МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА
Факультет економіки і підприємництва
Кафедра інформаційних технологій

Допущено до захисту
"_____" _____ 2024 р.
Зав. кафедри, к.т.н., доцент
_____ **Ліщук Р.І.**

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ РОЗПІЗНАВАННЯ ЛІТАКІВ НА ЗОБРАЖЕННЯХ

Виконав – студент 2 курсу, групи 21м-кн
освітньої програми "Комп’ютерні науки"
спеціальності 122 Комп’ютерні науки
Галаган Михайло Михайлович

Науковий керівник – к.т.н., доцент
Ліщук Роман Ігорович

Умань – 2024

Зміст

ВСТУП 3

РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ 6

1.1. Сучасні програмні системи розпізнавання образів 6

1.2. Місце нейромереж в задачі розпізнаванні образів 11

1.3. Класичні методи для розпізнавання образів 17

1.4. Постановка задачі 22

РОЗДІЛ 2 ОГЛЯД НЕЙРОМЕРЕЖ ТА МЕТОДІВ ЇХ НАВЧАННЯ..... 23

2.1. Типи нейронних мереж 23

2.2. Навчання нейронних мереж..... 26

2.4. Обґрунтування архітектури системи розпізнавання зображень 34

РОЗДІЛ 3 РОЗРОБКА МОДУЛЯ ДЛЯ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ 45

3.1. Вибір середовища для розробки..... 45

3.2. Перелік використаних технологій..... 47

3.3. Розробка коду додатку..... 53

ВИСНОВКИ..... 60

СПИСОК ЛІТЕРАТУРИ..... 62

ДОДАТКИ..... 66

ВСТУП

З розвитком комп’ютерних технологій багато завдань, що виникають у процесі життя, можна вирішувати для полегшення, прискорення та підвищення якості результатів. Наприклад, робота різноманітних систем життєзабезпечення, взаємодія людини з комп’ютером, поява роботизованих систем. Однак слід зазначити, що через недосконалість розроблених алгоритмів розпізнавання він наразі не може забезпечити задовільний результат у деяких завданнях (швидке розпізнавання схожих об’єктів, рукописного тексту).

Розпізнавання образів - науковий напрям, пов'язаний з розробкою принципів і побудовою систем, спрямованих на визначення приналежності даного об'єкта до одного з раніше обраних класів об'єктів.

Завдання пошуку зображень на основі шаблонів є частиною (підзадачами) більш загального завдання розпізнавання образів. Шляхом несистематичного і ненаправленого пошуку "подібних" об’єктів з великої кількості об’єктів їх можна перераховувати нескінченно і не закінчувати із заданою ймовірністю. У деяких випадках об’єкти характеризуються ідентифікаційними параметрами (сигнатурами), такими як форма, колір, розташування, рухливість тощо, унікальними характеристиками та їх комбінаціями. Суб'єкти класифікуються відповідно до цих факторів. Часто не стоїть глобальне завдання класифікувати всі навколишні об'єкти, а необхідно виділити певний тип об'єкта у вхідному відеопотоці.

Завдання розпізнавання – це інформаційне завдання, яке складається з двох етапів:

- Перетворення необроблених даних у форму, яку легко ідентифікувати;
- Правильно визначено (вказує на приналежність об’єкта до класу).

У цих завданнях можна ввести поняття аналогії або подібності об’єктів і створювати правила на основі цих правил для об’єктів, що містяться в одному або різних класах. У цих завданнях ви можете оперувати набором прикладів випадків відомої класифікації, які можна представити у формальному описі за

допомогою алгоритму розпізнавання, щоб відповідати навчальному завданню.

Для вирішення цих завдань важко розробити формальні теорії та застосувати класичні математичні методи (часто немає інформації про точні математичні моделі, або переваги використання моделей і математичних методів не можна порівняти з витратами).

Розрізняють наступні типи завдань розпізнавання:

- Завдання на розпізнавання - віднести пред'явлений об'єкт до однієї із поданих категорій за його описом (розучувати з учителем);

- завдання автоматичної класифікації – класифікація набору об'єктів, ситуацій, явищ у непересічні системи класів на основі описів (класифікація, кластерний аналіз, самонавчання);

- завдання виділення набору інформативних ознак при розпізнаванні;

- завдання перетворення необроблених даних у легко впізнавану форму;

- Динамічне розпізнавання та динамічна класифікація

- Прогнозні завдання є попередніми типами, і рішення повинні прийматися з прив'язкою до деякого часу в майбутньому.

У ході біологічної еволюції багато тварин вирішували завдання розпізнавання образів за допомогою органів зору і слуху. Складною теоретико-технічною проблемою залишається створення систем штучного розпізнавання образів. Ця ідентифікація потрібна в різних сферах, від військових і систем безпеки до оцифровки різних аналогових сигналів.

Традиційно задачі розпізнавання образів відносяться до задач штучного інтелекту.

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 автоматизований метод розпізнавання зображень з використанням нейронної мережі.

Предмет дослідження – процес розробки застосунку для розпізнавання зображень з використанням нейронної мережі.

Метою даної кваліфікаційної роботи є розробка застосунку для розпізнавання літаків на зображеннях використовуючи нейронну мережу. Отримати навички розробки програмного забезпечення для розпізнавання зображень на основі нейронних мереж за допомогою IDE спільноти PyCharm. Цей програмний модуль буде реалізовувати такі функції:

1) Виконати виявлення зображень на завантажених графічних зображеннях;

2) Виявлення зображень у завантажених відеофайлах;

3) Визначити зображення на зображенні, отриманому з веб-камери;

4) Вивести результат у командний рядок;

5) Можливість переходу на інші версії нейронних мереж YOLO (YOLOv2, TinyYOLOv2, YOLOv3, TinyYOLOv3).

Наукова новизна полягає у розробці нової методики розпізнавання зображень із застосуванням можливостей бібліотеки OpenCV та мови програмування Python для створення застосунку для розпізнавання зображень використовуючи нейронну мережу.

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

РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Сучасні програмні системи 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), візуалізації 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 (рис. 1.1), яка може включати 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.

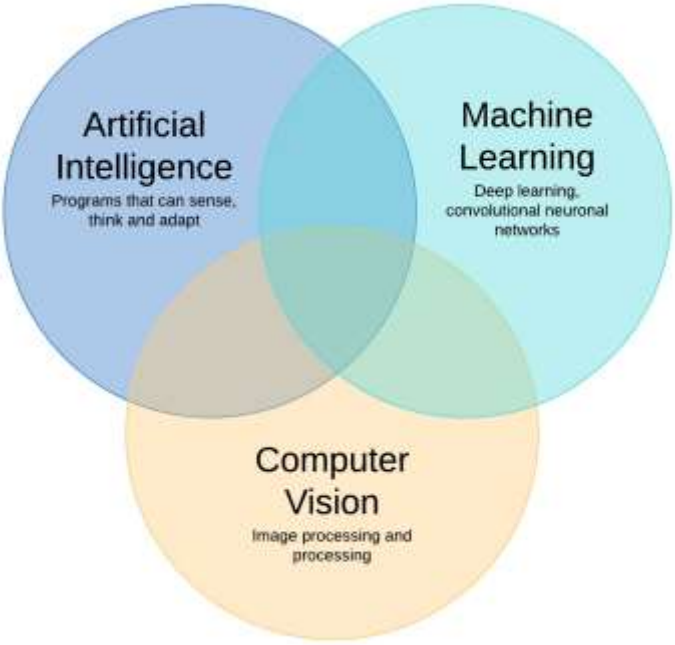


Рисунок 1.1 – Огляд зв’язку між штучним інтелектом, машинним навчанням і комп’ютерним зором

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). Цей підхід вимагає або великої кількості прикладів задач 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 самодостатніми та вирішують конкретні проблеми перевірки та вимірювання, тоді як інші 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 заздалегідь 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 2D-зображення, 3D-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 відбиття 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. приклад:

- Повторна дискретизація для забезпечення правильності системи координат зображення;

- шумозаглушення для усунення спотворень, внесених датчиком;

- збільшити контраст, щоб можна було виявити необхідну інформацію;

- Масштабування, щоб краще розрізняти структури 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 крапки: 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 точок, які нас цікавлять;

- Сегментувати одну або декілька областей зображення, що 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:

- перевірка відповідності даних умовам, які залежать від методу та застосування;

- Оцінка характерних параметрів, таких як розташування або розмір об'єктів;

- Класифікрція знайдених предметів за різними категоріями.

Відомі такі програмні продукти для розпізнавання зображень:

- CuneiForm - засіб оптичного розпізнавання символів, розроблений російською компанією Cognitive Technologies. Програма перетворює файли зображень, отримані зі сканера або іншим способом, у текст (рис. 1.2);

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

- Readiris - програма для перетворення документів у різні формати. Сканер або МФУ дозволяє сканувати та розпізнавати текст за допомогою цієї утиліти.

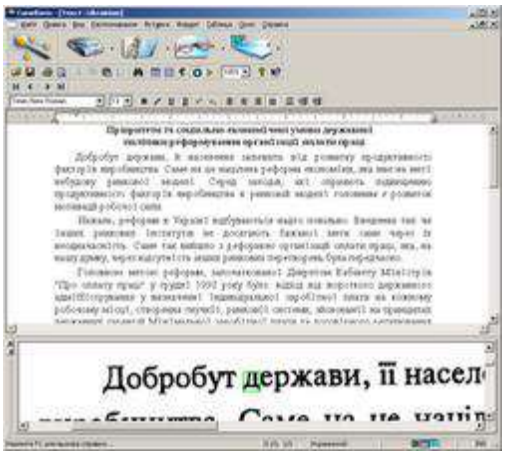


Рисунок 1.2 – Головне вікно CuneiForm

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 200x100 DPI. CuneiForm зберігає форматування тексту та розпізнає складні форми довільної структури.

FineReader (рис. 1.3) швидко й точно розпізнає відскановані або сфотографовані документи та перетворює їх у редаговані електронні формати або PDF-файли з можливістю пошуку.



Рисунок 1.3 – Головне вікно FineReader

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. Місце нейромереж в задачі розпізнаванні образів

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 рисунку 1.4 зображено принципову схему нейрона.

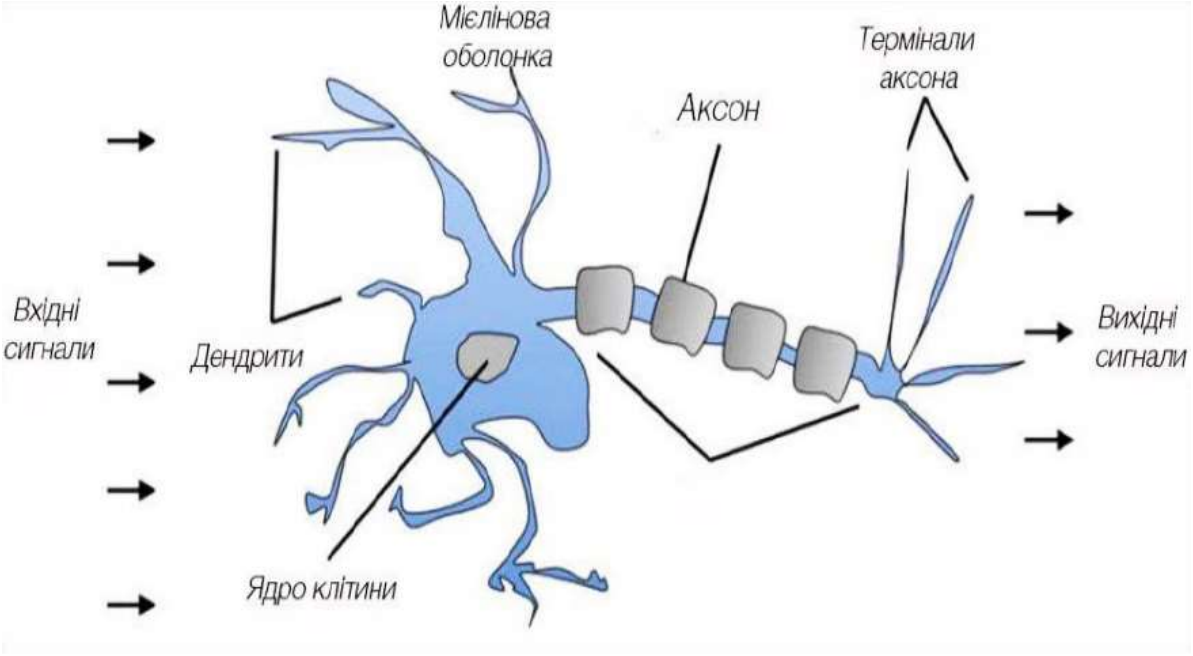


Рисунок 1.4 – Зображення біологічного нейрона

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 на відео;

- Виявляти звуки, визначати, хто говорить, транскрибувати розмову в текст і розпізнавати почуття в звуках;

- Класифікуйте текст як спам (в електронних листах) або шахрайство (у страхових претензіях).

Будь-яку мітку, яку може згенерувати людина, будь-який цікавий вам результат, 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.

- Відмова обладнання (ЦОД, виробництво, транспортування);

- Розлади здоров'я (інсульт, інфаркт на основі життєво важливих статистичних даних і даних, які можна носити);

- Замовлення клієнта (прогнозування ймовірності відходу клієнта на основі мережевої активності та метаданих);

- Старанність 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 має велике значення для речей, які ви хочете виявити та запобігти, наприклад шахрайства.

Глибоке навчання – це назва, яка використовується для композитних 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 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. Схематичне зображення одного нейровузла зображено на рисунку 1.5.

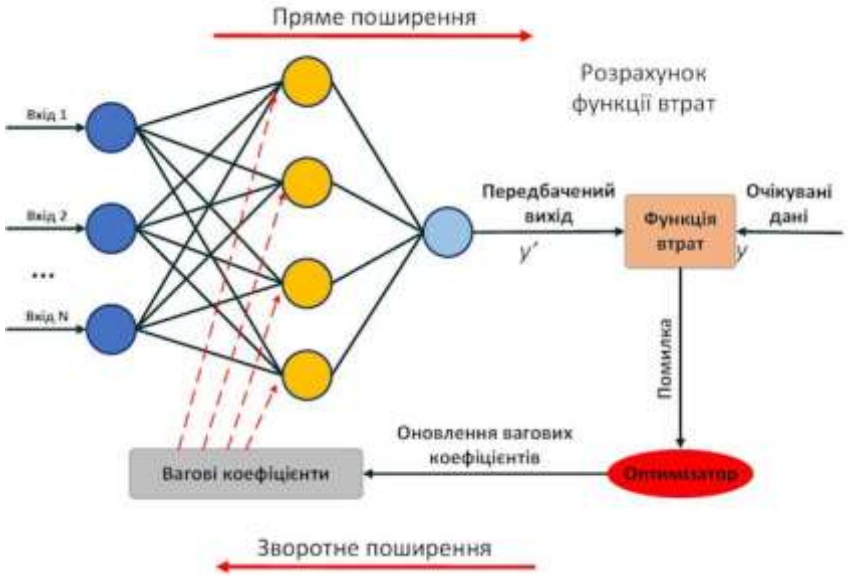


Рисунок 1.5 – Схематичне зображення одного нейровузла

Рівень вузла – це серія нейронних перемикачів, які вмикаються або вимикаються, коли вхідні дані надходять через мережу. Вихідні дані кожного шару також є вхідними даними для наступного рівня, починаючи з оригінального вхідного рівня, який отримав дані. На рисунку 1.6 показано простий приклад нейронної мережі.

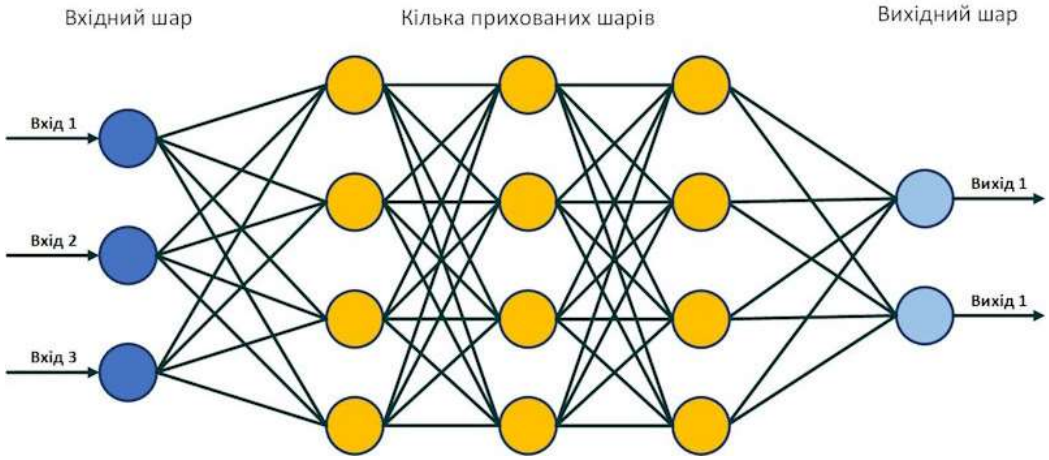


Рисунок 1.6 – Приклад типової нейронної мережі

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.

Мережі глибокого навчання відрізняються від більш поширених одношарових нейронних мереж глибиною, 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. Більше трьох рівнів (включно з входом і виходом) називається глибоким навчанням, що означає, що глибина – це термін, який означає більше одного прихованого шару.

Глибоке навчання (також відоме як структуроване глибоке 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 наступному рівню. 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 складається 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 рисунку 1.7 показано дедалі складнішу та абстрактнішу ієрархію функцій.

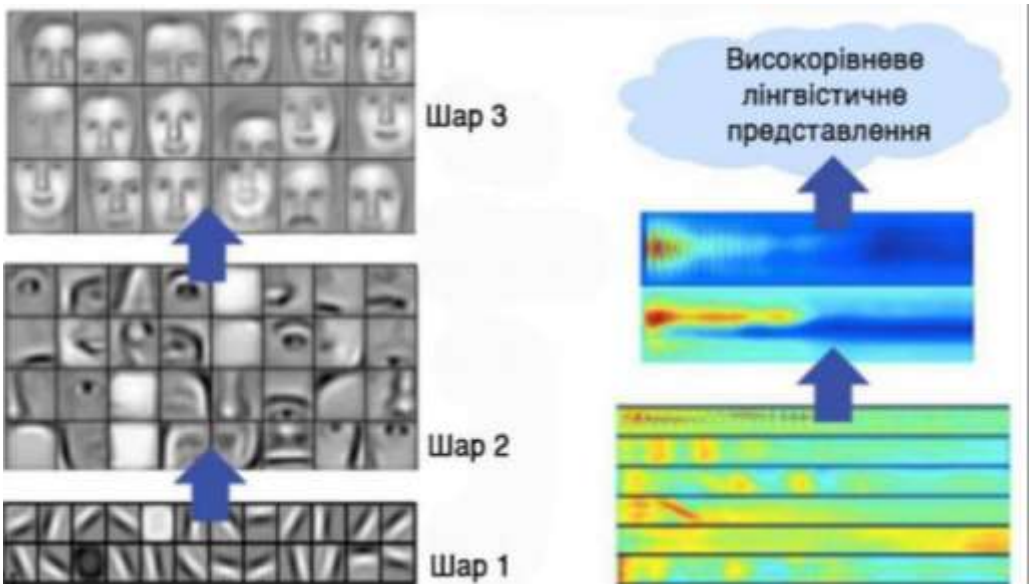


Рисунок 1.7 – Ієрархія особливостей представлення образів нейромережею

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, наповнені гнівними скаргами, можуть групуватися в одному кутку векторного простору, тоді як задоволені клієнти або повідомлення про спам-сайти можуть групуватися в інших кутах. Це основа для різноманітних фільтрів повідомлень, які можна використовувати для управління взаємовідносинами з клієнтами. Те саме стосується голосових повідомлень.

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 вивчає функції автоматично, 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 ці нейронні мережі навчаються виявляти кореляції між певними релевантними функціями та найкращими результатами – вони встановлюють зв’язки між сигналами ознак і тим, що ці характеристики представляють, незалежно від того, 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. Класичні методи для 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 про розпізнані об'єкти відіграє важливу роль у процесі вирішення проблеми, яка необхідна для формування алфавіту класу розпізнавання, словника ознак і матриці навчання. 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].

Алфавіт для заданого класу розпізнавання:

$$\{X_{mo} | m = 1. .. M\},$$

На етапі навчання необхідно:

1) побудувати кілька оптимальних (тут і далі інформаційного змісту) методів розбиття R простору ознак Ω|N| щодо ідентифікаційних класів;

2) Геометричні параметри R* оптимально розподіляються на класи розпізнавання на основі побудованого простору ознак, і встановлюється безпомилкове правило прийняття рішень на основі 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 ідентифікаційних ознак, включаючи приведення значень до форми, яку легко обробляти на комп’ютері, формування реалізованого зображення вектора 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 декілька прикладів формування вирішальних правил. Нехай X_1 і X_2 – два класи розпізнавання, розподіл реалізацій яких показано на рис. 1.8.

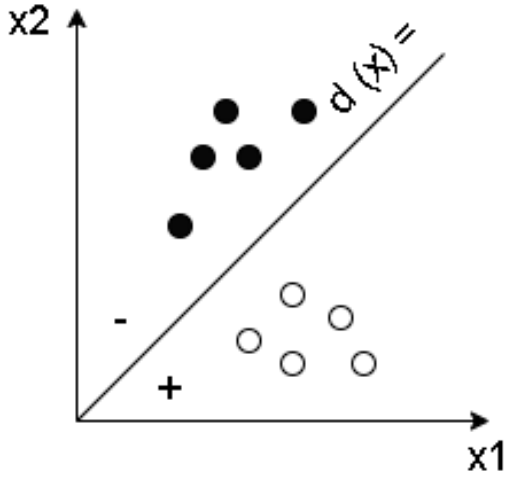


Рисунок 1.8 – Розподіл реалізацій двох класів при розпізнаванні

Кожна реалізація є вектором ознак розпізнавання x = (x1, x2)^T, заданим у просторі ознак x1 - x2. На рисунку 1.7 реалізації, що належать до класу *X*1, позначені світлими кружками, а реалізації, що належать до класу *X*2, позначені чорними кружками.

Реалізуйте, як показано. 1.7 Клас розпізнавання можна розділити прямою лінією: d(x) = w0 + w1x1 + w2x2 = 0.

При цьому значення коефіцієнтів w0, w1 і w2 визначаються так, що нерівність d(x) > 0 виконується для всіх реалізацій класу X1, а d(x) < 0 для всіх реалізацій класу X2. Крім того, для будь-якої реалізації x, якщо апріорі відомо, що вона належить до одного з класів X1 або X2, ви можете обчислити значення d(x) і визначити, що x належить X1, якщо d(x) > 0, або x належить X2, якщо d (x) < 0. Сепарабельна функція d(x), сформована таким чином, називається лінійним вирішальним правилом класу X1.

Для N-вимірного випадку x = (x1, x2, ... , xN)T пряма стає гіперплощиною:

d(x) = w0 + w1x1 + w2x2 + … + wNxN = w0 + w Tx = 0, де w = (w1, w2, ... , wN) – вектор коефіцієнтів, значення якого є вимогою.

Зазвичай векторна реалізація x розширюється до x = (1, x1, x2, ... , xN)T, а вектор коефіцієнтів - до w = (w0, w1, w2,..., wN). Тоді лінійне вирішальне правило набуває такого вигляду d(x) = w Tx = 0.

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 шкалами кількісного вимірювання [7].

Найпоширенішою мірою близькості в теорії розпізнавання образів є евклідова відстань:

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

де xm – реалізація класу xm; x_l – реалізація класу xl; $x_m{}^i$ – і-та ознака розпізнавання класу Xmo; $x_l{}^i$ – і-та ознака розпізнавання класу $X_l{}^i$; N – кількість ознак розпізнавання 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 визначених класів, які знаходяться далеко один від одного, використовуйте квадрат евклідової відстані:

$$d_e^2(x_m, x_l) = \sum_{i=1}^{N} \left(x_m^i - x_l^i\right)^2$$

Ця міра дозволяє збільшити внески віддалених реалізацій, оскільки вони зведені в квадрат. Щоб зменшити вплив великих відмінностей, використовуйте лінійну міру наближення:

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

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

Використовуйте узагальнену відстань потужності в ситуаціях, коли необхідно зменшити або збільшити ваги реалізації зображень із великими відмінностями:

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

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.

1.4. Постановка задачі

Відповідно до поставлених цілей в роботі потрібно вирішити наступні завдання:

- Аналіз сучасних програмних комплексів для розпізнавання образів;
- Проаналізувати типи нейронних мереж, доступних для програмних модулів розпізнавання зображень;
- Проаналізувати архітектуру згорткової нейронної мережі та вибрати програмний модуль для розпізнавання зображень;
- Розробка програмних модулів розпізнавання зображень на основі нейронної мережі;
- Перевірка роботи програмних модулів.

РОЗДІЛ 2 ОГЛЯД НЕЙРОМЕРЕЖ ТА 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. 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).

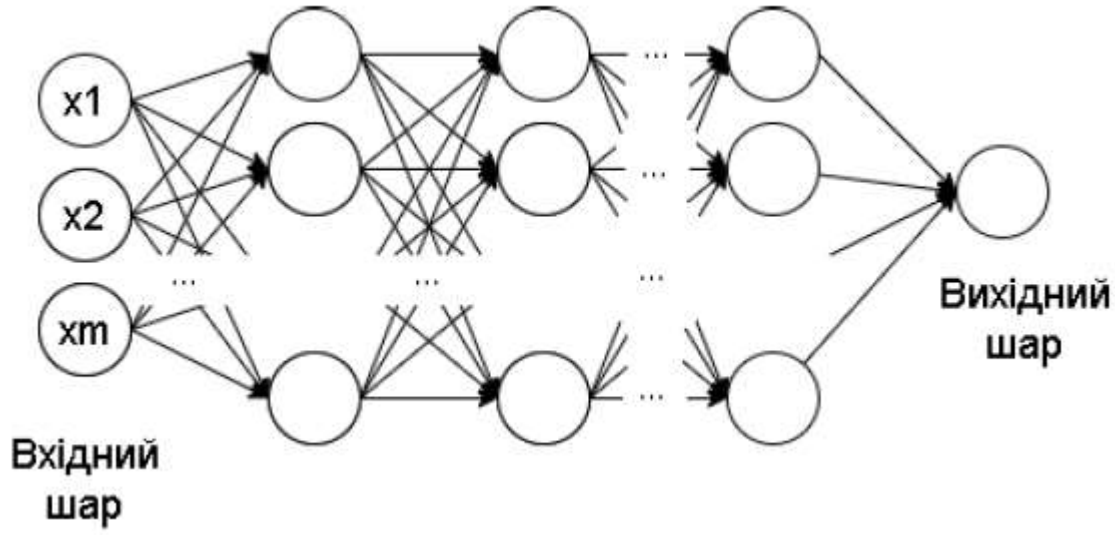


Рисунок 2.1 – Багатошаровий перцептрон

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 рисунку 2.2 показано, як виглядає згорткова нейронна мережа.

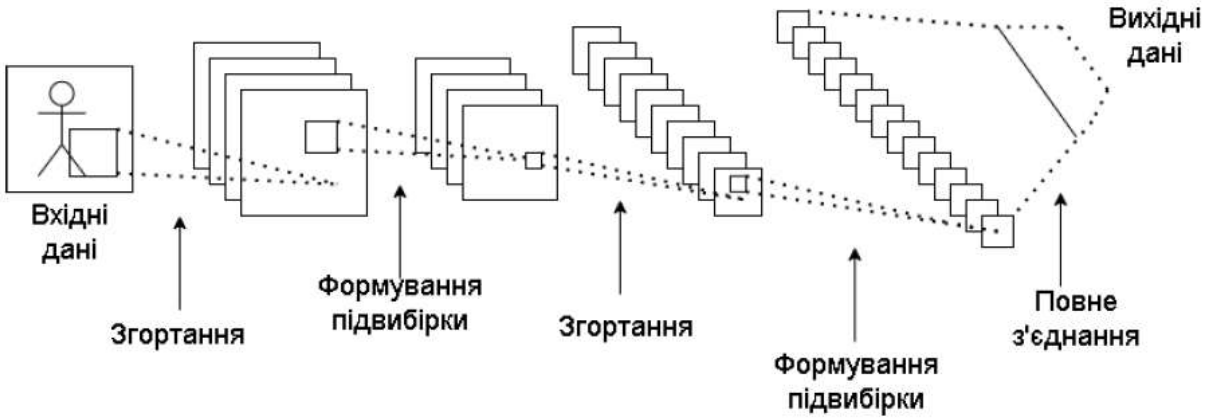


Рисунок 2.2 – Конволюційна (згорткова) нейромережа

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 рисунку 2.3 зображено принципову схему рекурентної нейронної мережі.

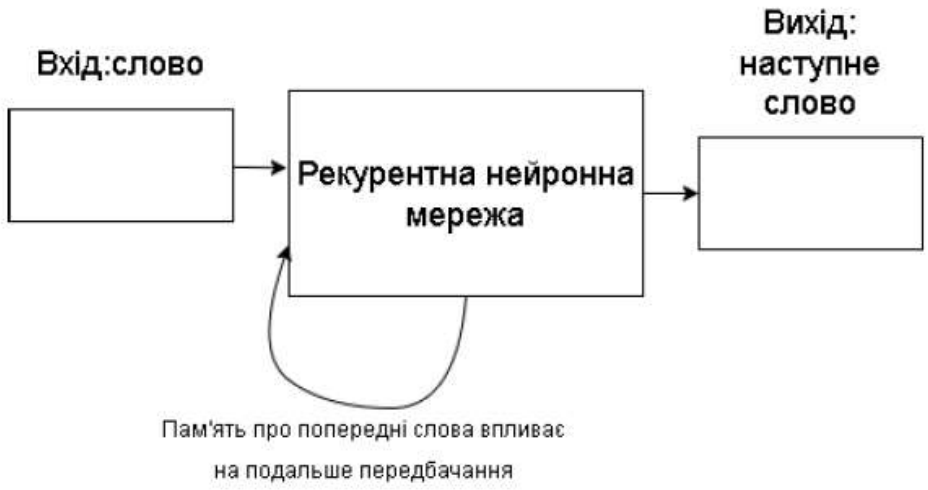


Рисунок 2.3 – Рекурентна нейромережа

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.

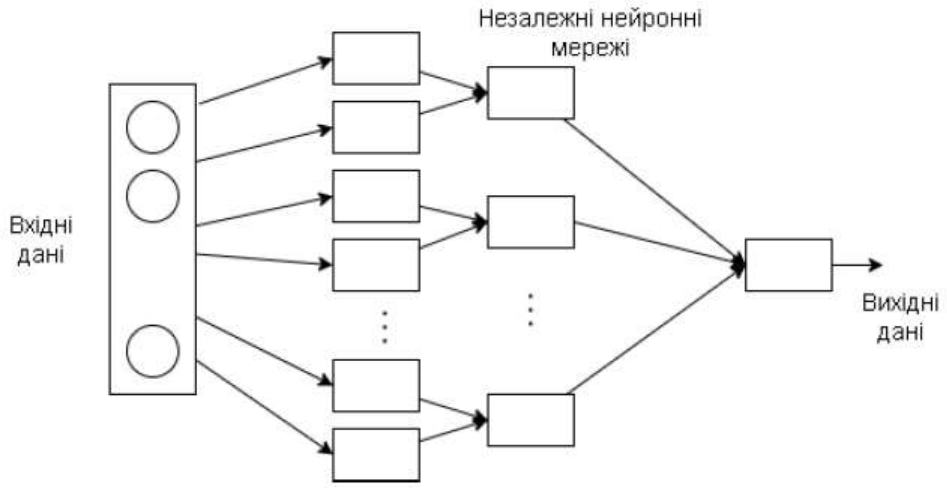


Рисунок 2.4 – Схема модульної нейромережі

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 рисунку 2.4 показано візуальне представлення модульної нейронної мережі.

2.2. Навчання 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 як найточніше відповідають цим цільовим значенням. На рисунку 2.5 показана проста модель частини мережі.

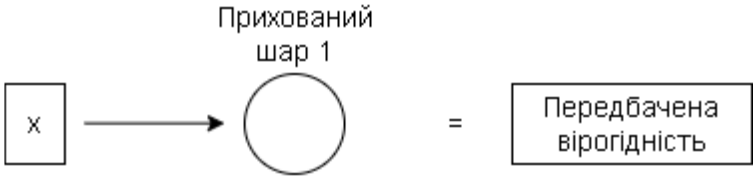


Рисунок 2.5 – Реалізована через одну умову логістична операція

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 рисунку 2.6 показано рівняння для цієї операції.

Sigmoid(B1*X+B0) = передбачена вірогідність

Рисунок 2.6 – Рівняння логістичної регресії

Розглянемо кожен елемент:

- X – це вхідні дані, єдина функція, яку ми надаємо моделі для обчислення прогнозів;

- B1 – приблизний параметр нахилу логістичної регресії;

- B0 – це зміщення, дуже схоже на член перехоплення в регресії. Ключова відмінність полягає в тому, що в нейронній мережі кожен нейрон має власний термін зміщення.

- Блакитні нейрони також включають функцію активації сигмоподібної кишки (позначену кривою всередині синього кола). Сигмоїдна функція використовується для переходу від логарифма до ймовірності;

- Нарешті, ми отримуємо прогнозовану ймовірність, застосовуючи сигмоїдну функцію до значення (B1 * X + B0).

Отже, 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, пов’язане з певним нейроном), вага якого перетворює ваш вхід (за допомогою B1) і передає його нейрону;

- Нейрон, що містить зсувний член (B0) 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. Ця більш складна мережа зв’язків дозволяє нейронній мережі створювати передбачувані

відповіді. на рис. На рисунку 2.7 зображено принципову схему складної нейронної мережі.

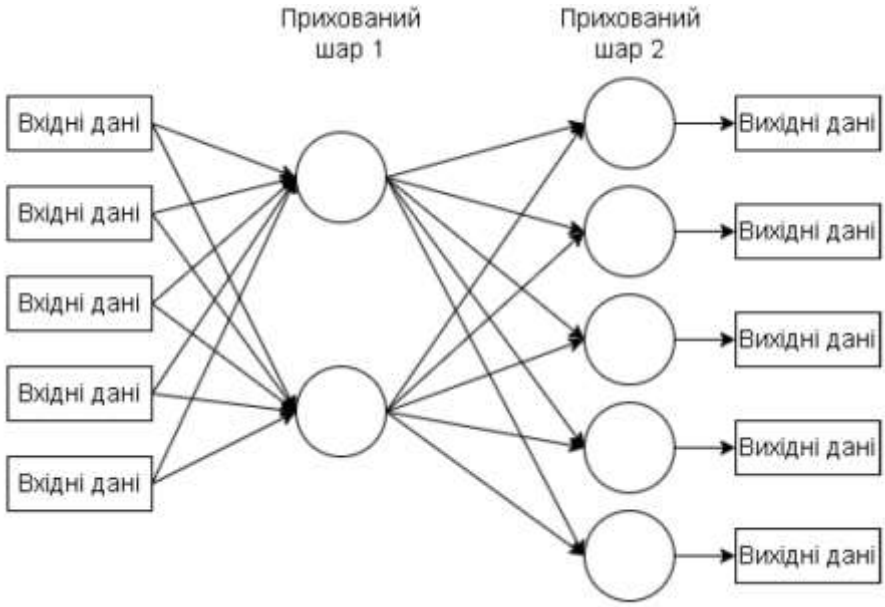


Рисунок 2.7 – Схема складної нейромережі

Перший прихований шар складається з двох нейронів. Отже, щоб підключити всі п’ять входів до нейронів у прихованому шарі 1, нам потрібно десять з’єднань. На рисунку 2.8 показано лише зв’язок між входом 1 і прихованим шаром 1.

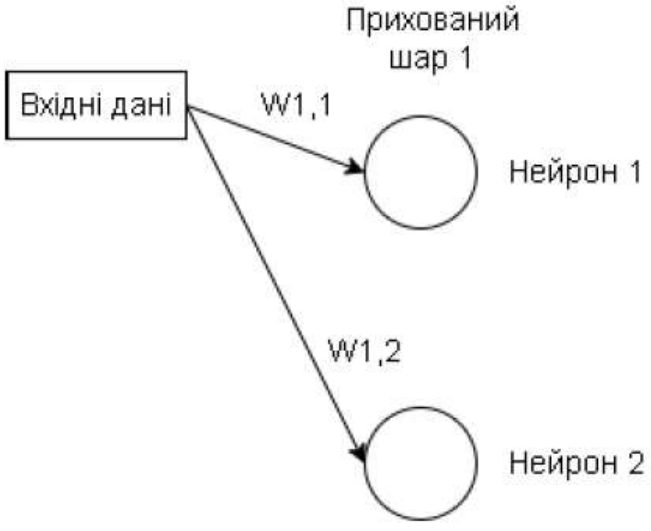


Рисунок 2.8 – Зв'язок між входом 1 і прихованим шаром

Представлення вагових коефіцієнтів у зв’язках – W1,1 представляє ваги у зв’язку між входом 1 і нейроном 1, а W1,2 представляє вагомість у зв’язку між

входом 1 і нейроном 2. Загальне позначення W a , b позначає вагові коефіцієнти, що з'єднують вхід a (або нейрон a) і нейрон 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:

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

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

[W] * [X] + [bias] = [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], що дає нам нейронні виходи (активації) поточного рівня. На рисунку 2.10 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 (B0, B1, B2 тощо), які мінімізують функцію витрат. Для нейронних мереж ми робимо те ж саме, але в більшому масштабі та складніше.

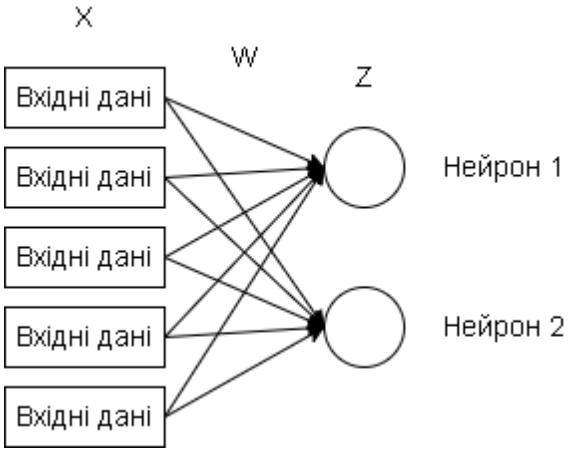


Рисунок 2.10 – Графічна інтерпретація [W], [X] і [Z]

У традиційній регресії ми можемо змінити будь-яке конкретне значення параметра, не впливаючи на інші бета-коефіцієнти. Таким чином, застосовуючи невеликі окремі зміни до кожної бета-версії та вимірюючи їхній вплив на функцію витрат, ми можемо відносно просто визначити, у якому напрямку нам потрібно рухатися, щоб зменшити та зрештою мінімізувати витрати.

У нейронній мережі зміни вагових коефіцієнтів 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 – це, по суті, група 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).

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

$$SCF = СУМА\ [(прогноз - фактичний)^2] * (1 / кількість_спостережень)$$

SCP – 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, нам потрібно знати градієнт функції вартості, яка є вектором, спрямованим у найкрутішому напрямку (щоб отримати результат, нам 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:

$$\text{Градієнт } C\ (B0, B1) = [[dC / dB0], [dC / dB1]]$$

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

- Розрахувати градієнт нашого поточного положення;

- Змінюйте кожен параметр на величину, пропорційну його градієнтному елементу та в протилежному напрямку його градієнтного елемента. Наприклад, якщо наша функція авитрат 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.



Рисунок 2.11 – Графічна інтерпретація 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 рисунку 2.12 показана проста нейронна мережа, яка поширюється вперед від входу до виходу. Процес можна підсумувати в наступних кроках: - Вхідні дані надходять до синього шару нейронів і змінюються ваговими коефіцієнтами, зміщеннями та сигмою в кожному нейроні, щоб отримати активацію. Наприклад: activation_1 = sigmoid (offset_1 + W1 * input_1); - Активація 1 і активація 2 із синього шару надходять до пурпурового нейрона, який використовує їх для отримання остаточної активації на виході. Метою зворотного поширення є обчислення активації кожного нейрона кожного наступного прихованого шару, поки ми не досягнемо результату.

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

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

- Вхідні дані надходять до синього шару нейронів і змінюються ваговими коефіцієнтами, зміщеннями та сигмою в кожному нейроні, щоб отримати активацію. Наприклад: activation_1 = sigmoid (offset_1 + W1 * input_1);

- Активація 1 і активація 2 із синього шару надходять до пурпурового нейрона, який використовує їх для отримання остаточної активації на виході.

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

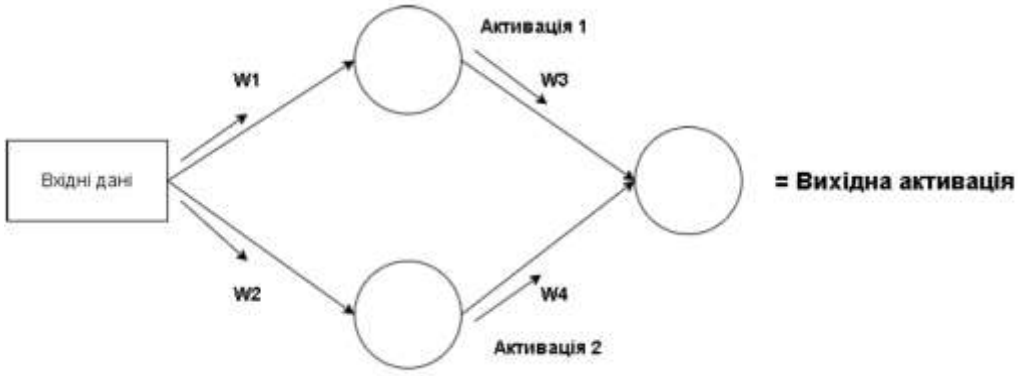


Рисунок 2.12 – Поширення градієнту нейромережею

На рисунку 2.13 показано вихідну активацію, яка використовується для прогнозування та як кінцеве джерело помилки в моделі. Потім ця помилка повертається до моделі з використанням тих самих ваг і зв’язків, що використовуються для передачі сигналу (тому ми тепер маємо помилку 1 замість активації 1 – помилку, приписану 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.

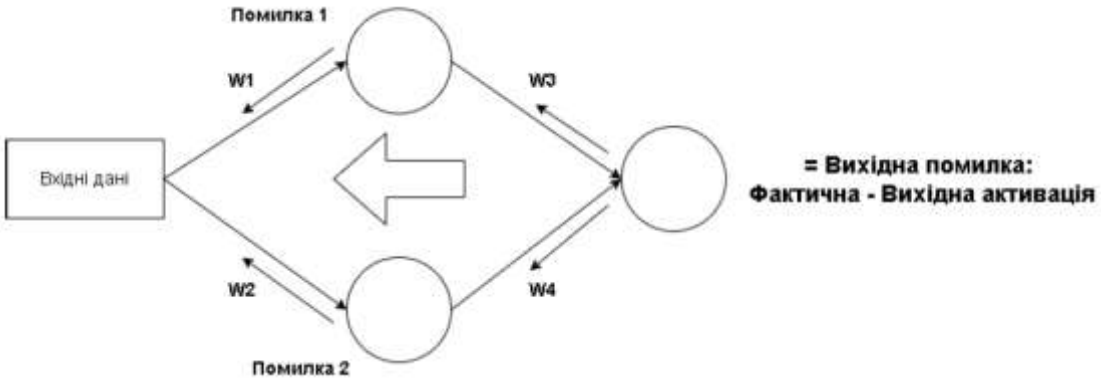


Рисунок 2.13 – Зворотнє поширення градієнту нейромережею

Двома компонентами 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 відносно входу 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 спуск.

2.4. Обґрунтування архітектури системи 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;

- Mask R-CNN – це розширена Fast R-CNN;

- YOLO. Ілюстрація роботи повністю відрізняється від попереднього, а площа взагалі не використовується, що є найшвидшим;

- SSD. За принципом схожий на YOLO, але використовує VGG16 як мережу вилучення функцій. Він також досить швидкий і підходить для роботи в реальному часі;

- Feature Pyramid Network (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.

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 функцій CNN на основі лінійної регресії. 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.

Отже, процес виявлення цілі мережею R-CNN можна розділити на наступні етапи:

- Використовуйте вибірковий пошук для вибору регіонів-кандидатів;

- Перетворити регіон до розміру, прийнятного CNN CaffeNet;

- Використовуйте CNN для отримання 4096-вимірних векторів ознак;- Використовуйте N лінійних SVM для виконання N бінарних класифікацій для кожного вектора ознак;

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

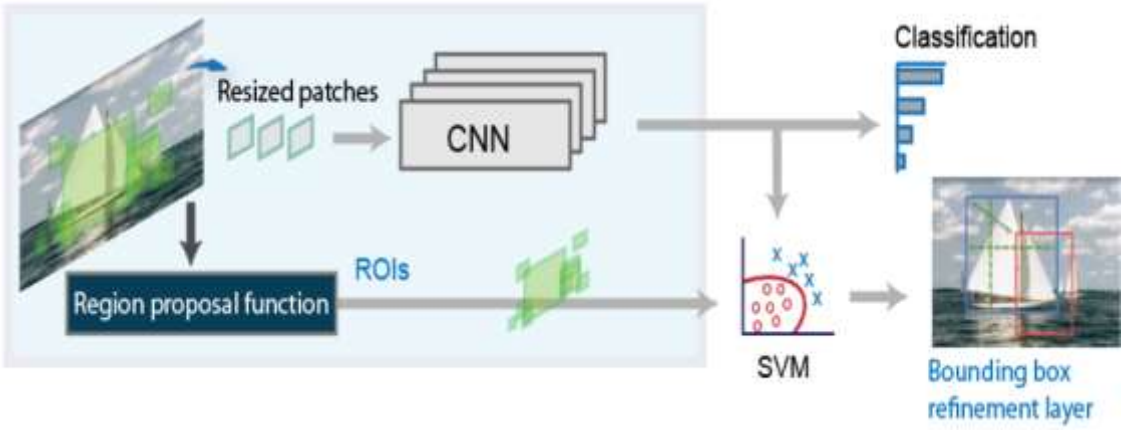


Рисунок 2.14 – Ілюстрація роботи архітектури R-CNN

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. Швидкий 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, VGG16). 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:

- Передача CNN проходить не через кожен регіон-кандидат окремо, а через весь образ в цілому. Потім запропоновані регіони накладаються на згенеровану загальну карту характеристик;

- Замість незалежного навчання трьох моделей (CNN, SVM, регресора bbox), усі процедури навчання об’єднані в одну.

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 × W розміром h / H × w / W. Виконайте Max Pooling для кожної такої комірки, щоб вибрати лише одне значення, що дасть результуючу матрицю ознак H × W.

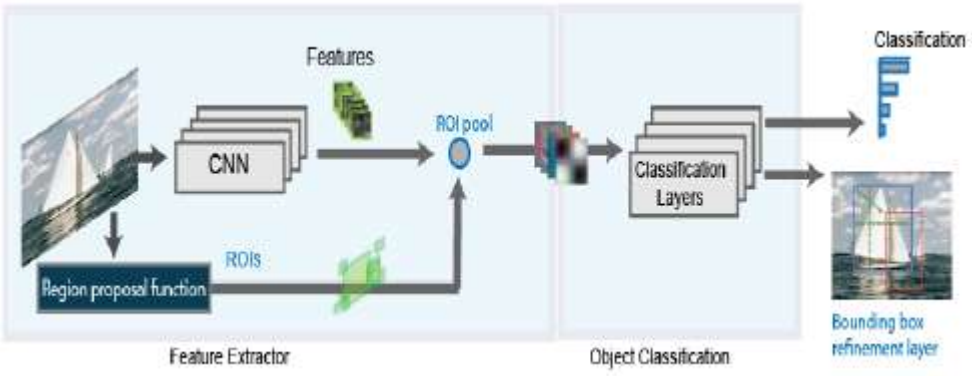


Рисунок 2.15 – Ілюстрація роботи архітектури Fast R-CNN

Швидший 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.

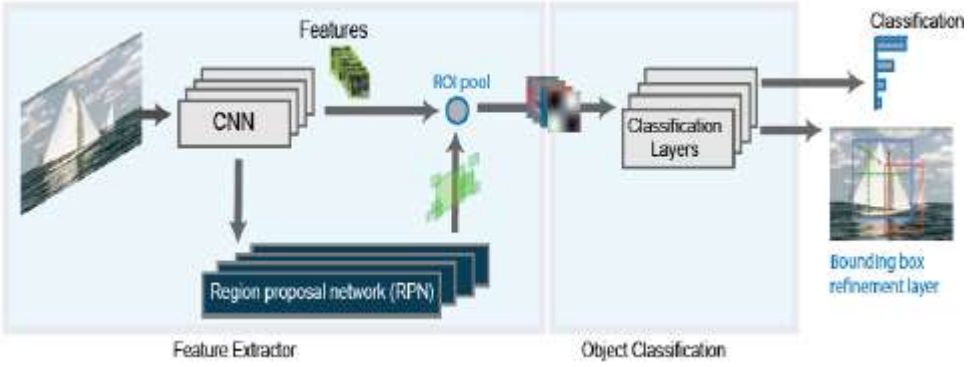


Рисунок 2.17 – Ілюстрація роботи архітектури Faster R-CNN

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.

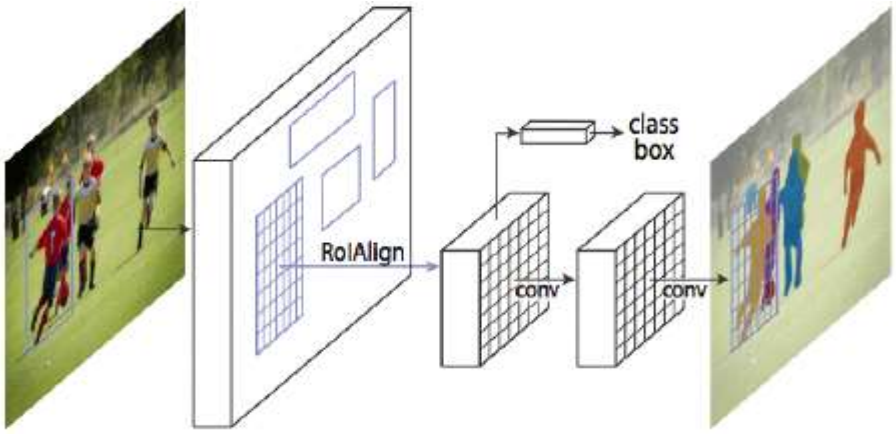


Рисунок 2.18 – Ілюстрація роботи архітектури Faster R-CNN

Автори статті зазвичай поділяють розроблену архітектуру на мережу CNN для обчислення характеристик зображення, яку вони називають хребтом, і голову – комбінацію частин, відповідальних за прогнозування кадрів, класифікацію об’єктів і визначення їхніх 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.



Рисунок 2.19 – Порівняння вихідного зображення і карти ознак

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. Передбачається лише один об’єкт на клітинку сітки. Наприклад, сітка жовтих клітинок нижче намагається передбачити об’єкт "собака", центр якого знаходиться в клітинці сітки.

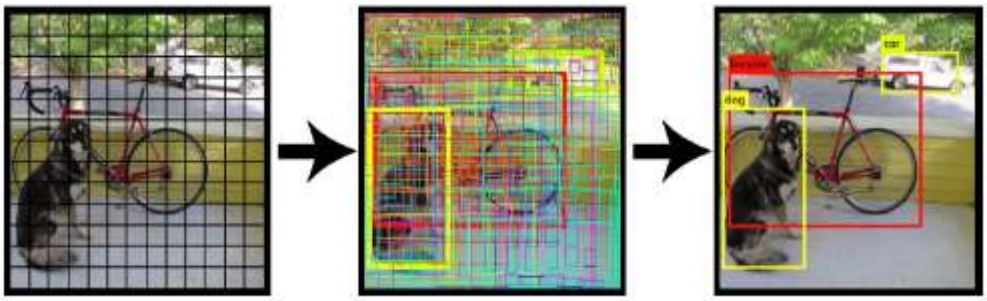


Рисунок 2.20 – Приклад накладання комірної сітки архітектури YOLO

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 граничних полів, кожне з рівнем довіри;
- Виявляти лише один об’єкт незалежно від кількості вікон B;

Передбачте умовні ймовірності класу C (одна ймовірність класу об’єкта на клас).

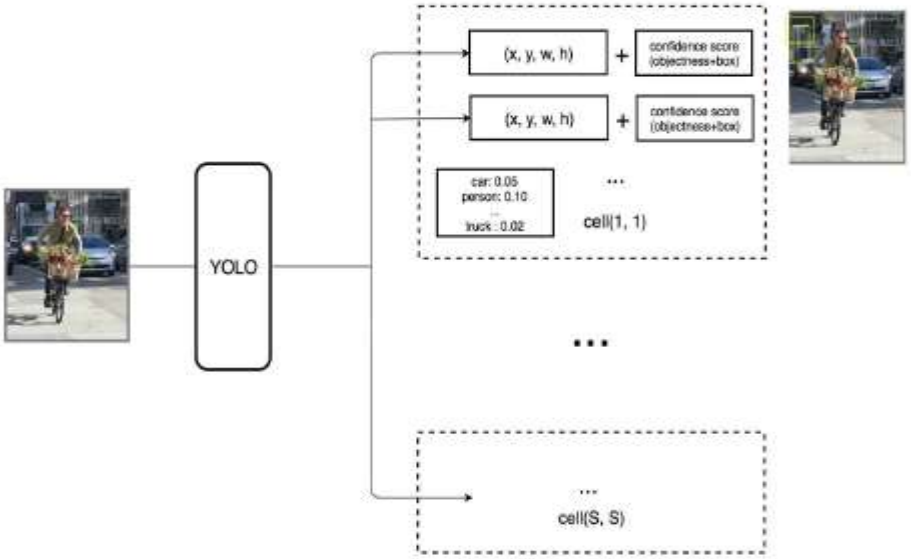


Рисунок 2.21 – Ілюстрація роботи архітектури YOLO

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. Отже, 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). Отже, прогноз YOLO має форму (S, S, B × 5 + C) = (7, 7, 2 × 5 + 20) = (7, 7, 30).

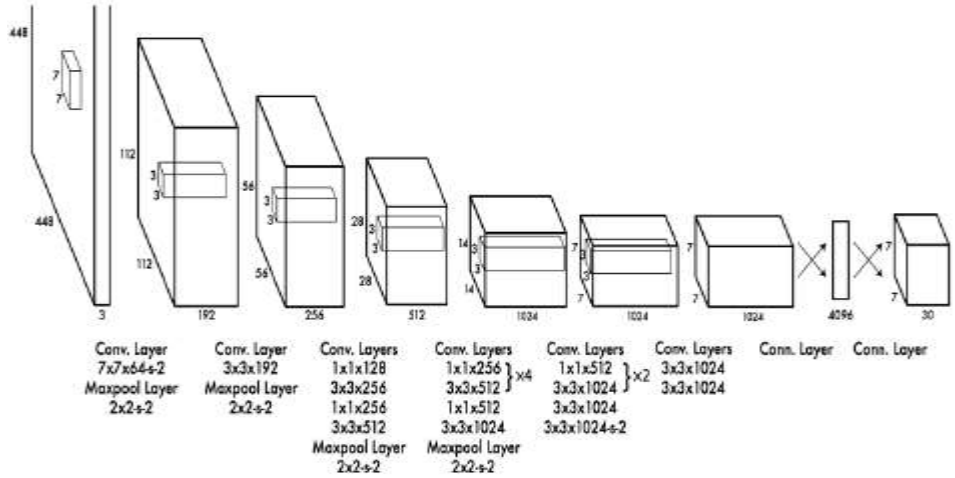


Рисунок 2.22 – Схема-проект нейромережі YOLO

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 з’єднаних шарів для прогнозування граничного поля 7×7×2 (нижнє середнє зображення). 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 тензори сплющуються. 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 веде до спеціалізації у передбаченні обмежувальної структури. 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;

- втрати локалізації (похибка між оціненим граничним полем та істинністю землі);

- Втрата впевненості (об’єктивність коробки).

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 VGG16 для отримання карт функцій. Потім він виявляє об’єкти за допомогою рівня Conv4_3. Для ілюстрації ми просторово намалюємо Conv4_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. Conv4_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].

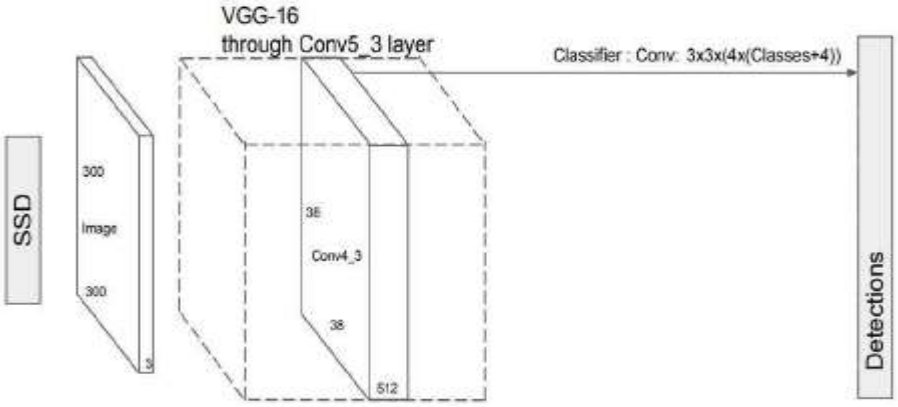


Рисунок 2.23 – Ілюстрація роботи архітектури SSD

В експериментах Mask R-CNN, поряд із звичайним ResNet-50/101 CNN як магістраль, також було проведено техніко-економічне обґрунтування з використанням мережі Feature Pyramid Network (FPN). Вони показують, що застосування FPN в магістральній мережі покращує як точність, так і продуктивність Mask R-CNN. Це робить доцільним також описати це вдосконалення, хоча йому присвячено окремий документ і він мало пов’язаний із серією статей, що розглядаються. 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, отримані шляхом послідовного зменшення шарів CNN, розглядаються як свого роду ієрархічна "піраміда", яка називається шляхом "знизу вгору". При цьому карти ознак нижнього та верхнього шару піраміди мають свої переваги та недоліки: перша має високу роздільну здатність, але низька семантика та здатність до узагальнення, а друга – навпаки:

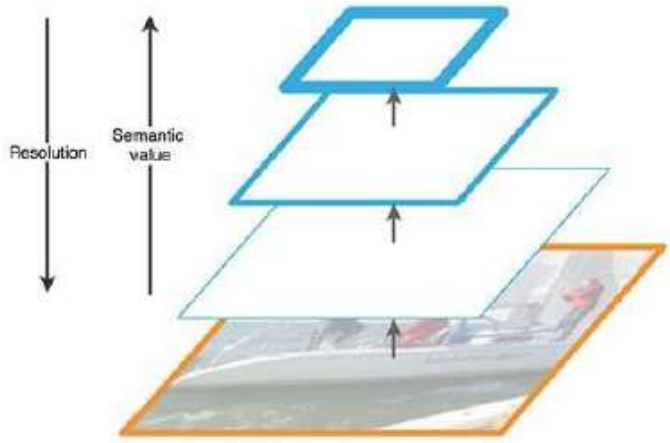


Рисунок 2.24 – Ілюстрація роботи архітектури FPN

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.

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).

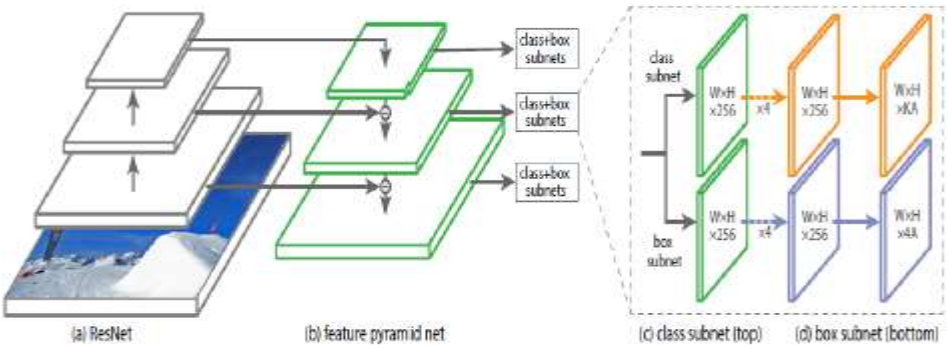


Рисунок 2.25 – Ілюстрація роботи архітектури RetinaNet

РОЗДІЛ 3 РОЗРОБКА МОДУЛЯ ДЛЯ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

3.1. Вибір середовища для розробки

Інтегроване середовище розробки (IDE) - 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 [15].

Інтегровані середовища розробки зазвичай включають редактори вихідного коду, компілятори або інтерпретатори, інструменти автоматизації збірки 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, автоформатування та автоматичний відступ;

- Проста та потужна навігація в коді;

- Допомога з написанням коду, включаючи автозаповнення, автоімпорт, шаблони коду, перевірку сумісності версії мовного інтерпретатора тощо;

- Швидкий перегляд документації для будь-якого елемента безпосередньо у вікні редактора, перегляд надісланої документації через браузер, підтримка рядків документів – 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 Jinja2;

- Редактор Javascript, Coffeescript, HTML/CSS, SASS, LESS, HAML;

- Інтеграція з Version Control System (VCS);

- Діаграми класів UML, діаграми моделей Django та Google App Engine;

- Інтеграційне модульне тестування;

- Інтерактивна консоль для Python, Django, SSH, відладчик і база даних;

- Повнофункціональний графічний відладчик (Debugger);

- Підтримка найпопулярніших рішень IDE, таких як Netbeans, Eclipse, Emacs, емуляція редактора VIM;

- Мови: Python (версії: 2.x, 3.x), Jython, Cython, IronPython, PyPy, Javascript, CoffeScript, HTML/CSS, Django/Jinja2 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].

3.2. Перелік використаних 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:

- Лаконічний синтаксис (блоки мають бути виділені з відступом);

- Портативність програми (характеристика більшості інтерпретованих мов);

- Стандартний дистрибутив має низку корисних модулів (включаючи один для розробки графічних інтерфейсів);

- Можливість використовувати Python у режимі розмови (дуже 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 розв’язувати математичні задачі (є інструменти для роботи з комплексними числами, може обробляти цілі числа будь-якого розміру, може використовуватися як потужний калькулятор у режимі розмови);

- Відкритий код (можливість редагування іншими користувачами).

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 написання сценаріїв і швидкої розробки додатків у багатьох галузях промисловості на більшості платформ.

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.

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 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 в той же час вона дуже специфічна порівняно з іншими об’єктно-орієнтованими мовами.

Особливості та особливості:

- Клас також є об’єктом з усіма наведеними нижче можливостями;

- Спадкування, в тому числі множинне;

- Поліморфізм (усі функції віртуальні);

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

- спеціальні методи керування життєвим циклом об’єкта: конструктор, деструктор, розподільник пам’яті;

- перевантаження операторів (крім is, '.', '=' і символьної логіки);

- властивості (живе моделювання з функціями);

- Керувати доступом до полів (макет полів і методів, частковий доступ тощо);

- методи керування найпоширенішими операціями (truth, len(), deepcopy, serialization, object iteration...);

- Метапрограмування (керування створенням класів, тригери створення класів тощо);

- повна інтроспекція;

- Класи та статичні методи, поля класів;

- Класи, вкладені в функції та інші класи.

Python підтримує парадигми функціонального програмування, зокрема:

- Функція – це об’єкт;

- функції вищого порядку;

- рекурсія;

- Розвинена обробка списків (спискові вирази, операції послідовності, ітератори);

- аналоги закриття;

- Часткове застосування цієї функції;

- Можливість реалізації інших способів у самій мові (наприклад, каррінг).

Програмне забезпечення (додатки або бібліотеки) на Python розроблено у вигляді модулів, 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 інтроспекцію під час виконання. Це означає, що для будь-якого об’єкта можна отримати всю інформацію про його внутрішню структуру.

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.

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 [21].

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 поведінки людей у відео, відстеження руху камери, відстеження рухомих об'єктів, вилучення 3D-моделей об'єктів, отримання 3D-хмар точок із стереокамер, з'єднання зображень для отримання зображень із високою роздільною здатністю. зображення цілих сцен, знаходити схожі зображення в базах даних зображень, усувати ефект червоних очей із зображень Flash, відстежувати рухи очей, розпізнавати пейзажі та встановлювати маркери для накладання доповненої реальності 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. Розгорнутий OpenCV використовує різноманітні способи: від зшивання зображень Street View, виявлення вторгнень на відео з ізраїльських камер спостереження, допомоги роботам у навігації та підбиранні об'єктів у Willow Garage, виявлення потопельників у європейських басейнах, швидкого розпізнавання обличчя.

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 і 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.

3.3. Розробка коду додатку

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 з архітектурою YOLOv7, 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.

Після цього відбувається навчання нейронної мережі з алгоритмом YOLOv7 використовуючи дані з датасету COCO.

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 файлу зображення об’єкта;

- команда video-path відповідає за ідентифікацію шляху до відеофайлу об'єкта;

- Порядок достовірності відповідає за фільтрацію мінімальної ймовірності слабких виявлень. Елементи з імовірністю нижче заданої не будуть виділені;

- Порогові команди відповідають за немаксимально 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/yolov3.cfg','./yolo_model/yolov3.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)

Відеофайли обробляються кадр за кадром подібним чином, а результат перетворюється та зберігається назад у відеофайл із розширенням avi.

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()
```

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

```
count = 0
vid = cv.VideoCapture(0)
while True:
_, frame = vid.read()
height, width = frame.shape[:2]
if count == 0:
```

frame, boxes, confidences, classids, idxs = infer_image(net, layer_names, height, width, frame, colors, labels, FLAGS)

```
count += 1
else:
```

frame, boxes, confidences, classids, idxs = infer_image(net, layer_names, height, width, frame, colors, labels, FLAGS, boxes, confidences, classids, idxs, infer=False)

```
count = (count + 1) % 6
cv.namedWindow('webcam', cv.WINDOW_NORMAL)
cv.imshow('webcam', frame)
if cv.waitKey(1) & 0xFF == ord('q'):
break
vid.release()
```

cv.destroyAllWindows()

Для завантаження та обробки зображень за допомогою розробленої системи достатньо ввести в термінал або командний рядок операційної системи таку команду: "*python yolo.py*

-i=input_files/image1.jpg", де останнім параметром є каталог із зображеннями.

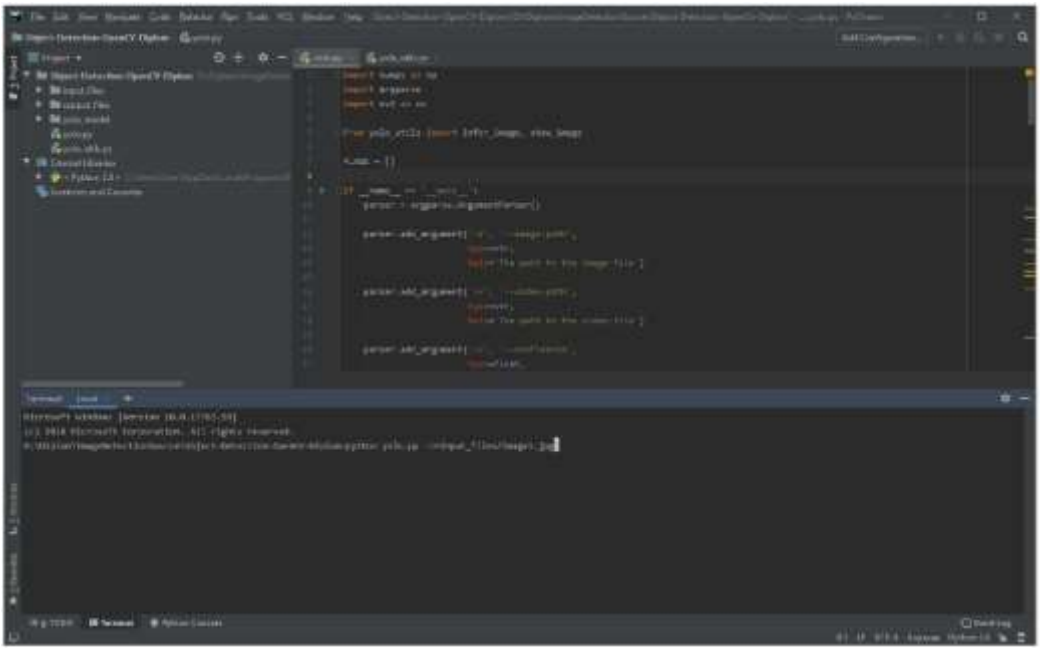


Рисунок 3.1 – PyCharm термінал

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!
```

Рисунок 3.2 – PyCharm термінал

Оброблене зображення також відкриється і виглядатиме так.



Рисунок 3.3 – Зображення оброблені за допомогою розробленої програми

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

Таблиця 3.1 Статистики роботи програми

Кількість оброблених зображень	Середня кількість знайдених об’єктів	Середня кількість всіх об’єктів на зображенні, які можливо детектувати	Середній час на обробку 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, має бути таким самим, як об’єкт для навчання нейронної мережі;

- Об’єкти, що ідентифікуються, повинні бути відокремлені один від одного;

- Обробка зображення повинна підкреслювати об’єкти, які потрібно розпізнати, а не спотворювати їх.

Щоб завантажити та обробити відеофайли за допомогою розробленої системи, просто введіть наступну команду в терміналі або командному рядку операційної системи: "*python yolo.py*

-v=input_files/false.mp4", де останнім параметром є каталог відеофайлів. Відеофайли, оброблені програмою, зберігаються в папці "output_files" папки проекту. Приклад роботи застосунку з відео знаходиться в додатку В.

Однак обробка відеофайлів займає значну кількість часу. Наприклад, обробка 5-хвилинного відеофайлу 640x360, 24 кадри в секунду за допомогою архітектури нейронної мережі YOLOv7 займає 34 хвилини.

Під час тестування програмних модулів була проаналізована робота сотень різних зображень. Програмний модуль може розпізнавати більше 78% об'єктів на зображенні (за винятком об'єктів, які не входять до списку об'єктів для навчання цієї нейронної мережі). 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

У роботі досліджуються 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, і вона може передбачати використання спеціалізованих методів і використання алгоритмів навчання загального призначення.

Виходячи з завдань проаналізовано питання в галузі досліджень, розглянуто портфоліо сучасних програм для розпізнавання образів і описано методи розпізнавання образів. Існує 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.

Проаналізовано різні типи нейронних мереж: нейронні мережі прямого поширення. Багатошарові, 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.

Проаналізовано характеристики 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 YOLOv7 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) виробляються 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.

Розроблено застосунок для розпізнавання літаків на зображеннях, в ньому було успішно задіяно нейронну мережу, яку було навчено розміченими даними з датасету COCO. Використовуючи архітектуру YOLOv7 для згорткової мережі, успішно проведено тестування розпізнавання зображень за допомогою програмного додатку, та статистику розпізнавання об’єктів занесено в таблицю.

СПИСОК ЛІТЕРАТУРИ

1. Комп'ютерний зір. – Електрон. дан. – Режим доступу: https://en.wikipedia.org/wiki/Computer_vision, вільний.
2. A Gentle Introduction to Computer Vision. – Електрон. дан. – Режим доступу: https://machinelearningmastery.com/what-is-computer-vision/, вільний.
3. Гудфелов І.А. Глибоке навчання / Гудфелов І.А. // MIT Press. – 2016. – №2. – с. 50.
4. Neural Network fundamentals. – Електрон. дан. – Режим доступу: https://medium.com/datadriveninvestor/neural-network-fundamentals-1956a3000c24, вільний.
5. Краснопоясовський А. С. Інформаційний синтез інтелектуальних систем керування: підхід, що ґрунтується на методі функціонально–статистичних випробувань / А. С. Краснопоясовський. -Суми: Видавництво СумДУ, 2014. -261 с. –Бібліогр.: с. 76.
6. A Comprehensive Guide to Types of Neural Networks. – Електрон. дан. – Режим доступу: https://www.digitalvidya.com/blog/types-of-neural-networks/, вільний.
7. Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN. – Електрон. дан. – Режим доступу: https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html, вільний.
8. YOLO: Real-Time Object Detection. – Електрон. дан. – Режим доступу: https://pjreddie.com/darknet/yolov2/, вільний.
9. SSD object detection: Single Shot MultiBox Detector for real-time processing. – Електрон. дан. – Режим доступу: https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing, вільний.
10. Understanding the mAP Evaluation Metric for Object Detection. – Електрон. дан. – Режим доступу:

https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3, вільний.

11. YOLOv3: An Incremental Improvement. – Електрон. дан. – Режим доступу: https://pjreddie.com/media/files/papers/YOLOv3.pdf, вільний.
12. Інтегроване середовище розробки. – Електрон. дан. – Режим доступу: https://life-prog.ru/ukr/view_zam2.php?id=103&cat=1&page=1, вільний.
13. Опис продукту PyCharm. – Електрон. дан. – Режим доступу: https://itpro.ua/product/jetbrains-pycharm/, вільний.
14. Практичний Python 3 для початківців. – Електрон. дан. – Режим доступу: https://pythonworld.ru/samouchitel-python/, вільний.
15. Опис Python. – Електрон. дан. – Режим доступу: https://uk.wikipedia.org/wiki/Python, вільний.
16. About OpenCV. – Електрон. дан. – Режим доступу: https://opencv.org/about/, вільний.
17. Іванченко М.В. Аналіз існуючих інструментів для розпізнавання образів / М.В. Іванченко // ВНТУ. - 2022. Електрон. дан. - Режим доступу:https://d.conf.vntu.edu.ua/index.php/mn/mn2022/paper/viewFile/141
18. Han J. Data mining: concepts and techniques. – 3rd ed. / J. Han, M. Kamber, J. Pei. – Morgan Kaufmann / Elsevier, 2012. – 744 p.
19. Турти М. В. Теорія однозначних нечітких систем та нейронні мережі // Проблеми автоматики та електрообладнання транспортних засобів: Матеріали Всеукраїнської науково-технічної конференції з міжнародною участю. – Миколаїв : НУК, 2007. – С. 229–237.
20. Hopfield J. J. Neural Networks and Physical Systems with Emergent Collective Computational Abilities // Proc. of the National Academy of Science, 1982. – 79. – P. 2554–2558.
21. Venelinov Topalov, A. Kaynak. Online learning in adaptive neurocontrol schemes with a sliding mode algorithm // IEEE Transactions on Systems,

Man, and Cybernetics, Part B: Cybernetics. – 2001. – Vol. 31. – I. 3. – P. 445–450.

22. Девід Е. Голберг «Дзен та мистецтво генетичного алгоритму». 3rd International Conference on Genetic Algorithms, pp. 80-85.
23. Hastie T. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. – 2nd ed. / T. Hastie, R. Tibshirani, J. Friedman. – Springer-Verlag, 2009. – 746 p.
24. Yann LeCun Leon Bottou, Y. B. Gradient-based learning applied to document recognition / Yoshua Bengio Yann LeCun, Leon Bottou, Patrick Haffner // IEEE. – 1998
25. Duda R. O. Pattern Classification, second ed. / R. O. Duda, P. E. Hart, D. G. Stork. – John Wiley & Sons, New York, 2001. – 738 p.
26. Карпович А. В. Використання згорткових нейронних мереж для задачі класифікації текстів // Міжнародний науковий журнал "Інтернаука". – 2018. – №14. https://doi.org/10.25313/2520-2057-2018-14-4105
27. Son Lam Phung, Abdesselam Bouzerdoum. MATLAB Library for Convolutional Neural Networks. Technical report, http://www.uow.edu.au/~phung/
28. 7 Amazing Examples Of Computer And Machine Vision In Practice Apr 8, 2019 by Bernard Marr - https://www.forbes.com/sites/bernardmarr/2019/04/08/7-amazing-examples-of-computer-and-machine-vision-in-practice/
29. An Energy-Efficient Hardware Implementationof HOG-Based Object Detection at 1080HD60 fps with Multi-Scale Support Amr Suleiman, Vivienne Sze, Journal of Signal Processing Systems, December 2015
30. Intersection over Union (IoU) for object detection by Adrian Rosebrock on November 7, 2016 - https://www.pyimagesearch.com/2016/11/07/intersection- over-union-iou-for-object-detection/

31. You Only Look Once:Unified, Real-Time Object Detection Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, University of Washington, Allen Institute for AI, Facebook AI Research 9 May 2016
32. Bottom-up Object Detection by Grouping Extreme and Center Points, Xingyi Zhou, Jiacheng Zhuo, Philipp Krähenbühl, Jan 23, 2019
33. 6. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, 6 Jan 2016
34. SSD: Single Shot MultiBox Detector, Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, UNC Chapel Hill 2Zoox Inc. Google Inc. University of Michigan, Ann-Arbor, 29 Dec 2016
35. YOLO: Real-Time Object Detection, Joseph Redmon – https://pjreddie.com/darknet/yolo/
36. YOLOv4 — the most accurate real-time neural network on MS COCO dataset. Aleksey Bochkovskiy 21 may 2020 – https://medium.com/@alexeyab84/yolov4-the-most-accurate-real-time-neural- network-on-ms-coco-dataset-73adfd3602fe
37. "Glossary:WebSockets". Mozilla Developer Network. 2015 – https://developer.mozilla.org/en-US/docs/Glossary/WebSockets
38. HTML5 WebSocket: A Quantum Leap in Scalability for the Web By Peter Lubbers & Frank Greco, Kaazing Corporation – http://www.websocket.org/quantum.html
39. RFC 6455 The WebSocket Protocol: Relationship to TCP and HTTP, Ian Fette; Alexey Melnikov (December 2011) – https://tools.ietf.org/html/rfc6455#section-1.7
40. Node.js v14.13.1 Documentation: Child process – https://nodejs.org/api/child_process.html
41. A Python Book: Beginning Python, Advanced Python, and Python Exercises, Dave Kuhlman April 22, 2012

ДОДАТКИ