутої моделі нейронної мережі, згорткові нейронні мережі вибрано тому, що завдяки своїй структурі вони можуть дуже добре витягувати ознаки із зображень, кількість ваг коригування набагато менша, оскільки одне ядро ваг використовується повністю для всього зображення, а не виконуючи свої індивідуальні вагові коефіцієнти для кожного пікселя вхідного зображення. Це спонукає нейронну мережу узагальнювати відображену інформацію під час навчання, а не запам'ятовувати кожне відображене зображення піксель за пікселем із незліченними ваговими коефіцієнтами, як це робить перцептрон. Крім того, *CNN* має найвищу продуктивність у розпізнаванні зображень, а також добре підходить для обчислювальних даних в умовах реального часу, розпаралелювання обчислень є зручним, і він стійкий до обертання та зсуву зображення.

РОЗДІЛ 3
АНАЛІЗ РІШЕНЬ ДЛЯ РОЗРОБКИ СИСТЕМИ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

3.1. Вибір архітектури *CNN* для розпізнавання зображень

Існують наступні популярні архітектури *CNN* для розпізнавання об’єктів:

– *R-CNN*. Ми, мабуть, перша модель, яка вирішила це завдання. Це як звичайний класифікатор зображень. На вхід мережі надходять різні області зображення, по яких робляться прогнози. дуже повільно, оскільки воно запускає зображення тисячі разів;

– *Fast R-CNN*. Покращена та швидша версія *R-CNN* працює аналогічно, але спочатку передає все зображення на вхід *CNN*, а потім генерує області зі згенерованого внутрішнього представлення. Але це все ще дуже повільно для завдань у реальному часі;*Fast R-CNN*. Основна відмінність від попереднього полягає в тому, що замість використання алгоритму вибіркового пошуку він використовує нейронну мережу для вивчення регіонів;

– маска *R-CNN*. Швидше розширення *R-CNN*;

– *YOLO*. Принцип роботи повністю відрізняється від попереднього, а площа взагалі не використовується, що є найшвидшим;

– *SSD*. За принципом схожий на *YOLO*, але використовує *VGG*16 як мережу вилучення функцій. Він також досить швидкий і підходить для роботи в реальному часі;

– функціональна мережа піраміди (*FPN*). Інший тип мережі *Single Shot Detector*, завдяки характеристикам виділення ознак, може ідентифікувати невеликі об’єкти краще, ніж *SSD*;

– *Retina Web*. Використовується комбінація *FPN + ResNet*, яка забезпечує більш високу точність завдяки спеціальній функції похибки (фокусні втрати).

Детектор *R-CNN* спочатку генерує пропозиції регіонів за допомогою таких алгоритмів, як крайові рамки. Область пропозиції обрізається та змінюється розмір зображення. Потім *CNN* класифікує обрізані та змінені регіони. Нарешті, обмежувальні прямокутники пропозицій регіонів уточнюються машиною опорних векторів (*SVM*), навченою за допомогою функцій *CNN* [9].

Слід зазначити, що область, отримана за допомогою вибіркового пошуку, може містити лише деякі об’єкти, а не всі. Чи розглядаються регіони як об’єкти, що містять, визначається показником *Intersection over Union* (*IoU*). Ця міра є відношенням площі перетину площі прямокутника-кандидата і прямокутників, які фактично покривають об'єкт, до площі об'єднання цих прямокутників. Якщо співвідношення перевищує заданий поріг, вважається, що регіон-кандидат містить бажаний об’єкт.

IoU також використовується для фільтрації занадто великої кількості регіонів, що містять об’єкт (немаксимальне придушення). Якщо *IoU* регіону та регіону, який отримав найбільший результат для того самого об’єкта, перевищує певне порогове значення, перший регіон просто відкидається.

Під час аналізу помилок автори також розробили метод, який зменшує помилки у виборі меж об’єктів – регресію обмежувальної рамки. Після класифікації вмісту регіонів-кандидатів чотири параметри - (*dx, dy, dw, dh*) визначаються за допомогою лінійної регресії на основі ознак *CNN*.

Вони описують, наскільки центр рамки області має бути зміщений за *x* і *y*, і наскільки її ширина та висота мають бути змінені, щоб точніше охоплювати ідентифіковані об’єкти.

Процес виявлення цілі мережею *R-CNN* можна розділити на наступні етапи:

– використовуйте вибірковий пошук для вибору регіонів-кандидатів;

– перетворення регіонів на розміри, прийняті *CNN CaffeNet*;

– використовуйте *CNN*, щоб отримати 4096-вимірний вектор ознак;

– виконайте *N* бінарних класифікацій для кожного вектора ознак, використовуючи *N* лінійних *SVM*;

– лінійна регресія параметрів кадру області для більш точного покриття об'єкта.

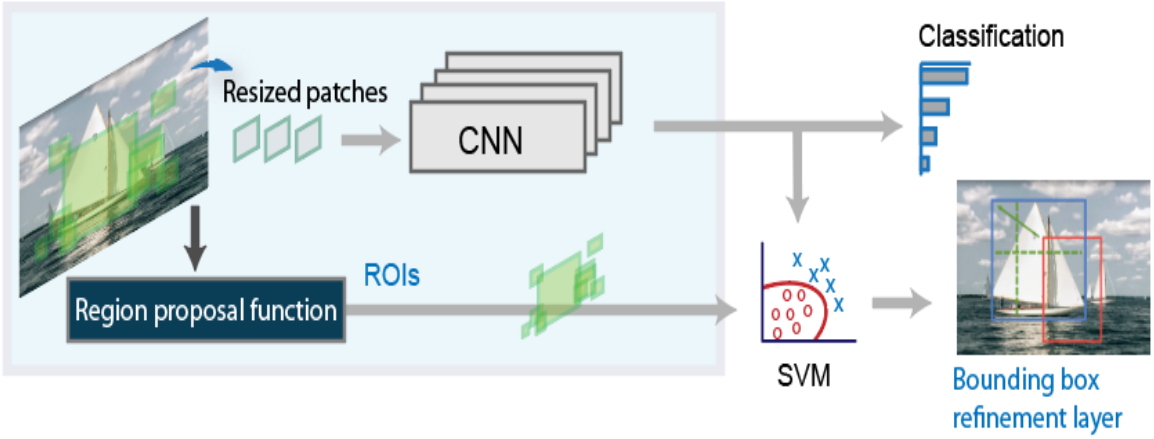

Рис. 3.1. Як працює архітектура *R-CNN*

Як і детектор *R-CNN*, *Fast R-CNN* також використовує алгоритм *Edge Boxes* для створення пропозицій регіону. На відміну від детекторів *R-CNN*, які збирають пропозиції регіонів і змінюють розміри, детектори *Fast R-CNN* обробляють все зображення. У той час як детектор *R-CNN* має класифікувати кожен регіон, функції *Fast R-CNN* поєднують функції *CNN*, що відповідають кожній пропозиції регіону. *Fast R-CNN* більш ефективний, ніж *R-CNN*, оскільки обчислення областей, що перекриваються, розподіляється між швидкими детекторами *R-CNN* [9].

Незважаючи на високі результати, продуктивність *R-CNN* все ще низька, особливо для мереж глибше *CaffeNet* (наприклад, *VGG*16). Крім того, регресор обмежувальної рамки та навчання *SVM* вимагають зберігання великої кількості функцій на диску, тому вони дорогі з точки зору розміру сховища.

Автори *Fast R-CNN* пропонують кілька модифікацій, щоб прискорити процес:

– замість того, щоб передавати кожен регіон-кандидат окремо, все зображення передається через *CNN* в цілому. Потім запропоновані регіони накладаються на згенеровану загальну карту характеристик;

– замість самостійного навчання трьох моделей (*CNN*, *SVM*, *bbox regressor*) ми об'єднуємо всі навчальні процеси в один.

Об'єкти, що потрапляють у різні регіони, перетворюються на фіксований розмір за допомогою програми *RoIPooling*. Розділіть вікно області шириною *w* і висотою *h* на сітку з клітинками $H \times W$ розміром $h / H \times w / W$. Виконайте *Max Pooling* для кожної такої комірки, щоб вибрати лише одне значення, що дасть результуючу матрицю ознак $H \times W$.

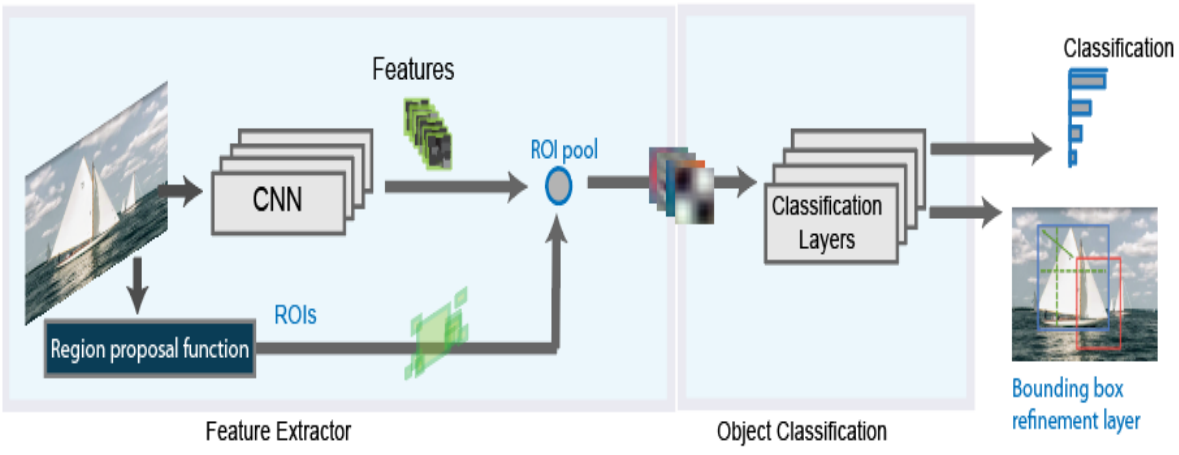

Рис. 3.2. Як працює архітектура *Fast R-CNN*

Fast R-CNN додає мережу регіональних пропозицій (*RPN*) для створення регіональних пропозицій безпосередньо в сітчастій мережі за допомогою зовнішніх алгоритмів, таких як крайові блоки. *RPN* використовує

блоки підтримки для виявлення об’єктів. Створення регіональних пропозицій в мережі відбувається швидше і краще адаптується до даних користувачів [9].

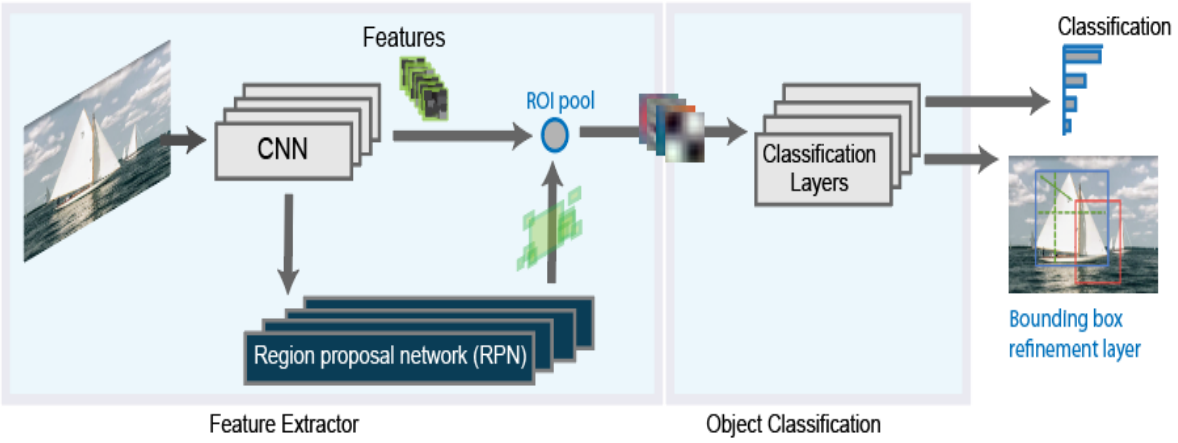

Рис. 3.3. Як працює архітектура *Faster R-CNN*

Mask R-CNN розвиває архітектуру *Faster R-CNN*, додаючи ще одну гілку, яка передбачає, де маска покриває знайдений об’єкт, таким чином уже вирішуючи завдання сегментації екземпляра. Маска — це просто прямокутна матриця, де 1 у позиції означає, що відповідний піксель належить об’єкту даного класу, а 0 означає, що піксель не належить цьому об’єкту.

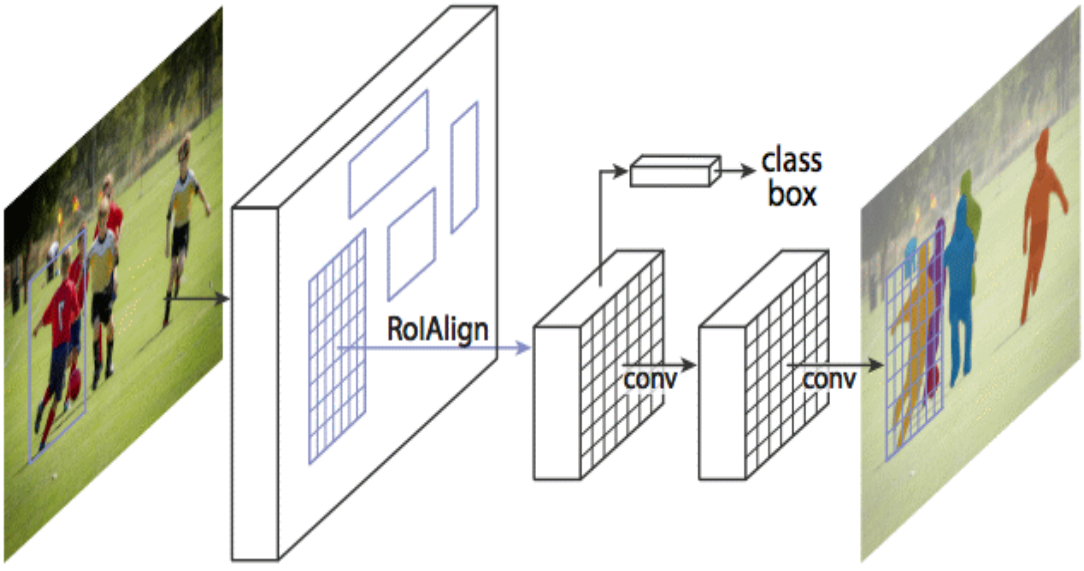

Рис. 3.4. Як працює архітектура *Faster R-CNN*

Автори статті умовно поділяють розроблену архітектуру на мережу *CNN* для обчислення характеристик зображення, яку вони називають

магістральною мережею, і голову – об’єднання, відповідальне за прогнозування частин, що покривають кадр, Визначення класифікацій об'єктів та їх масок. Їх функція втрат є загальною і складається з трьох компонентів:

$$L = L_{cls} + L_{box} + L_{mask}$$

Вибір маски здійснюється незалежно від класу: маски прогнозуються для кожного класу окремо, без попереднього знання того, що представлено в регіоні, і просто вибирається маска для класу, який перемагає в незалежному класифікаторі. Було стверджено, що цей підхід ефективніший, ніж покладатися на попередні знання про клас.

Однією з головних модифікацій, пов’язаних із необхідністю прогнозування маски, є зміна процесу *RoIPool* (обчислення матриці ознак регіону-кандидата) на так званий *RoIAlign*. Справа в тому, що карта функцій, отримана від *CNN*, менша за вихідне зображення, і область на зображенні, яка охоплює цілу кількість пікселів, не може бути відображена в області масштабу цілої кількості карт функцій.

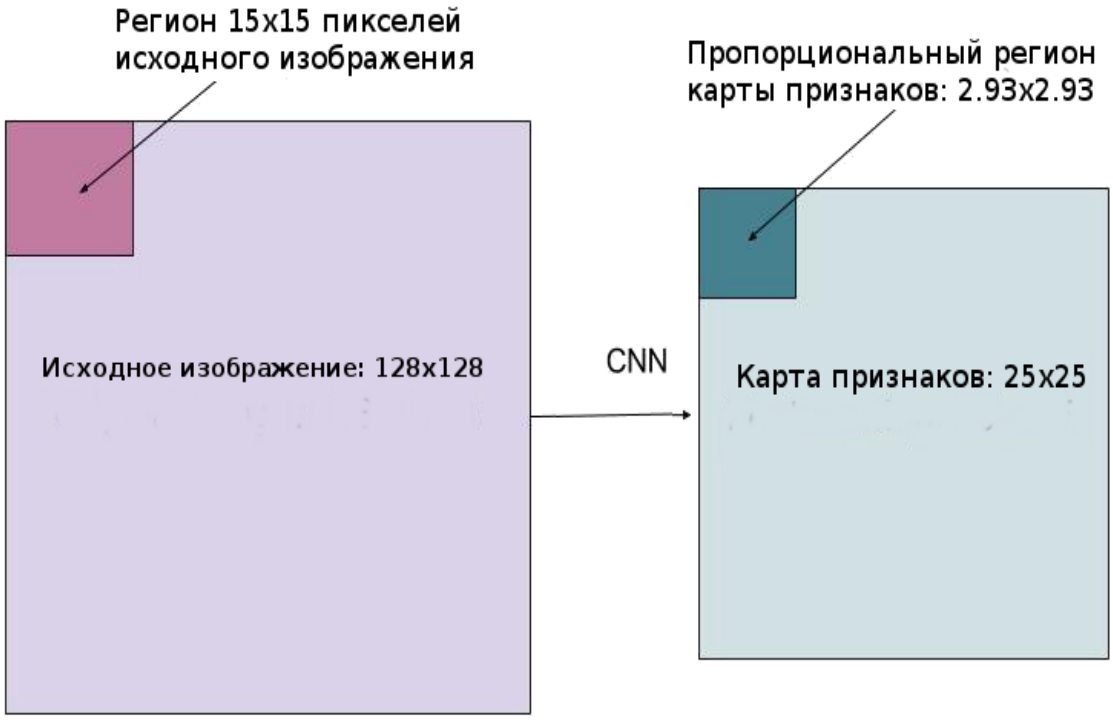

Рис. 3.5. Порівняння карти функцій і оригінального зображення

У *RoIPool* проблему можна вирішити, просто округливши невеликі значення до цілих. Цей підхід добре працює для вибору покриття, але маски, обчислені на основі цих даних, надто неточні.

Навпаки, *RoIAlign* не використовує округлення, усі числа залишаються дійсними, а власні значення обчислюються за допомогою білінійної інтерполяції до чотирьох найближчих цілих точок.

Найбільша перевага режиму *YOLO* насправді відображена в назві - *You Look Only Once*. Модель застосовує сітку до зображення, розділяючи його на комірки. Кожен блок намагається передбачити координати виявленої області, використовуючи достовірні оцінки ймовірностей цих полів і класів. Оцінка достовірності для кожної виявленої області потім множиться на ймовірність класу, щоб отримати остаточну оцінку [10].

YOLO ділить вхідне зображення на сітку *S*×*S*. На комірку сітки надається лише один об’єкт. Наприклад, сітка жовтих клітинок нижче намагається передбачити об’єкт «собака», центр якого знаходиться в клітинці сітки.

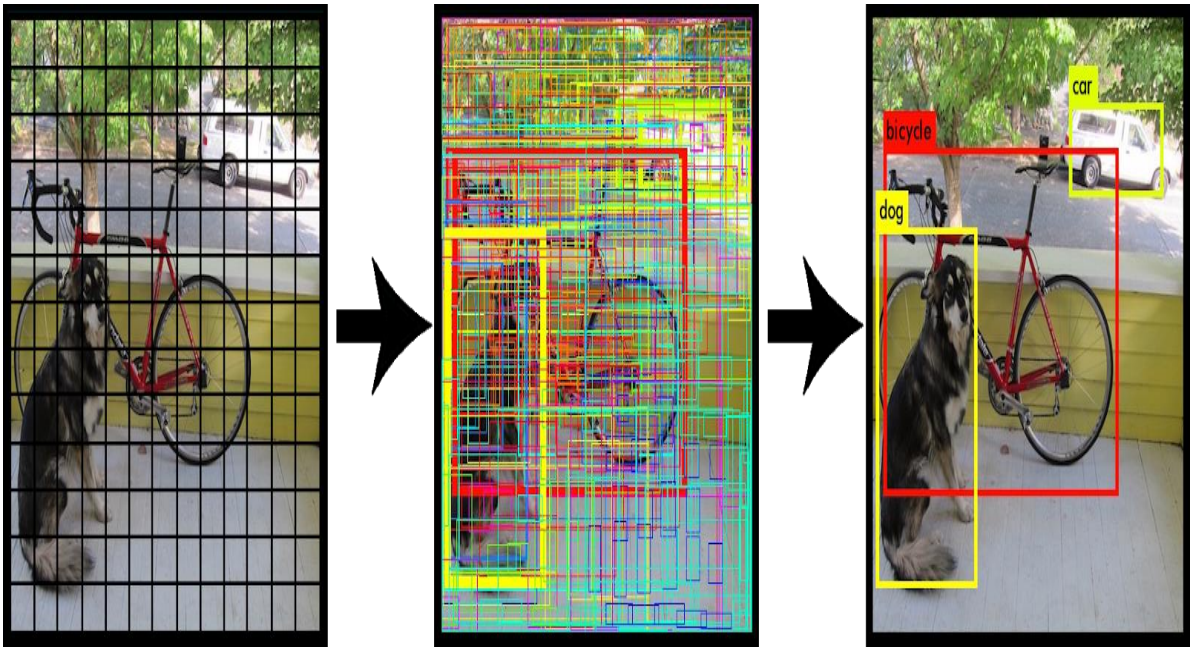

Рис. 3.6 Приклад архітектури *Collar Grid YOLO*

Однак правило одного об’єкта обмежує віддаленість виявлення об’єктів. З цією метою *YOLO* має деякі обмеження щодо близькості об’єктів.

Для кожної комірки сітки *YOLO*:

– припустимо *B* граничні поля, кожне зі шкалою довіри;
– виявляється лише один об’єкт незалежно від кількості вікон *B*;
– передбачте умовні ймовірності класу *C* (одна ймовірність класу об’єкта на клас).

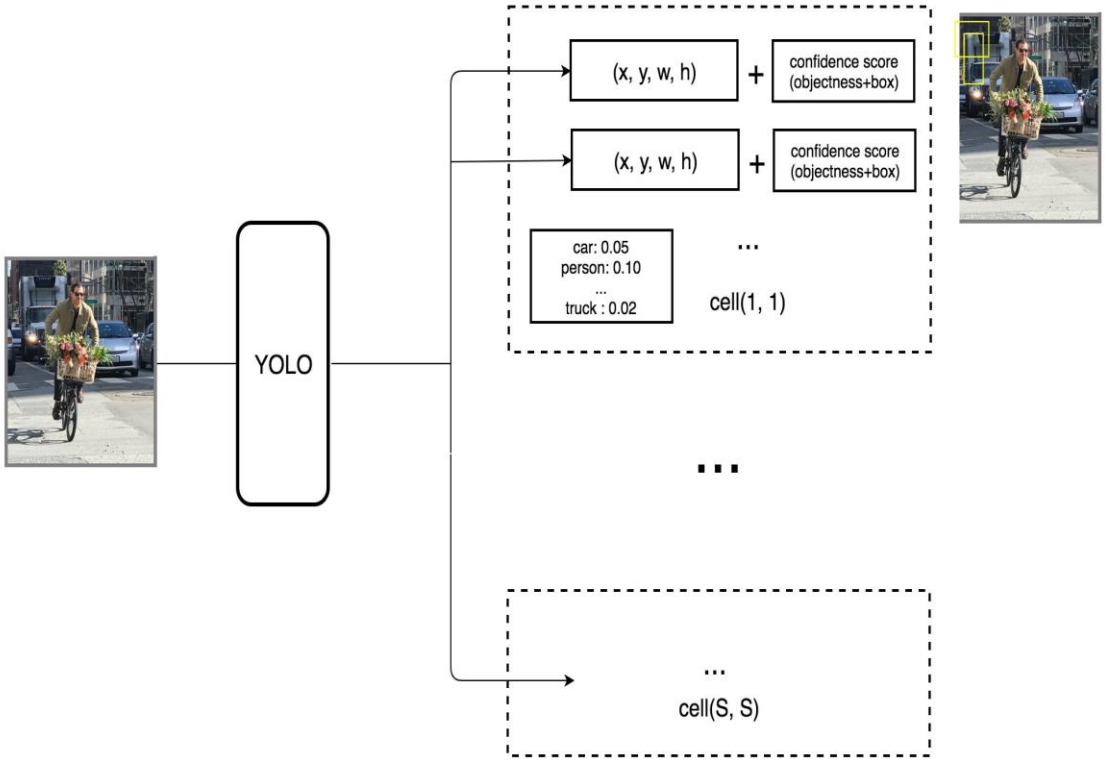

Рис. 3.7. Як працює архітектура *YOLO*

Кожне граничне поле містить 5 елементів: (*x*, *y*, *w*, *h*) та показник надійності поля. Коефіцієнт надійності відображає, наскільки ймовірно коробка містить об’єкти (об’єктивність) і наскільки точні межі. Ми нормалізуємо ширину обмежувальної рамки *w* і висоту *h* до ширини і висоти зображення. *X* і *y* — переміщення відповідних комірок, x, *y*, *w* і *h* від 0 до 1. Кожна комірка має 20 умовних класів ймовірностей. Умовна ймовірність

класу — це ймовірність того, що виявлений об’єкт належить до класу (одна ймовірність на клас на клітинку). Прогноз *YOLO* має форму (*S*, *S*, *B* × 5 + *C*) = (7, 7, 2 × 5 + 20) = (7, 7, 30).

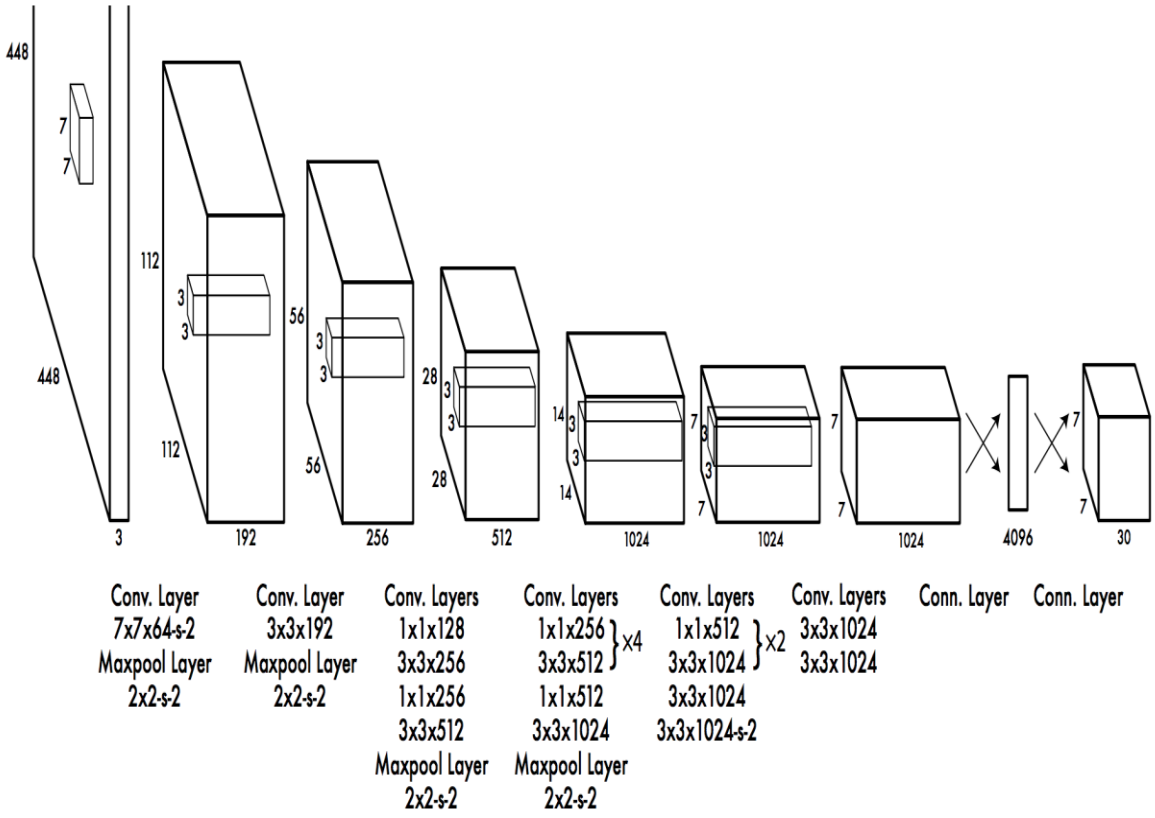

Рис. 3.8. Схема проектування мережі *YOLO*

Основною концепцією *YOLO* є побудова мережі *CNN* для прогнозування (7, 7, 30) тензорів. Він використовує мережу *CNN* для зменшення просторового розміру до 7×7 із 1024 вихідними каналами на місце. *YOLO* виконує лінійну регресію з використанням двох повністю пов’язаних шарів для прогнозування граничного поля 7×7×2 (посередині внизу). Для остаточних прогнозів ми беремо прогнози з високим показником достовірності поля (більше 0,25) як наші остаточні прогнози (правильні зображення).

YOLO має 24 згорткових шари, за якими слідують 2 повністю зв’язані (*FC*) шари. Деякі згорткові шари чергуються з шарами відновлення 1×1, щоб зменшити глибину карти. Для останнього згорткового шару він виводить

тензор у формі (7, 7, 1024). Потім тензор сплющується. Використовуючи 2 повністю зв’язані шари як форму лінійної регресії, він виводить параметри 7 × 7 × 30, які потім перетворюються на (7, 7, 30), тобто 2 прогнози крайового поля на місце.

Швидша, але менш точна версія *YOLO*, яка називається *Fast* (*Tiny*) *YOLO*, використовує лише 9 згорткових шарів і менші карти функцій.

YOLO передбачає кілька обмежувальних рамок на клітинку сітки. Щоб обчислити збитки для справжнього позитиву, ми хочемо, щоб лише один із них відповідав за цей об’єкт. Для цього ми вибираємо той, у якого найвищий *IO* (перетин над об’єднанням) із основною правдою. Ця стратегія призводить до спеціалізації, яка обмежує передбачення рамки. Кожен прогноз покращує прогнозування певних розмірів і пропорцій.

YOLO використовує квадрат помилки між прогнозом і основною правдою для розрахунку втрати. Функції втрати включають:

– втрата класифікації; втрата локалізації (похибка між оціненим граничним полем і правдою землі);

– втрата довіри (об'єктивність ящика).

За допомогою *SSD* нам потрібно зробити лише один знімок, щоб виявити кілька об’єктів на зображенні, тоді як мережі регіональних пропозицій (*RPN*), засновані на таких методах, як сімейство *R-CNN*, потребують двох знімків: одного для створення пропозицій регіону та одного для кожної пропозиції об'єкта виявлення. Таким чином, *SSD* є набагато швидшим порівняно з двосхилим методом *RPN*.

SSD (*Single Shot Detector*) використовує *VGG*16 для отримання карт функцій. Потім він виявляє об’єкти за допомогою рівня *Conv*4_3. Для ілюстрації ми просторово намалюємо *Conv*4_3 розміром 8 × 8 (це має бути 38 × 38). Для кожної комірки (також називається розташуванням) він робить прогнози 4 об’єктів.

Кожен прогноз складається з обмеженого поля та 21 оцінки для кожного класу (додатковий клас без об’єктів), і ми вибираємо найвищу

оцінку як клас із обмеженими об’єктами. *Conv*4_3 робить загалом 38×38×4 передбачень: 4 передбачення на комірку, незалежно від глибини карти зображення. Як і очікувалося, багато прогнозів не містять жодних об'єктів. *SSD* резервує клас 0, щоб вказати, що він не має об’єктів [11].

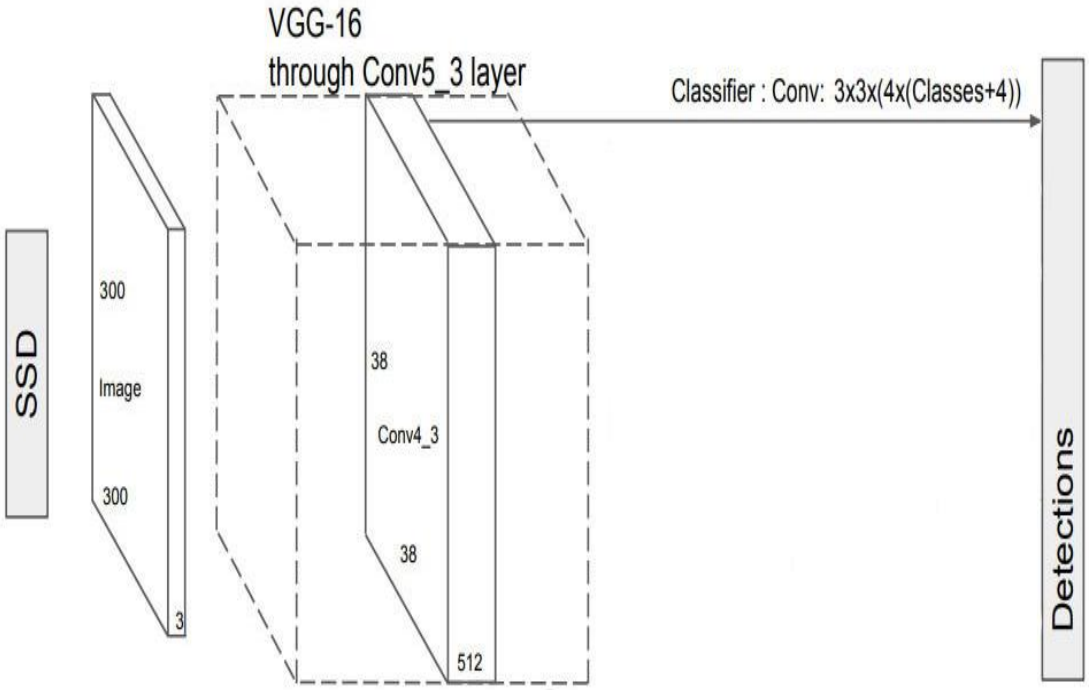

Рис. 3.9. Як працює архітектура *SSD*

В експериментах *Mask R-CNN*, разом із звичайним *ResNet*-50/101 *CNN* як магістраль, також було проведено техніко-економічне обґрунтування з використанням *Feature Pyramid Network* (*FPN*). Вони показують, що застосування *FPN* в магістральній мережі покращує як точність, так і продуктивність *Mask R-CNN*. Тому корисно також описати це вдосконалення, хоча існує окремий документ, присвячений цьому, і він мало пов’язаний із серією, що розглядається. Як і піраміди зображень, піраміди ознак спрямовані на покращення якості виявлення об’єктів, враховуючи їх можливий діапазон розмірів [12].

У мережі піраміди функцій карти функцій, витягнуті шляхом послідовного зменшення шарів *CNN*, розглядаються як свого роду ієрархічна «піраміда», яка називається шляхом «знизу вгору». Водночас карти ознак

нижнього та верхнього шару піраміди мають свої переваги та недоліки: перша має високу роздільну здатність, але низьку семантичну та узагальнюючу здатність, друга – навпаки:

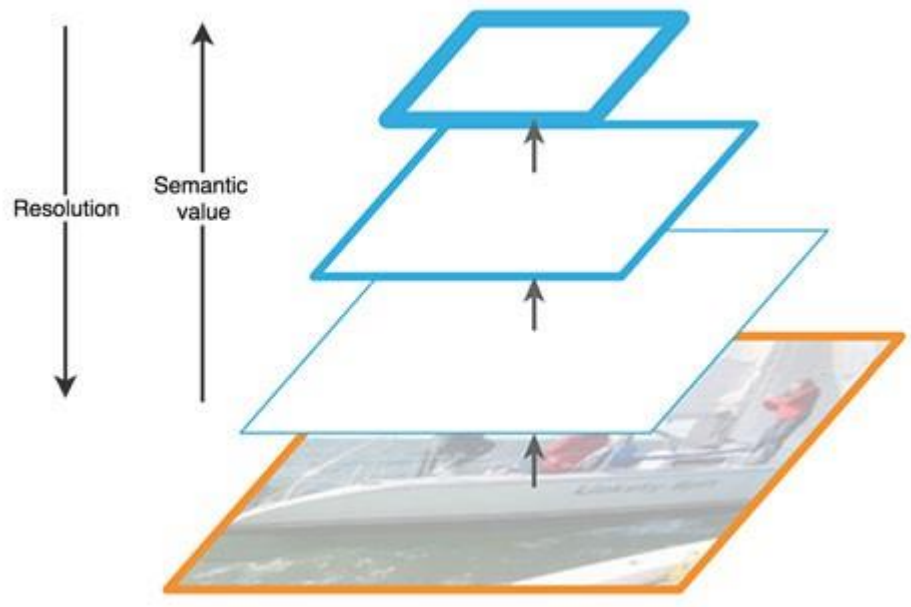

Рис. 3.10. Як працює архітектура *FPN*

Архітектура *FPN* дозволяє поєднувати переваги верхнього та нижнього рівнів шляхом додавання шляхів зверху вниз та бічних з'єднань. Для цього карти кожного вищого шару збільшуються до розмірів нижчих шарів, а їх вміст складається поелементно. Фінальний прогноз використовує остаточну карту всіх рівнів [12].

RetinaNet складається з базової мережі та двох підмереж, які використовують карти функцій базової мережі. Класифікаційна підмережа визначає класи зображень, а регресійна підмережа визначає обмежувальні рамки. Вхідні зображення відрізняються за роздільною здатністю та розміром, тому *RetinaNet* використовує карти функцій різної роздільної здатності. Це робить навчання швидшим і менш громіздким, ніж передача в мережу однакових зображень у різних розширеннях.

Замість використання останніх карт основних функцій ми використовуємо карти функцій, створені на різних рівнях базової мережі. Цей тип мережі також відомий як мережа функціональної піраміди (*FPN*).

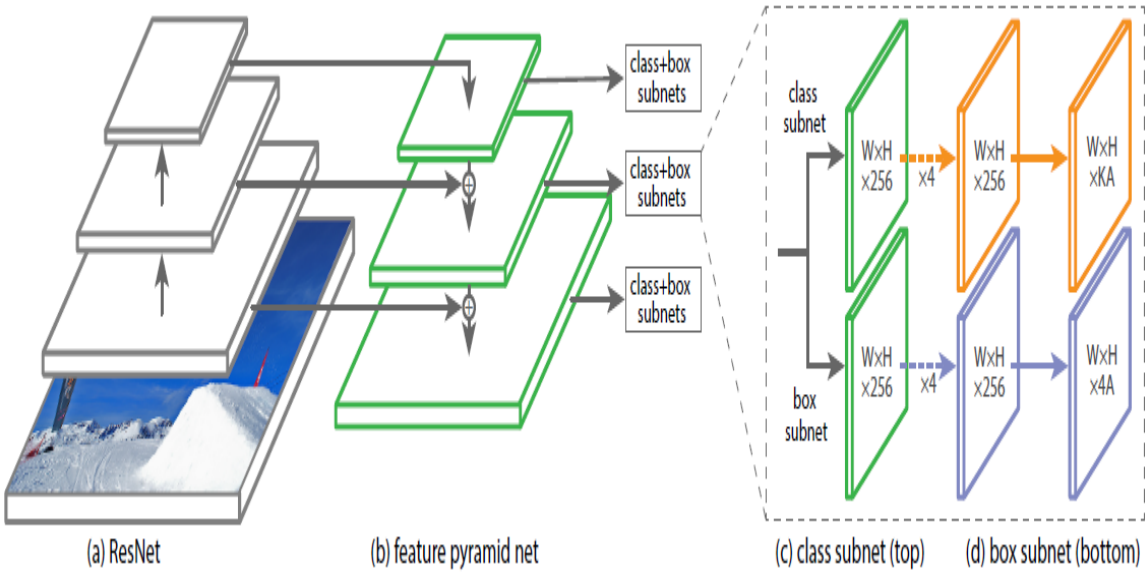

Рис. 3.11. Як працює архітектура *RetinaNet*

3.2. Порівняння продуктивності архітектур *CNN*

Середня точність або «оцінка *mAP*» є поширеним методом, який використовується для оцінки конкурсів виявлення об’єктів, таких як *PASCAL VOC, ImageNet* і *COCO*.

Під час виявлення об’єктів оцінка дуже важлива, оскільки існують дві різні задачі вимірювання:

– визначити, чи є на зображенні об’єкт (класифікація);

– визначити місце розташування об’єкта (локалізація, регресійні завдання).

Крім того, типовий набір даних матиме багато класів, і їх розподіл не є рівномірним (наприклад, собак може бути набагато більше, ніж морозива). Тому прості вимірювання, засновані на точності, вносять упередження. Також важливо оцінити ризик неправильної класифікації. Таким чином, необхідно пов’язати «індекс довіри» або оцінку моделі з кожним виявленим граничним вікном і оцінити модель з різними рівнями довіри [13].

Щоб обчислити *AP* для певного класу (наприклад, «людина»), крива точності-пригадування обчислюється на основі виявлень моделі шляхом зміни порогового значення оцінки моделі, що визначає позитивні виявлення для класу, передбаченого моделлю.

Останнім кроком у розрахунку оцінки *AP* є отримання середнього значення точності для всіх значень виклику. Це стає одним значенням, яке підсумовує форму кривої точності виклику. З цією метою оцінка *AP* однозначно визначається як середня точність набору з 11 рівновіддалених значень запам'ятовування, *Recall_i* = [0, 0,1, 0,2, ..., 1,0].

$$AP = \frac{1}{11} \sum_{\text{Recall}_i} \text{Precision}(\text{Recall}_i)$$

Точність відкликання *i* вважається максимальною точністю, виміряною, коли відкликання перевищує *Recall_i* [13].

Поки що були описані лише завдання класифікації. Для компонента локалізації необхідно враховувати ступінь перекриття між частиною зображення, яку модель сегментує на реальну, та частиною зображення, де фактично розташований об'єкт.

Для задоволення цих потреб було введено *Average Precision* (*AP*). Щоб зрозуміти *AP*, потрібно розуміти точність і пам'ятати про класифікатори. Простіше кажучи, у цьому випадку точність вимірює «коефіцієнт помилкових позитивних результатів» або відношення справжніх виявлень об'єктів до загальної кількості об'єктів, передбачених класифікатором. Якщо ваша точність близька до 1,0, є хороший шанс, що те, що класифікатор передбачив як позитивне виявлення, насправді було правильним прогнозом. Нагадаємо, що він вимірює «коефіцієнт помилкових негативних результатів» або відношення справжніх виявлень об'єктів до загальної кількості об'єктів у наборі даних. Якщо ваше відкликання наближається до 1.0, модель виконає позитивні виявлення майже для всіх об'єктів у вашому наборі даних. Нарешті, дуже важливо зазначити, що існує зворотний зв'язок між точністю та запам'ятовуванням, і що ці показники залежать від встановленого вами

порогу оцінки (і, звичайно, якості моделі). Наприклад, на цьому зображенні з *TensorFlow Object Detection API*, якщо ми встановимо порогове значення оцінки моделі для об'єкта змії на 50%, Ми отримаємо 7 позитивних виявлень, але якщо ми встановимо поріг оцінки моделі на 90%, буде 4 позитивних виявлення.

Для того, щоб оцінити модель для завдання локалізації об'єкта, спочатку слід визначити, наскільки добре модель передбачає розташування об'єктів. Зазвичай це робиться шляхом малювання обмежувальної рамки навколо цікавого об'єкта, але в деяких випадках це *N*-сторонній багатокутник або навіть попіксельна сегментація. Для всіх цих випадків завдання локалізації зазвичай оцінюються за пороговим значенням перетину через об'єднання (*IoU*). Для повноти решти цієї роботи припускається, що модель передбачає обмежувальні прямокутники, але практично все сказане також стосується сегментації пікселів або *N*-сторонніх багатокутників. Є багато хороших пояснень для *IoU*, але основна ідея полягає в тому, що він узагальнює, наскільки наземні об'єкти правди перекриваються з межами об'єктів, передбаченими моделлю.

Визначення об'єктів моделі визначаються як істинні або хибні на основі порогового значення *IoU*. Цей поріг *IoU* змінюється для кожного змагання, але, наприклад, завдання *COCO* розглядає 10 різних порогових значень *IoU* в діапазоні від 0,5 до 0,95 з кроком 0,05 [13].

На наступному рисунку показано порівняння продуктивності архітектури *Faster R-CNN, YOLO, SSD* і *RetinaNet* на основі набору даних *COCO*.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5
SSD513	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Рис. 3.12. Порівняння продуктивності архітектур *CNN*

Проаналізувавши результати, можна сказати, що серед двоетапних методів архітектура *Faster R-CNN Top-Down Modulation* досягає найвищого рейтингу за точністю розпізнавання зображень. Серед однокрокових методів найкраще працює архітектура *RetinaNet*.

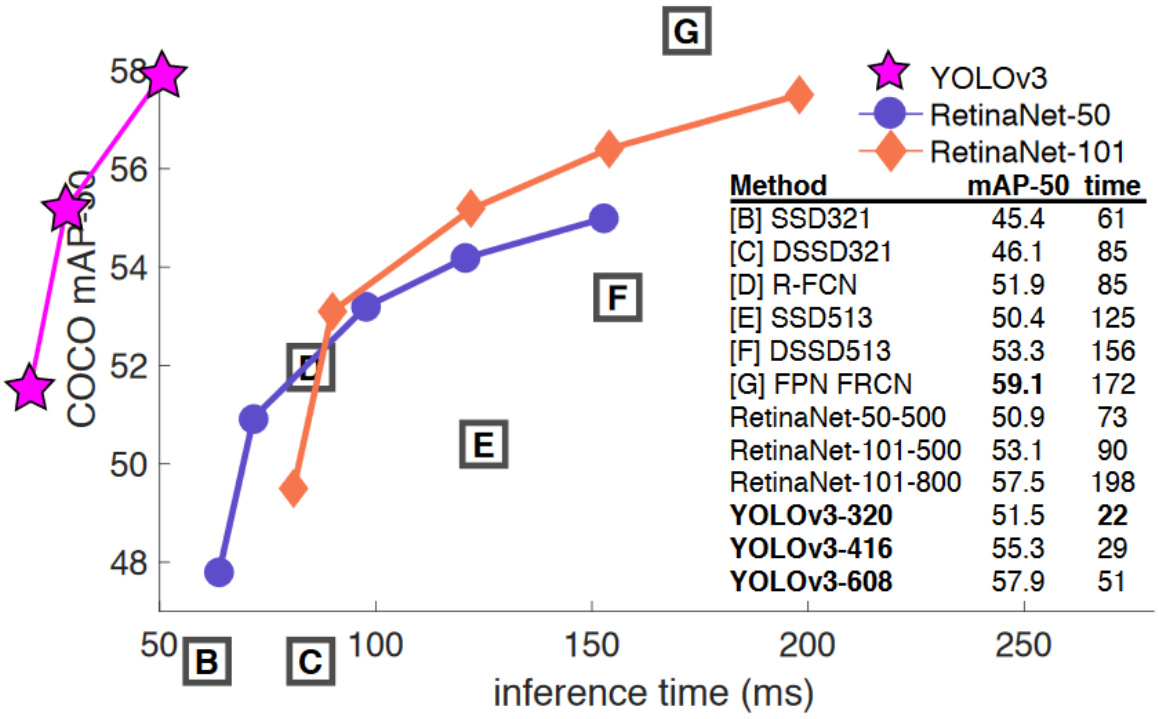

Рис. 3.13. Відношення точності розпізнавання до часу висновку архітектури *CNN*

Інший графік показує залежність між точністю розпізнавання зображення та часом, протягом якого архітектура зробить висновки щодо елементів зображення. Враховуючи це співвідношення та той факт, що архітектура *YOLOv*3 обробляє зображення в 3,8 рази швидше, ніж *RetinaNet*, програма розпізнавання зображень обрала архітектуру *YOLOv*3 [14].

ВИСНОВКИ ДО РОЗДІЛУ 3

Розділ 3 описує особливості популярних архітектур *CNN*: *R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN, YOLO, SSD, Feature Pyramid Network* (*FPN*), *RetinaNet*. Розглянуто складові елементи та принципи роботи цих архітектур.

Проаналізовано продуктивність архітектури *CNN* та співвідношення точності розпізнавання до часу висновку.

Архітектуру *YOLOv*3 було обрано, тому що вона має хорошу точність до співвідношення часу висновку, обробляє зображення швидше, ніж інші архітектури *CNN*, добре працює в обробці зображень у реальному часі, а передбачення (розташування об’єктів і класи) створюються з однієї мережі. Методи пропозиції регіону обмежують класифікатор певним регіоном. *YOLO* обробляє все зображення в режимі прогнозування меж. У додатковому контексті *YOLO* демонструє менше хибних спрацьовувань у фонових регіонах, а також застосовує просторову різноманітність у прогнозах.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

4.1. Вимоги до програмного та апаратного забезпечення

– *Microsoft Windows* 10/8/7 (64-розрядна), *Mac OS X, Linux*;
– не менше 1 Гб оперативної пам'яті;
– рекомендовано 2 ГБ оперативної пам’яті;
– мінімальний розподіл екрана 1024*x*768;
– *python*3.8;
– бібліотека *OpenCV Python*;
– інтегроване середище розробки *PyCharm*.

4.2. Вибір середовища розробки

Інтегроване середовище розробки (*ISR*) — від *Integrated Development Environment* (також може тлумачитися як *Integrated Design Environment* — інтегроване середовище проектування або *Integrated Debugging Environment* — інтегроване середовище налагодження) — це комп’ютер, який допомагає програмістам розробляти нове програмне забезпечення або змінювати (покращувати) існуюче програмне забезпечення. процедура [15].

Інтегровані середовища розробки зазвичай включають редактори вихідного коду, компілятори або інтерпретатори, інструменти автоматизації побудови та аналізу в цілому, а також різні інструменти та утиліти для спрощення розробки *GUI*. Багато сучасних *IDE* також містять браузер класів,

інспектор об’єктів та діаграму ієрархії класів для розробки програмного забезпечення з використанням об’єктно-орієнтованого підходу. Сучасні *ISR* часто підтримують розробку кількома мовами програмування.

PyCharm — це найрозумніше середовище розробки *Python* із повним набором інструментів для ефективної розробки на *Python*. Він доступний у двох варіантах: це безкоштовна версія *PyCharm Community Edition* і підтримує більше функцій *PyCharm Professional Edition*. *PyCharm* виконує перевірку коду на льоту, автозаповнення, включаючи навігацію по коду на основі інформації, отриманої під час виконання коду, пропонує багато рефакторингів [16].

Основні риси:

– потужний редактор коду, який підтримує підсвічування синтаксису мови, автоматичне форматування та автоматичний відступ;

– проста та потужна кодова навігація;

– допомога з написанням коду, включаючи автозавершення, автоматичний імпорт, шаблони коду, перевірки сумісності версії інтерпретатора мови тощо;

– швидкий перегляд документації для будь-якого елемента безпосередньо у вікні редактора, перегляд поданої документації через браузер, підтримка рядків документації - генерація, підсвічування, автозаповнення тощо [17];

– багато перевірок коду;

– потужний рефакторинг коду, що надає широкі можливості для швидкого внесення глобальних змін у проект;

– повна підтримка останньої версії фреймворку *Django*;

– підтримка *Google App Engine*;

– підтримка *IronPython, Jython, Cython, PyPy wxPython, PyQt, PyGTK* тощо;

– підтримка фреймворку *Flask* і мов *Mako* і *Jinja*2;

– редактор *Javascript, Coffescript, HTML/CSS, SASS, LESS, HAML*;

– інтеграція з *Version Control System* (*VCS*);

– діаграми класів *UML*, діаграми моделей *Django* та *Google App Engine*;

– інтеграційні модульні тести;

– інтерактивні консолі для *Python*, *Django*, *SSH*, відладчиків і баз даних;

– повнофункціональний графічний відладчик (*Debugger*);

– підтримка найпопулярніших рішень *IDE*, таких як *Netbeans*, *Eclipse*, *Emacs*, симуляція *VIM*-редактора;

– мови: *Python* (версії: 2.*x*, 3.*x*), *Jython*, *Cython*, *IronPython*, *PyPy*, *Javascript*, *CoffeScript*, *HTML*/*CSS*, *Django*/*Jinja*2 *Templates*, *Gql*, *LESS*/*SASS*/*SCSS*/*HAML*, *Mako*, *Puppet*, Регулярні вирази, *Rest*, *SQL*, *XML*, *YAML*;

– *PyCharm* має кілька колірних схем і настроюване підсвічування синтаксису коду;

– інтеграція з трекерами помилок, такими як *JIRA*, *Youtrack*, *Lighthouse*, *Pivotal Tracker*, *GitHub*, *Redmine*, *Trac*;

– величезна, постійно оновлювана колекція плагінів;

– кросплатформна сумісність (*Windows*, *Mac OS X*, *Linux*) [18].

4.3. Перелік застосовуваних технологій

Python — це інтерпретована об’єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблено Гвідо ван Россумом у 1990 році. Розширені структури даних разом із динамічною семантикою та динамічним зв’язуванням роблять його привабливим для швидкої розробки додатків і як засіб об’єднання існуючих компонентів. *Python* підтримує модулі та пакети модулів, які сприяють модульності та повторному використанню коду [19].

Інтерпретатор *Python* і стандартна бібліотека доступні у скомпільованій та вихідній формі на всіх основних платформах. Мова програмування *Python*

підтримує декілька парадигм програмування, зокрема: об’єктно-орієнтоване, процедурне, функціональне та аспектно-орієнтоване.

Його основні переваги:

– чистий синтаксис (для виділення блоків потрібно використовувати відступи);

– переносимість програми (характеристика більшості інтерпретованих мов);

– стандартний дистрибутив має велику кількість корисних модулів (включаючи один для розробки графічних інтерфейсів);

– вміння використовувати *Python* в розмовному режимі (корисно для експериментів і вирішення простих задач);

– стандартний дистрибутив має просте, але в той же час досить потужне середовище розробки під назвою *IDLE*, яка написана на мові *Python*;

– легко вирішувати математичні задачі (має інструменти для роботи з комплексними числами, може обробляти цілі числа будь-якого розміру, може використовуватися як потужний калькулятор у режимі розмови);

– відкрити код (інші користувачі можуть його редагувати) [20].

Python має ефективні високорівневі структури даних і простий, але ефективний підхід до об’єктно-орієнтованого програмування. Елегантний синтаксис *Python*, динамічна обробка типів і той факт, що це інтерпретована мова, роблять його ідеальним для створення сценаріїв і швидкої розробки додатків у багатьох галузях промисловості на більшості платформ.

Інтерпретатор мови *Python* і багата стандартна бібліотека (вихідні та двійкові дистрибутиви для всіх основних операційних систем) доступні на веб-сайті *Python* і вільно розповсюджуються. На цьому сайті є дистрибутиви та посилання на багато модулів, програм, утиліт та іншої документації.

Інтерпретатор мови *Python* можна розширити за допомогою функцій і типів даних, розроблених на *C* або *C++* (або інших мовах, які можна викликати з *C*). *Python* також зручний як мова розширення для програм, які потребують подальшого налагодження.

Python портований і доступний практично для всіх відомих платформ - від КПК до мейнфреймів. Доступно для *Microsoft Windows*, усіх варіантів *UNIX* (включаючи *FreeBSD* і *GNU/Linux*), *Plan* 9, *Mac OS* і *Mac OS X, iPhone OS* 2.0 і новіших версій, *Palm OS*, *OS*/2, *Amiga*, *AS*/400 і навіть *OS*/390 , *Symbian* і *Android* [20].

Оскільки платформа застаріла, її підтримку основних мовних гілок було припинено. Наприклад, підтримка *Windows* 95, *Windows* 98 і *Windows ME* припинена з версії 2.6. Однак на цих платформах можна використовувати попередні версії *Python* — спільнота активно підтримує версії *Python*, починаючи з 2.3 (для них випущено виправлення). Крім того, на відміну від багатьох систем портування, *Python* підтримує специфічні для платформи технології (наприклад, *Microsoft COM*/*DCOM*) для всіх основних платформ. Більше того, віртуальна машина *Java* також має спеціальну версію *Python*, *Jython*, яка дозволяє інтерпретатору працювати на будь-якій системі, яка підтримує *Java*, тоді як класи *Java* можна використовувати безпосередньо з *Python* або навіть писати на *Python*. Деякі проекти також забезпечують інтеграцію з платформою *Microsoft.NET*, головним чином *IronPython* і *Python.Net*.

Python підтримує динамічну типізацію, тобто тип змінної визначається лише під час виконання. Серед примітивних типів слід подбати про підтримку цілих і комплексних чисел довільної довжини. *Python* має багату бібліотеку для роботи з рядками, особливо з рядками, закодованими *Unicode*.

З наборів *Python* підтримує кортежі (кортежі), списки (масиви), словники (асоціативні масиви) і набори з версії 2.4.

Система класів підтримує множинне успадкування та метапрограмування. Будь-який тип, включаючи базові типи, є частиною системи класів і навіть може успадковувати базові типи, якщо необхідно.

Подібно *Lisp* і *Prolog* в режимі налагодження, інтерпретатор *Python* має інтерактивний режим роботи, в якому введені з клавіатури вирази виконуються негайно, а результати виводяться на екран. Цей режим цікавий

не тільки новачкам, а й досвідченим програмістам, які можуть протестувати будь-який фрагмент коду в інтерактивному режимі, а потім використовувати його в основній програмі або просто використовувати як багатофункціональний калькулятор.

Дизайн мови *Python* побудований навколо моделі об’єктно-орієнтованого програмування. Реалізація ООП у *Python* є елегантною, потужною та добре продуманою, але водночас дуже специфічною порівняно з іншими об’єктно-орієнтованими мовами.

Особливості та особливості:

– клас також є об’єктом, який має всі наступні функції;

– успадкування, в тому числі множинне;

– поліморфізм (всі функції віртуальні);

– інкапсуляція (два рівні - відкритий і прихований методи і поля). Можливість - приховані учасники доступні та позначені як приховані лише спеціальним іменем;

– спеціальні методи управління життєвим циклом об'єктів: конструктори, деструктори, розподільники пам'яті;

– перевантаження операторів (крім *is*, '.', '=' і символічної логіки);

– властивості (живі макети з функціями);

– керувати доступом до полів (макет полів і методів, частковий доступ тощо);

– методи керування найпоширенішими операціями (*truth, len*(), *deep copy, serialization, object iteration*...);

– метапрограмування (керування створенням класів, тригери створення класів тощо);

– повна самоаналіз;

– класи та статичні методи, поля класів;

– класи, вкладені в функції та інші класи [20].

Python підтримує парадигми функціонального програмування, зокрема:

– функція - це об'єкт;

– функції вищого порядку;

– рекурсія;

– розвинена обробка списків (спискові вирази, операції послідовності, ітератори);

– аналоги затворів;Часткове застосування цієї функції;

– можливість реалізації інших способів (наприклад, каррінг) у самій мові.

Програмне забезпечення (додатки або бібліотеки) на *Python* розроблено в модулях, які, у свою чергу, можуть бути зібрані в пакети. Модулі можуть бути розташовані або в каталогах, або в *ZIP*-архівах. Модулі можна розділити на два типи: модулі, написані на «чистому» *Python*, і модулі розширення (модулі розширення), написані на інших мовах програмування. Наприклад, стандартна бібліотека має «чистий» модуль *pickle* і його аналог *C*: *cPickle*. Модулі публікуються як окремі файли, а пакети – як окремі каталоги. Модуль зв'язується з програмою за допомогою оператора імпорту. Після імпортування модуль представляється окремим об’єктом, який має доступ до простору імен модуля. Під час виконання програми модулі можна перезавантажувати за допомогою функції *reload*().

Python підтримує повну самоаналіз під час виконання. Це означає, що для будь-якого об’єкта можна отримати всю інформацію про його внутрішню структуру [20].

Використання інтроспекції (метапрограмування) є важливою частиною так званого «стилю *Python*» і широко використовується в бібліотеках і фреймворках *Python*, таких як *PyRO*, *Pyro*, *PLY*, *CherryPy*, *Django* тощо, заощаджуючи час програмістів на їх використання .

Python підтримує обробку винятків, використовуючи оператори *try*, *osim*, *else* і *finally*, які формують блоки обробки винятків.

Ітератори широко використовуються в програмах *Python*. цикли *for* можна використовувати з послідовностями та ітераторами. Усі колекції зазвичай забезпечують ітератор. Об'єкти визначених користувачем класів

також можуть бути ітераторами. Модуль *itertools* стандартної бібліотеки містить багато корисних функцій для роботи з ітераторами. На відміну від звичайної послідовності, де всі її елементи зберігаються в пам'яті, отримання наступного елемента забезпечує генератор – спеціальна функція, доступ до якої обчислює та повертає наступний елемент генератора.

Цікавою особливістю мови є генератори - функції, які зберігають внутрішній стан між викликами: значення локальних змінних і поточну інструкцію (див. також: підпрограми). Генератори можна використовувати як ітератори для структур даних і відкладених обчислень.

У *Python* 2.5 з'явилися інструменти для управління контекстом виконання блоків коду - оператор *with* і модуль *contextlib*.

Цей оператор можна використовувати, коли потрібно виконати якусь іншу операцію до та після певної операції, незалежно від винятків, створених у блоці чи операторі повернення: файл має бути закрито, ресурси мають бути звільнені, стандартне перенаправлення введення/виведення тощо. Оператори полегшують читання коду, що допомагає уникнути помилок.

Починаючи з версії 2.4, *Python* дозволяє використовувати так звані декоратори (не плутати з однойменним шаблоном проектування) для підтримки існуючої практики перетворення функцій і методів у точці визначення (може бути кілька декораторів). Після довгих дебатів декоратори почали використовувати символ @ у рядку перед визначенням функції або метод [21].

Python, як і багато інших інтерпретованих мов, які не використовуються, наприклад *JIT*-компілятори, має загальний недолік - відносно повільну швидкість виконання програми. Однак у випадку з *Python* цей недолік можна компенсувати скороченням часу розробки програми. У середньому програми, написані на *Python*, у 2-4 рази компактніші за програми, написані на *C++* або *Java*. Збереження байт-коду (файли *.pyc* і *.pyo*) дозволяє інтерпретатору не витрачати додатковий час на перекомпіляцію коду модуля щоразу під час його запуску, на відміну від

мови *Perl*. Також є спеціальна бібліотека *JIT psyco* (проте вона призводить до збільшення споживання оперативної пам'яті). Ефективність *psycho* багато в чому залежить від архітектури програми [16].

Відсутність статичної типізації є не стільки недоліком інтерпретатора, скільки вибором розробників мови. Насправді *Python* використовує те, що називається «качиним набором». Через це тип переданого значення недоступний під час компіляції, і під час виконання можуть виникати такі помилки, як *AttributeError*. Відсутність статичної типізації також є однією з головних причин низької продуктивності.

Порівняно з *Ruby* та деякими іншими мовами, *Python* не має можливості змінювати вбудовані класи, такі як *int, str, float, list* тощо, але це змушує *Python* споживати менше оперативної пам’яті та працювати швидше. Ще однією причиною введення цього обмеження є необхідність узгодження з модулями розширення. Багато модулів (з метою оптимізації продуктивності) перетворюють примітивні типи об’єктів *Python* у відповідні їм типи *C*, а не використовують *C API* для маніпулювання ними.

GIL (*Global Interpreter Lock*) є внутрішньою проблемою для *CPython*, *Stackless* і *PyPy*, але не для *Jython* і *IronPython*. У своїй роботі основні інтерпретатори *Python* постійно використовують великі обсяги даних про ризик потоку.

В основному це словники, які зберігають властивості об'єктів. Щоб уникнути пошкодження цих даних під час спільної модифікації різними потоками, потік інтерпретатора захоплює *GIL* перед початком виконання кількох інструкцій (за замовчуванням 100) і відпускає його після завершення. Завдяки цій функції лише один потік коду *Python* може виконуватися в будь-який момент часу, навіть якщо на комп’ютері є кілька процесорів або процесорних ядер (*GIL* звільняється під час блокування операцій, таких як введення/виведення), змініть/перевірте стан синхронізації примітиви тощо - тому, якщо один потік блокується, інші потоки можуть виконуватися). Спробував перейти до більш детальної синхронізації, але ця реалізація

виявилася надто повільною через часте отримання/розблокування. Найближчим часом переходу від *GIL* до інших технологій не очікується, але є *python-safethread - CPython* без *GIL* і з деякими іншими змінами (за словами його авторів, швидкість однопоточної програми відповідає 60- 65% швидкості оригінального *CPython*) [20].

Є два основних рішення цієї проблеми. По-перше, це відмова ділитися даними, що змінюються. У той же час дані повторюються між потоками, і необхідність забезпечити їх синхронізацію (якщо потрібно) лягає на програміста. Такий підхід призводить до збільшення споживання оперативної пам'яті (правда, не так сильно, як при використанні процесів).

Другий підхід полягає в забезпеченні більш тонкої синхронізації окремих структур даних. У цьому випадку продуктивність знизиться через збільшення кількості звільнень/отримання блокувань.

Якщо вам потрібно виконати декілька потоків коду *Python* паралельно, ви можете використовувати процеси, наприклад модуль обробки, який імітує семантику стандартного модуля потоків, але використовує процеси замість потоків. Існує багато модулів, які полегшують написання паралельних і/або розподілених програм на *Python*, наприклад *parallelpython*, *Pypar*, *pympi* тощо. *GIL* звільняється, коли виконується код більшості розширень (таких як *NumPy*/*SciPy*), що дозволяє іншому потоку *Python* виконувати обчислення. Іншим рішенням може бути використання *IronPython* або *Jython*, які не мають цього недоліку [20].

OpenCV (Бібліотека комп’ютерного бачення з відкритим вихідним кодом) — це бібліотека функцій і алгоритмів з відкритим кодом для комп’ютерного зору, обробки зображень і загальних чисельних алгоритмів. *OpenCV* має на меті створити спільну інфраструктуру для програм комп’ютерного бачення та прискорити використання машинного сприйняття в комерційних продуктах. Як продукт з ліцензією *BSD*, *OpenCV* спрощує використання та модифікацію коду підприємствами [22].

Бібліотека містить понад 2500 алгоритмів оптимізації, включаючи повний набір класичних і сучасних алгоритмів комп'ютерного зору та машинного навчання. Ці алгоритми можна використовувати для виявлення та розпізнавання облич, ідентифікації об'єктів, класифікації дій людей у відео, відстеження руху камери, відстеження рухомих об'єктів, вилучення 3*D*-моделей об'єктів, отримання 3*D*Хмари точок зі стереокамер, з'єднання зображень, щоб отримати зображення всієї сцени з високою роздільною здатністю, пошук схожих зображень у базі даних зображень, видалення ефекту червоних очей із зображень зі спалахом, відстеження рухів очей, ідентифікація пейзажів і встановлення маркерів для накладання покращення *Reality* , тощо. Спільнота *OpenCV* налічує понад 47 000 користувачів, а кількість завантажень становить понад 18 мільйонів. Ця бібліотека широко використовується компаніями, науковими групами та державними установами.

На додаток до відомих компаній, таких як *Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota* тощо, які використовують бібліотеку, є багато стартапів, таких як *Applied Minds, VideoSurf* і *Zeitera*, які широко використовують *OpenCV*. Розгорнуті використання *OpenCV* включають зшивання зображень *Street View*, виявлення вторгнень у відео з ізраїльських камер спостереження, моніторинг шахтного обладнання в Китаї, допомогу роботам у навігації та підбиранні об'єктів у *Willow Garage*, виявлення потопельників у басейнах у Європі, запуск інтерактивного мистецтва в Іспанії та Нью-Йорку, у Туреччині перевіряє злітно-посадкові смуги на наявність сміття, а заводи по всьому світу сканують етикетки продуктів для швидкого розпізнавання обличчя в Японії.

OpenCV має інтерфейси *C++, Python, Java* і *MATLAB* і підтримує *Windows, Linux, Android* і *MacOS*. *OpenCV* насамперед орієнтований на програми бачення в реальному часі та використовує інструкції *MMX* та *SSE*, якщо вони доступні. Повнофункціональні інтерфейси *CUDA* та *OpenCL* зараз активно розробляються. Існує понад 500 алгоритмів, а функцій, які їх

складають або підтримують, приблизно в 10 разів більше. *OpenCV* написано мовою *C++* і має інтерфейс шаблону, який можна легко використовувати з контейнерами *STL* [22].

4.4. Розробка програмного коду

На основі розглянутих засобів і моделей необхідно розробити систему розпізнавання зображень.

Першим кроком є завантаження готового, попередньо навченого файлу *.weights* набору даних *COCO* для мережі *YOLOv3*, файлу конфігурації для тієї ж мережі та файлу з мітками *COCO* (тип зображень, на яких навчається нейронна мережа, і він здатний розпізнавати). Після завантаження цих файлів вам потрібно зберегти їх у папці *raw* вашого проекту для подальшого використання.

Оскільки взаємодія користувача з програмою відбуватиметься за допомогою командного рядка, необхідно додати програмний блок, який буде відповідати за інтерпретацію команд, отриманих від користувача.

```
parser = argparse.ArgumentParser()
```

Цей об’єкт аналізує рядки з командного рядка та перетворює їх на об’єкти *Python*. Додамо наступний блок:

```
parser.add_argument('-i', '--image-path',
                    type=str,
                    help='The path to the image file')

parser.add_argument('-v', '--video-path',
                    type=str,
                    help='The path to the video file')

parser.add_argument('-c', '--confidence',
                    type=float,
                    default=0.5,
                    help='The model will reject boundaries which has a \
                probabiity less than the confidence value. \
                default: 0.5')

parser.add_argument('-th', '--threshold',
                    type=float,
                    default=0.3,
                    help='The threshold to use when applying the \
                Non-Max Suppresion')
```

– команда *image-path* відповідає за визначення шляху до файлу зображення об'єкта;

– команда *video-path* відповідає за визначення шляху до відеофайлу об'єкта;Порядок достовірності відповідає за фільтрацію мінімальної ймовірності слабких виявлень. Елементи з імовірністю нижче заданої не будуть виділені;

– команда *threshold* відповідає за немаксимальні обмежені пороги. Суть цього параметра полягає в тому, що мережа лише один раз вибирає об’єкт на зображенні, вибирає один із кількох прямокутників з найвищою ймовірністю та відсікає інші прямокутники, які його перекривають;

Інша команда завантажить *.weights* і *.cfg* мережі *YOLO*:

net=cv.dnn.readNetFromDarknet('*./yolo_model/yolov*3*.cfg*','*./yolo_model/yolov*3*.weights*')

Наведений нижче блок коду відповідає потоку програми, коли вводиться команда зі шляхом зображення. Спочатку зчитується зображення, і якщо зображення не знайдено у введеному каталозі, у командному рядку відображається повідомлення про помилку. Далі зображення буде перетворено у формат, який можна надіслати на вхід нейронної мережі *YOLO*. З вихідного рівня нейронної мережі отримують елементи, навколо яких малюються прямокутники з ймовірностями та мітками *COCO*, і лише потім до зображення наноситься немаксимальний обмежений пороговий параметр.

```
if FLAGS.image_path:
    try:
        img = cv.imread(FLAGS.image_path)
        height, width = img.shape[:2]
    except:
        raise Exception('[ERROR] Image cannot be loaded! Please check the path provided!')
    finally:
        img, _, _, _, _ = infer_image(net, layer_names, height, width, img, colors, labels, FLAGS)
        print("[INFO] Image processing finished!")
        show_image(img)
```

Зображення з веб-камери обробляються так само, як і відеофайли, але відображаються безперервно в реальному часі, поки користувач не натисне кнопку, щоб закрити програму.

```
try:
    vid = cv.VideoCapture(FLAGS.video_path)
    height, width = None, None
    writer = None
except:
    raise Exception('[ERROR] Video cannot be loaded! Please check the path provided!')
finally:
    while True:
        grabbed, frame = vid.read()
        if not grabbed:
            break
        if width is None or height is None:
            height, width = frame.shape[:2]
        frame, _, _, _, _ = infer_image(net, layer_names, height, width, frame, colors, labels, FLAGS)
        if writer is None:
            fourcc = cv.VideoWriter_fourcc(*"MJPG")
            writer = cv.VideoWriter('./output_files/output.avi', fourcc, 30,
                                    (frame.shape[1], frame.shape[0]), True)
        writer.write(frame)
    print("[INFO] Video processing finished!")
    writer.release()
    vid.release()
```

4.5. Тестування програмних комплексів

Для завантаження та обробки зображень за допомогою розробленої системи введіть у терміналі або командному рядку операційної системи таку команду: «*python yolo.py -i=input_files/image1.jpg*», де останнім параметром є каталог зображення.

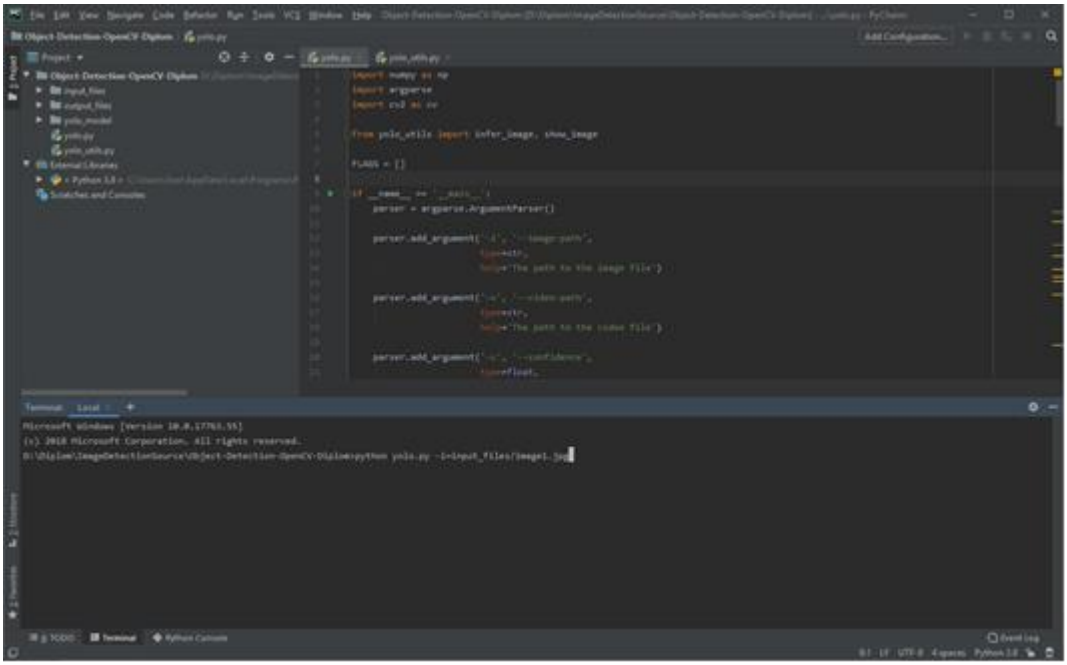

Рис. 4.1. Термінал *IDE PyCharm*

Після завершення обробки зображення програма виводить у командний рядок інформацію про час обробки зображення, список знайдених об’єктів з імовірністю від 0 до 1 і коротке повідомлення про виконання програми.

```
[INFO] Image processing finished!

D:\Diplom\ImageDetectionSource\Object-Detection-OpenCV-Diplom>python yolo.py -i=input_files/image2.jpg
[INFO] Time taken to infer image: 1.259918 seconds
[RESULTS] dog: 0.989900
[RESULTS] cat: 0.722426
[INFO] Image processing finished!

D:\Diplom\ImageDetectionSource\Object-Detection-OpenCV-Diplom>python yolo.py -i=input_files/nau4.jpg
[INFO] Time taken to infer image: 1.289916 seconds
[RESULTS] aeroplane: 0.988926
[RESULTS] person: 0.938847
[INFO] Image processing finished!
```

Рис. 4.2 Термінал *IDE PyCharm*

Оброблене зображення також відкриється і виглядатиме так:

Рис. 4.3. Зображення, оброблені за допомогою розроблених програм

Після завантаження та обробки 100 зображень у таблиці 4.1 можна спостерігати наступні результати.

Таблиця 4.1.

Статистика роботи програми

К-сть оброблених зображень	Середня к-сть знайдених об’єктів	Середня к-сть усіх виявлених об’єктів на зображенні, які можна зафіксувати	Середній час обробки зображення (секунд)
10	5	6	1,21
20	5,4	6,6	1,34
100	6,1	7,8	1,57

Проаналізувавши отримані результати, можна сказати, що розроблений програмний модуль розпізнає більше 78% об’єктів зображення (об’єкти, які не входять до переліку об’єктів для навчання цієї нейронної мережі, не враховуються).

Щоб розроблена програма отримувала більш точні результати, при виборі зображень необхідно дотримуватися наступних вимог:

– зображення мають бути максимально високої роздільної здатності;

– об'єкт, який розпізнається, повинен узгоджуватися з об'єктом для навчання нейронної мережі;

– об'єкти, що ідентифікуються, повинні бути відокремлені один від одного;

– обробка зображень повинна підкреслювати об’єкти, що розпізнаються, а не спотворювати їх.

Щоб завантажити та обробити відеофайли за допомогою розробленої системи, просто зайдіть у термінал або командний рядок операційної системи та введіть наступне Команда: "*python yolo.py -v=input_files/false.mp*4", де

останнім параметром є каталог відеофайлу. Відеофайли, оброблені програмою, зберігаються в папці «*output_files*» у папці проекту.

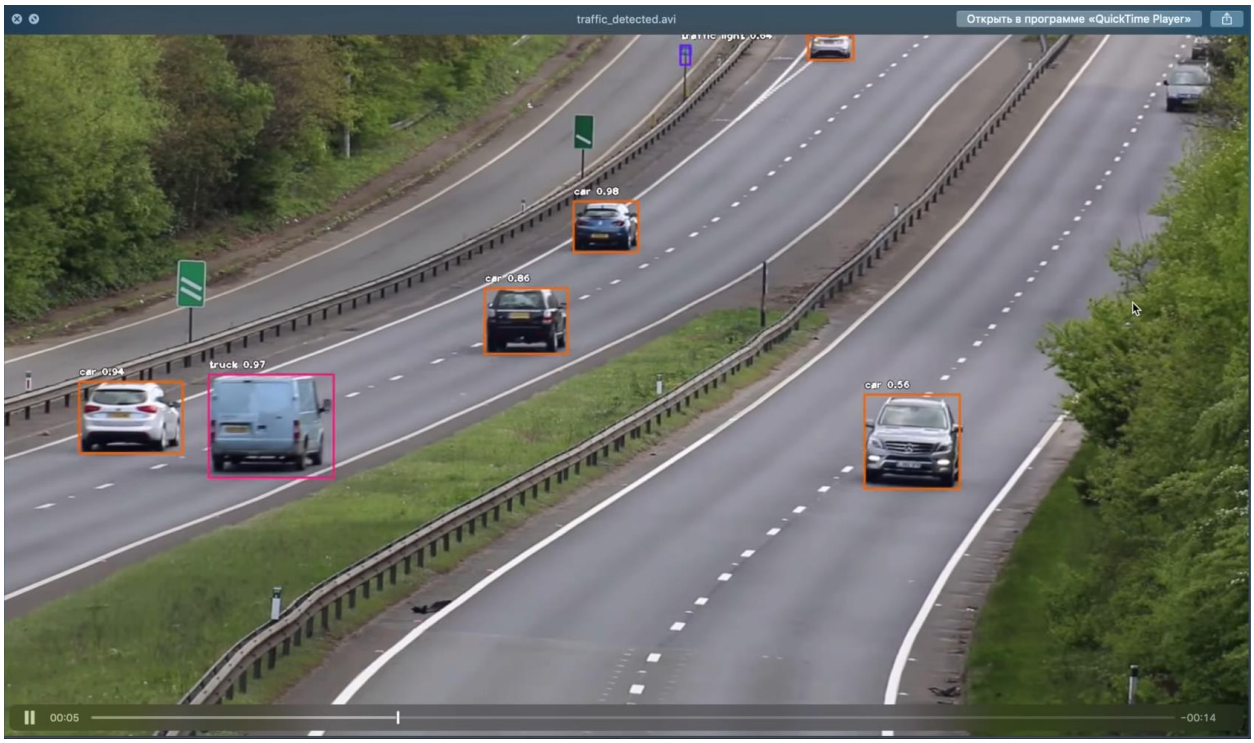

Рис. 4.4 Відеофайли, оброблені за допомогою розробленої програми

Однак обробка відеофайлів займає значну кількість часу. Наприклад, обробка 5-хвилинного відеофайлу 640*x*360, 24 кадри в секунду займає 34 хвилини за допомогою архітектури нейронної мережі *YOLOv*3.

ВИСНОВКИ ДО РОЗДІЛУ 4

Метою цього розділу є розробка програмного забезпечення для розпізнавання зображень за допомогою нейронних мереж.

У цьому розділі розділено вимоги до програмного забезпечення, описано використовуваний стек технологій і розроблено програмне забезпечення для розпізнавання зображень за допомогою нейронних мереж.

Програмний продукт реалізовано в інтегрованому середовищі розробки *PyCharm Community*.

Під час тестування програмних модулів була проаналізована робота сотень різних зображень. Програмний модуль розпізнає більше 78% об'єктів на зображенні (об'єкти, що не входять до переліку об'єктів для навчання цієї нейронної мережі, не враховуються). Отримані результати не ідеальні, але швидкість обробки файлів зображень і відео вкупі з мультиплатформенністю програми, простим і зручним інтерфейсом програми в командному рядку і можливістю переходу на інші версії *YOLO Neural Networks* переважує всі інші недоліки.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи були вирішені такі завдання:

– огляд сучасних програмних комплексів розпізнавання образів;

– переглянуто застосування нейронної мережі в розпізнаванні образів;

– досліджено методи розпізнавання образів;

– проаналізовано стандарти якості розпізнавання образів;

– розглянуто типи нейронних мереж;

– розглянуто методи навчання нейронних мереж;

– проаналізувано та обрано архітектури *CNN* для розпізнавання зображень;

– виконано порівняння продуктивності архітектури *CNN*;

– проаналізовано програмні та апаратні вимоги до розроблених програмних модулів;

– обрано середовища розробки та технології, які будуть використовуватися в розроблених програмних модулях;

– розроблено програмний код у інтегрованому середовищі розробки *PyCharm*;

– програмні модулі тестувалися на файлах зображень і відео.

У кваліфікаційній роботі досіджуються концепції нейронних мереж, глибокого навчання, згорткових нейронних мереж та інтегрованих середовищ розробки. Існує також потреба розділити поняття комп’ютерного зору — галузі досліджень, спрямованої на те, щоб допомогти комп’ютерам побачити проблеми. Це мультидисциплінарна сфера, яку можна загалом назвати підсферою штучного інтелекту та машинного навчання, яка може передбачати використання спеціалізованих методів і використання алгоритмів навчання загального призначення.

Реалізація системи комп'ютерного зору багато в чому залежить від сфери її застосування. Деякі системи є самодостатніми та вирішують конкретні проблеми виявлення та вимірювання, тоді як інші складаються з підсистем Більші системи, наприклад, можуть містити підсистеми керування,

такі як механічні маніпулятори, планування, інформаційні бази даних, людино-машинні інтерфейси тощо.

Розділ 1 аналізує питання в галузі досліджень, розглядає портфоліо сучасних програм для розпізнавання образів і описує методи розпізнавання образів. Розрізняють дві групи методів розпізнавання образів: методи, які використовують матриці навчання класифікації для формулювання правил прийняття рішень, і методи автоматичної класифікації (методи навчання без нагляду), які здатні працювати з некласифікованими даними.

І проаналізували стандарт якості розпізнавання образів.

Розділ 2 містить огляд різних типів нейронних мереж: нейронні мережі прямого поширення. Багатошарові відсотки, згорткові нейронні мережі, рекурентні нейронні мережі, модульні нейронні мережі та методи їх навчання.

Виходячи з розглянутої моделі нейронної мережі, згорткові нейронні мережі вибрано тому, що завдяки своїй структурі вони можуть дуже добре витягувати ознаки із зображень, кількість ваг коригування набагато менша, оскільки одне ядро ваг використовується повністю для всього зображення, а не визначаючи індивідуальні вагові коефіцієнти для кожного пікселя вхідного зображення.

Розділ 3 описує особливості популярних архітектур *CNN*: *R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN, YOLO, SSD, Feature Pyramid Network* (*FPN*), *RetinaNet*. Розглянуто складові елементи та принципи роботи цих архітектур. Проаналізовано продуктивність архітектури *CNN* та співвідношення точності розпізнавання до часу висновку.

Архітектуру *YOLOv3* було обрано, тому що вона має хорошу точність до співвідношення часу висновку, обробляє зображення швидше, ніж інші архітектури *CNN*, добре працює в обробці зображень у реальному часі, а передбачення (розташування об’єктів і класи) створюються з однієї мережі. Методи пропозиції регіону обмежують класифікатор певним регіоном. *YOLO* обробляє все зображення в режимі прогнозування меж. У додатковому

контексті *YOLO* демонструє менше хибних спрацьовувань у фонових регіонах, а також застосовує просторову різноманітність у прогнозах.

Метою розділу 4 є розробка програмного забезпечення програмного комплексу для розпізнавання об’єктів на зображенні.

Програмний продукт реалізовано в середовищі візуального програмування *PyCharm Community*.

Тестування програмних модулів виконується на сотнях відібраних зображень, що містять зображення для навчання нейронної мережі. Програмний модуль розпізнає більше 78% об'єктів на зображенні (об'єкти, що не входять до списку об'єктів для навчання даної нейронної мережі, не враховуються). Отримані результати не ідеальні, але швидкість обробки файлів зображень і відео вкупі з мультиплатформенністю програми, простим і зручним інтерфейсом програми в командному рядку і можливістю переходу на інші версії *YOLO Neural Networks* переважує всі інші недоліки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Комп'ютерний зір. - [Електронний ресурс] – Режим доступу до ресурсу: *https://en.wikipedia.org/wiki/Computer_vision*, вільний.

2. Короткий вступ до комп’ютерного зору. - [Електронний ресурс] – Режим доступу до ресурсу: *https://machinelearningmastery.com/what-is-computer-vision/*, вільний.

3. Гудфелов І. А. Глибинне навчання / Гудфелов І. А. // *MIT Press*. - 2016. - №2. - і. 50. – Бібліогр.: с. 140.

4. Основи нейронної мережі. - [Електронний ресурс]– Режим доступу до ресурсу: *https://medium.com/datadriveninvestor/neural-network-fundamentals-1956a3000c24*, вільний.

5. Гесті Т. Елементи навчання статистиці. / *Hesti T, Tibshirani M*. - *Springer-Verlag*, 2009. - 746 с.- Бібліографія: с. 123.

6. Краснопоясовський А. С. Інформаційний синтез інтелектуальних систем управління: методи на основі функціональних і статистичних методів тестування / А. С. Краснопоясовський. (Суми: Вид-во СумДУ, 2004. (261 *с*. – Бібліогр.: с. 76).

7. Айвазян С. А. Прикладна статистика: класифікація та зменшення обсягу: стаття. ред./ С. А. Айвазян, В. М. Бухштабер, І. С. Енюков, Л. Д. Мешалкін. В.М.: Фінанси і статистика, 1989. 607 с.

8. Вичерпний посібник із типів нейронних мереж. - [Електронний ресурс] . – Режим доступу до ресурсу: *https://www.digitalvidya.com/blog/types-of-neural-networks/*, вільний.

9. Початок роботи з *R-CNN, Fast R-CNN* і *Faster R-CNN*. - [Електронний ресурс] – Режим доступу до ресурсу: *https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html*, вільний.

10. *YOLO*: виявлення об'єктів у реальному часі. - [Електронний ресурс] – Режим доступу до ресурсу: *https://pjreddie.com/darknet/yolov2/*, вільний.

11. Виявлення об’єктів *SSD*: одноразовий детектор *MultiBox* для обробки в реальному часі. - [Електронний ресурс] – Режим доступу до ресурсу: *https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing*, вільний.

12. *Mask R-CNN*: сучасна архітектура нейронної мережі для сегментації об’єктів зображення. - [Електронний ресурс] - Режим доступу до ресурсу: *https://habr.com/ru/post/*421299/, вільний.

13. Дізнайтеся про показники оцінки *mAP* для виявлення об’єктів. - [Електронний ресурс] – Режим доступу до ресурсу: *https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a*07*fe*6962*cf*3, вільний.

14. *YOLOv*3: Поступові вдосконалення. - [Електронний ресурс] – Режим доступу до ресурсу: *https://pjreddie.com/media/files/papers/YOLOv*3*.pdf*, вільний.

15. Інтегроване середовище розробки. - [Електронний ресурс] – Режим доступу до ресурсу: *https://life-prog.ru/ukr/view_zam*2*.php?id=*103&*cat=*1&*page=*1, вільний.

16. Функції *PyCharm*. - [Електронний ресурс] – Режим доступу до ресурсу: *https://www.jetbrains.com/ru-ru/pycharm/features/*, вільний.

17. Особливості *PyCharm*. - [Електронний ресурс] – Режим доступу до ресурсу: *https://bmstu.cloud/docs/software/pycharm/*, вільний.

18. Опис продукту *PyCharm*. - [Електронний ресурс] – Режим доступу до ресурсу: *https://itpro.ua/product/jetbrains-pycharm/*, вільний.

19. Практичний *Python* 3 для початківців. - [Електронний ресурс] – Режим доступу до ресурсу: *https://pythonworld.ru/samouchitel-python/*, вільний.

20. Опис *Python*. - [Електронний ресурс] – Режим доступу до ресурсу: *https://uk.wikipedia.org/wiki/Python*, вільний.

21. Декоратор *Python*. - [Електронний ресурс] – Режим доступу до ресурсу: *https://tproger.ru/translations/demystifying-decorators-in-python/*, вільний.

22. Про *OpenCV*. - [Електронний ресурс] – Режим доступу до ресурсу: *https*://*opencv*.*org*/*about*/, безкоштовний.

23. ГОСТ 19.701-90 ЕСКД. Схеми алгоритмів, програм, даних і систем. Символи та правила застосування.

24. ГОСТ 19.003-80 ЕСКД. Алгоритми та схеми програм. Загальні графічні позначення.

25. Бойчено С В., Іванченко О. В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. Затверджено наказом т. в. о. ректора від 14.12.2017 № 594/од.