АНОТАЦІЯ

Через тісний зв’язок ідентифікації та класифікації об’єктів з аналізом відео та розумінням зображення, на це звернули увагу багато дослідників за останні роки. Традиційні методи виявлення об'єктів засновані на написаних вручну особливостях, є малоефективними. Їх продуктивність знижується під час конструювання складних систем, які поєднують в собі кілька низькорівневих особливостях зображення порівняно з детекторами об’єктів та класифікаторами зображень. Завдяки швидкому розвитку глибокого навчання, більш потужні інструменти, які здатні вивчати семантичні, високорівневі та глибші особливості, вводяться для вирішення існуючих проблем з якими не справляються традиційні архітектури. Ці моделі поводяться по-різному в залежно від архітектурі мережі, стратегії навчання та функцій оптимізації тощо.

У роботі я розглянув фреймворк виявлення об'єктів TensorFlow Object Detection API, який базується на основі глибокого навчання. Робота починається з короткого вступу до історії глибокого навчання та її репрезентативного інструмента, а саме загорткової нейронної мережі (CNN). Потім робота зосереджена на типових архітектурах класифікації об’єктів та описано структуру цих мереж. Для ідентифікації та класифікації біооб’єктів на зображеннях, була розроблена нейронна мережа в основі якої лежить мережа SSD ResNet50 v1 FPN 640x640, яка найбільше підходить для вирішення цієї задачі.

Ключові слова: глибоке навчання, нейронна мережа, ідентифікація, класифікація, виявлення об’єктів.

Розмір пояснювальної записки – __ аркушів, містить 37 ілюстрації, 2 таблиць, __ додатків.

SUMMARY

Due to the close connection between the identification and classification of objects with video analysis and image understanding, this has been pointed out by many researchers in recent years. Traditional methods of detecting objects based on handwritten features are ineffective. Their performance is reduced when designing complex systems that combine several low-level image features compared to object detectors and image classifiers. Due to the rapid development of deep learning, more powerful tools that are able to study semantic, high-level and deeper features are introduced to solve existing problems that traditional architectures do not cope with. These models behave differently depending on the network architecture, learning strategy and optimization functions, and so on.

In this paper, I reviewed the TensorFlow Object Detection API, which is based on deep learning. The work begins with a brief introduction to the history of deep learning and its representative tool, namely the wrapped neural network (CNN). The work then focuses on typical object classification architectures and describes the structure of these networks. To identify and classify bioobjects in images, a neural network based on the SSD ResNet50 v1 FPN 640x640 network best suited for this task has been developed.

Keywords: deep learning, neural network, identification, classification, object detection.

The size of the explanatory note - __ sheets, contains 37 illustrations, 2 tables, __ appendices.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 8
ВСТУП 9
1. ОЗНАЙОМЛЕННЯ З ПРЕДМЕТНОЮ ОБЛАСТЮ, АНАЛІЗ ТА ОГЛЯД НАУКОВИХ ТА ТЕХНІЧНИХ ДЖЕРЕЛ 11
1.1. Поняття “нейронна мережа” та принцип роботи 11
1.1.1. Штучні нейронні мережі 12
1.1.2. Згорткові нейронні мережі 14
1.1.3. Рекурентна нейронна мережа 16
1.2. Історія нейронних мереж 18
1.3. Аналіз існуючих рішень 22
1.3.1. Огляд системи “Google Lens” 22
1.3.2. Огляд системи “TapTapSee” 24
1.3.3. Аналіз системи “CamFind” 25
1.4. Порівняльний аналіз існуючих програмних рішень 27
1.5. Висновки до розділу 27
2. АНАЛІЗ АРХІТЕКТУР ТА ВИДІВ НАВЧАННЯ МЕРЕЖ 28
2.1. Вибір архітектури для класифікації біооб’єктів на зображеннях 28
2.1.1. Аналіз характеристик та принципу будови моделі AlexNet 28
2.1.2. Аналіз характеристик та принципу будови моделі VGGNet 30
2.1.3. Аналіз характеристик та принципу будови моделі ResNet 31
2.2. Вибір технології реалізації моделі та середовища 34
2.2.1. Опис фреймоврка Tensorflow 34
2.3. Методи навчання нейронних мереж 37
2.3.1. Контрольоване навчання нейронної мережі 37
2.3.2. Не контрольоване навчання нейронної мережі 39
2.4. Поняття функція активації 41
2.4.1. Функція активації ReLU 42
2.4.2. Функція активації Sigmoid 43
2.4.3. Функція активації Softmax 43

2.5. Поняття перенавчання та недонавчання моделі 44
2.5.1. Методи боротьби з недонавчанням 45
2.5.2. Методи боротьби з перенавчанням 46
2.6. Висновки до розділу 47
3. ПРОЕКТУВАННЯ ТА РОЗРОБКА МОДЕЛІ 49
3.1. Вибір архітектури та платформи для розробки моделі 49
3.1.1. Дистрибутив Anaconda 3 49
3.1.2. Мова програмування Python 50
3.1.3. Фреймворк TensorFlow та його переваги 51
3.2. Збір набору даних і тренування моделі 52
3.2.1. Вебсервер TensorBoard 56
3.3. Висновки до розділу 60
4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ ... 62
4.1. Охорона праці 62
4.1.1. Дії працівників в аварійних ситуаціях 64
4.1.2. Надання першої медичної допомоги 65
4.2. Підвищення стійкості об’єктів господарської діяльності в умовах надзвичайних ситуацій 66
4.2.1. Стійкість роботи об’єктів господарської діяльності і фактори, що впливають на їх стійкість 66
ВИСНОВКИ 70
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 72
ДОДАТКИ 74
ДОДАТОК А Технічне завдання
ДОДАТОК Б Тези
ДОДАТОК В Лістинг коду
ДОДАТОК Г Диск з проектом

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (application programming interface) – програмний посередник, який дозволяє двом додаткам спілкуватися один з одним;

ANN (artificial neural network) – це обчислювальна модель, яка імітує роботу нервових клітин у мозку людини;

CNN (convolutional neural network) – це тип штучної нейронної мережі, що використовується для розпізнавання та обробки зображень;

RNN (recurrent neural network) – це тип штучної нейронної мережі, яка використовує послідовні дані або дані часових рядів;

XOR (exclusive or) – логічна операція, яка є істинною тоді і тільки тоді, коли її аргументи відрізняються;

FFNN (feedforward artificial neural network) – це штучна нейронна мережа, в якій зв’язки між вузлами не утворюють цикл;

MLP (multilayer perceptron) – це штучна нейронна мережа з прямим зв’язком, яка генерує набір виходів із набору вхідних даних;

IDE (integrated development environment) – програмний додаток, який надає програмістам комплексні засоби для розробки програмного забезпечення;

OD (object detection) – процес ідентифікації та класифікації об’єкта на зображені;

ВСТУП

Нейронні мережі з'явилися в 1950-х роках, але не викликали значного інтересу до 1980-х років з появою зворотного поширення. Завдяки теоретичним і практичним результатам, отриманих протягом останніх десятиліть, нейронні мережі стали широко використовуваним інструментом у різних галузях. Вони довели свою ефективність у керуванні процесами, моделюванні, розпізнаванні образів та обробці сигналів тощо. Однак вони залишаються предметом великого інтересу для дослідників, які бажають покращити свою продуктивність та розширити область діяльності додатків

Актуальність дослідження. У сучасному світі системи нейронних мереж допомагають людям у широкому спектрі задач, таких як класифікація зображень, машинний переклад, моделювання мови. Вони допомагають скоротити використання людських ресурсів у багатьох сферах. Саме тому зростає актуальність розробки систем, моделей, методів та засобів обробки зображень, які давали б можливість автоматизації та підвищення точності і швидкодії вказаних процесів. На даний момент комп’ютерний зір є одним із найсвіжіших та одним з найактуальніших напрямків, а основними його задачами є:

1) розпізнавання тексту;
2) біометрія;
3) локалізація об’єктів;
4) відеоаналітика;
5) ідентифікація об’єкту;
6) розпізнавання інформації, що містить об’єкт на зображенні;
7) графічні редактори;
8) управління автомобілем.

Мета і завдання дослідження. Завдання класифікації та ідентифікації об’єктів полягає у віднесенні вихідних даних до певного класу за допомогою

відокремлення основних характеристик, що виділяють ці дані, на фоні решти незначимих даних та виділення цього об’єкту за допомогою обмежувальної рамки. Класифікація об’єктів це одна з найбільш фундаментальних задач в області інтелектуальних систем та нейронних мереж оскільки принципове значення проблеми міцно пов'язане з питанням про те, що може і що не може робити машина. Тому необхідно, щоб алгоритми виявлення об’єктів були високоточними.

Протягом останнього десятиліття методи виявлення візуальних об’єктів розвинулись досить швидко. Після того, як CNN стали популярними, звичайні алгоритми отримання ознак почали замінятись згортковими нейронними мережами. Першим хто доказав важливість архітектури CNN для отримання ознак в класифікаціях зображень, був Крижевський.

Після впровадження архітектур VGG та ResNet важливість нейронних мереж для отримання ознак ще більше зросла.

З практичної точки зору вирішення завдання класифікація об’єктів дозволяє автоматизувати процеси, які до цих пір асоціювались лише з діяльністю людини. Враховуючи те, що одним з найбільших джерел інформації є зорове сприйняття, до основних задач, що вирішуються в сучасних інформаційних системах можна віднести завдання класифікації та ідентифікації об'єктів в системах комп’ютерного зору, а також розпізнавання букв, рослин, тварин, людей або номерів автомобілів, ідентифікація за відбитками пальців, розпізнавання жестів та виявлення цілей.

1. ОЗНАЙОМЛЕННЯ З ПРЕДМЕТНОЮ ОБЛАСТЮ, АНАЛІЗ ТА ОГЛЯД НАУКОВИХ ТА ТЕХНІЧНИХ ДЖЕРЕЛ

1.1. Поняття “нейронна мережа” та принцип роботи

Нейронна мережа – це набір нейронів, які приймають інформацію як вхідні дані і разом з інформацією з інших вузлів виробляють вихідні дані без запрограмованих правил. По суті, вони вирішують проблеми методом спроб та помилок.

Нейронні мережі працюють за принципом роботи мозку людини та тварин. Хоча нейронні мережі досить розвинені, щоб перемагати людей у таких іграх, як шахи та го, їм все ще не вистачає когнітивних здібностей.

Нейронна мережа складається з міцно пов'язаних вузлів обробки, схожих на нейрони у головному мозку. Кожен вузол може бути підключений до різних вузлів на декількох рівнях вище та нижче від нього. Ці вузли переміщують дані мережі лише в одному напрямку. Вузол “загоряється” як нейрон, коли передає інформацію наступному вузлу.

Проста нейронна мережа має вхідний шар, вихідний шар та прихований шар між ними. Мережа з більш ніж трьома рівнями, включаючи вхідний та вихідний, називається мережею глибокого навчання (від англ. Deep learning). У мережі глибокого навчання кожен рівень вузлів навчається на основі вихідних даних попереднього рівня. Чим більше шарів, тим вища здатність розпізнавати складнішу інформацію на основі даних із попередніх шарів.

Мережа приймає рішення, привласнюючи кожному підключеному вузлу номер, так звана “вага”. Вага є цінність інформації, привласненої окремому вузлу (тобто її корисність для правильної класифікації інформації). Коли вузол отримує інформацію з інших вузлів, він обчислює вагу чи загальну цінність цієї інформації. Якщо число перевищує певний поріг, інформація передається на наступний рівень. Якщо вага нижча, то інформація не передається.

У новосформованій нейронній мережі всі ваги та пороги встановлюються на випадкові числа. У міру того, як навчальні дані надходять на вхідний шар, ваги та пороги уточнюються, щоб постійно видавати правильні вихідні дані.

Чи то біологічна чи штучна, сила нейронної мережі залежить від того, як прості нейрони пов'язані між собою, утворюючи складну систему.

Кожен нейрон може приймати найпростіші рішення на основі математичних розрахунків. Водночас багато нейронів можуть аналізувати складні проблеми та давати точні відповіді. Дрібна мережа складається з вхідного, прихованого та вихідного шарів. Глибока нейронна мережа має більше одного прихованого шару, що збільшує складність аналізованих проблем.

Нейронна мережа вчиться виконувати завдання, досліджуючи помічені навчальні приклади. Зразки повинні бути позначені, щоб мережа могла навчитися розрізняти елементи, використовуючи візуальні шаблони з “підписами”.

Нейронна мережа – це коригуючий контур зворотного зв'язку, який надає більшу вагу даним, що приводять до правильних припущень, і меншу вага даним, через які мережа допустила помилку. Ця характеристика відома як зворотне поширення, вона змушує мережу визначати правильні відповіді та ігнорувати неправильні відповіді.

1.1.1. Штучні нейронні мережі

Одиночний перцептрон (або нейрон) можна як зобразити логістичну регресію (рис. 1.1). Штучна нейронна мережа, або ANN, є групою з кількох перцептронів / нейронів на кожному рівні. ANN також відома як нейронна мережа з прямим зв'язком, тому що вхідні дані обробляються тільки в одному напрямку:

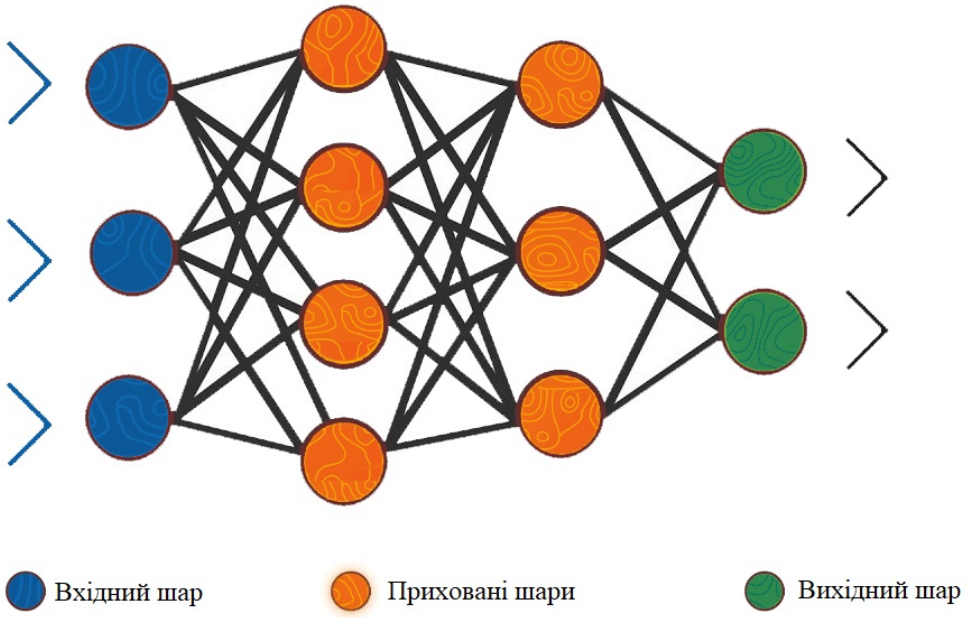

Рисунок 1.1 – Структура ANN

Як можна замітити, ANN складається з 3-х рівнів - вхідного, прихованого та вихідного. Вхідний шар приймає вхідні дані, прихований шар обробляє вхідні дані, а вихідний шар видає результат. По суті, кожен шар намагається дізнатися про певні ваги.

ANN можна використовувати для вирішення наступних задач пов’язаними з такими даними:

- табличними;
- зображення;
- текстовими.

Перевагою штучної нейронної мережі є те, що вона здатна навчатися будь-якої нелінійної функції. Отже, ці мережі широко відомі як універсальні апроксиматори функцій. ANN можуть запам'ятовувати ваги, які зіставляють будь-яке введення з висновком.

Однією з основних причин цієї особливості є функція активації. Функції активації надають мережі нелінійних властивостей. Це допомагає мережі вивчити будь-які складні відносини між вхідними та вихідними даними.

До недоліків такої мережі можна віднести те, що швидкість навчання такої мережі сильно залежить від потужності комп’ютера і інколи їх поведінку не можливо розрахувати наперед, що приводить до непередбачуваних результатів. Також, до основних мінусів ANN відносять відсутність єдиної правильної структури, досягти коректної будови можна лише методом проб та помилок.

1.1.2. Згорткові нейронні мережі

Згорткові нейронні мережі (CNN) зараз дуже популярні серед спільноти глибокого навчання. Ці CNN моделі використовуються у великій кількості додатків і вони особливо поширені в проектах обробки зображень та відео.

Будівельними блоками CNN є фільтри, також відомі як ядра. Ядра використовуються для отримання відповідних особливостей з вхідних даних за допомогою операції згортки. Згортка зображення за допомогою фільтрів призводить до карти особливостей (рис. 1.2):

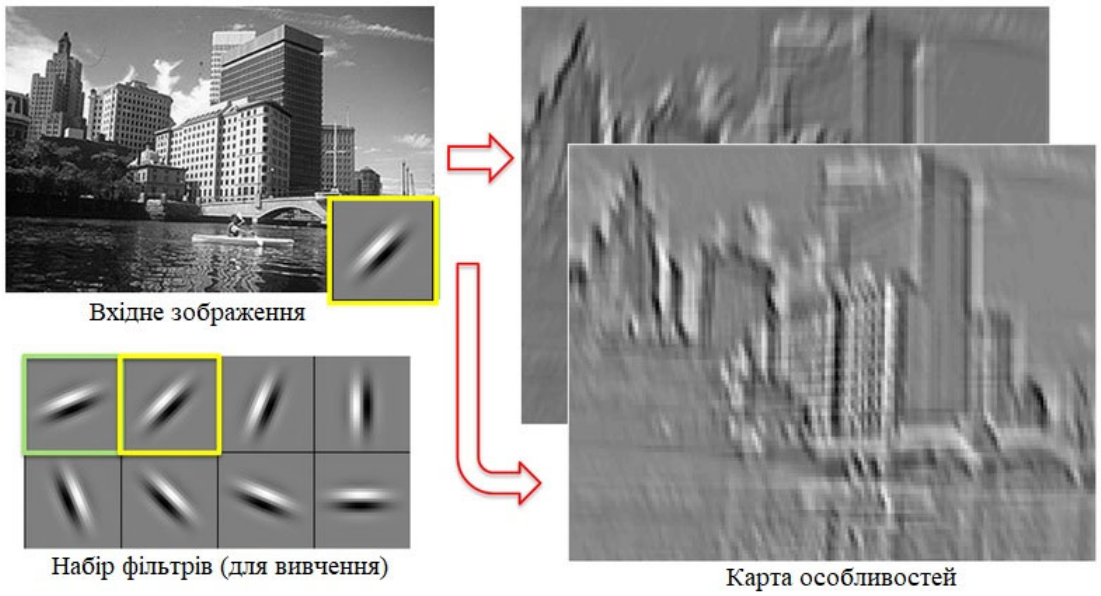

Рисунок 1.2 – Карта особливостей мережі CNN

Хоча згорткові нейронні мережі були створенні для вирішення проблем, пов'язаних із даними зображень, вони також ефективно працюють із послідовними входами.

Особливості згорткової мережі:

- CNN вивчає фільтри автоматично. Ці фільтри допомагають отримувати необхідні та актуальні особливості з вхідних даних (рис. 1.3);

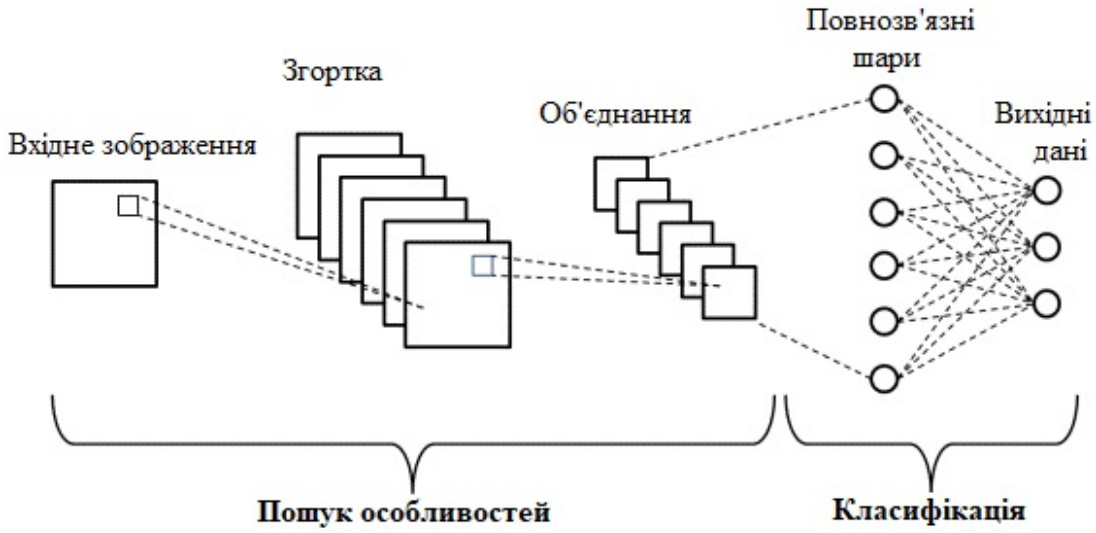

Рисунок 1.3 – Принцип будови CNN

- CNN фіксує просторові особливості зображення. Просторові об'єкти відносяться до розташування пікселів та співвідношення між ними на зображенні. Вони допомагають нам точно ідентифікувати об'єкт, розташування об'єкта, а також його зв'язок з іншими об'єктами на зображенні;
- CNN також слідує концепції спільного використання параметрів. Один фільтр застосовується до різних частин вхідних даних для створення карти особливостей.

Мінусами такої мережи є те, що вона не здатна визначити позицію об’єкта, а вхідні дані повинні бути просторово однаковими. Також, така мережа потребує багато тренувальних даних для того, щоб працювати ефективно та видавати точні результати.

1.1.3. Рекурентна нейронна мережа

Обмежувальний цикл на прихованому шарі ANN, перетворює цю мережу на RNN (рис. 1.4).

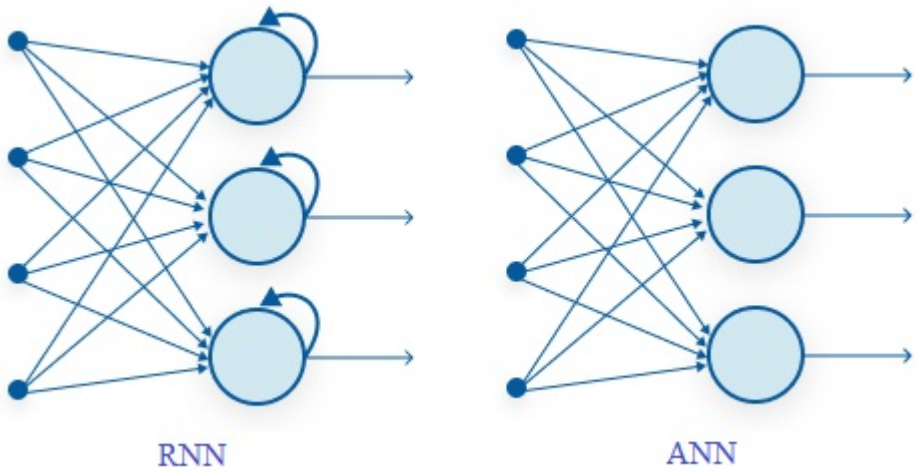

Рисунок 1.4 – Структурна різниця між RNN та ANN

Як можна замітити, RNN має з'єднання, що повторюється, в прихованому шарі. Цей обмежувальний цикл гарантує, що у вхідних даних буде записано послідовну інформацію.

Ми можемо використовувати рекурентні нейронні мережі для вирішення проблем, пов'язаних із:

- даними, отриманими за допомогою повторних вимірювань протягом певного періоду часу;
- текстовими даними;
- аудіоданими.

Особливості рекурентної мережі:

- RNN фіксує послідовну інформацію, яка є у вхідних даних, тобто залежність між словами в тексті, роблячи прогнози;

- RNN поширюють параметри на різних часових відбитках. Це явище відоме як поділ параметрів. Воно призводить до меншої кількості параметрів для навчання та збільшує обчислювальну ефективність.

Глибокі RNN (RNN з великою кількістю часових відбитків) страждають від проблеми градієнта, що зникає і збільшується (рис. 1.5), що є загальною проблемою для всіх типів нейронних мереж. Також, варто сказати, що тренування такого типу мереж є досить не простим завданням.

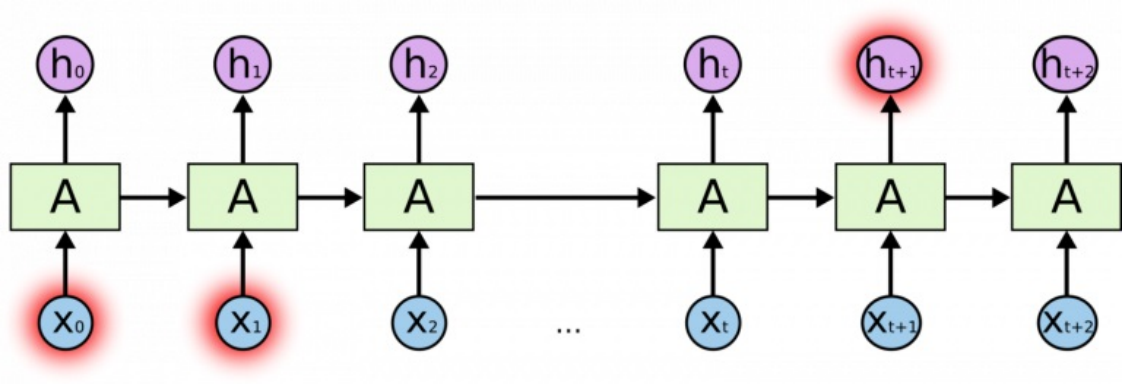

Рисунок 1.5 – Проблема зникаючого градієнта

Таблиця 1.1 – Порівняння нейронних мереж

	ANN	CNN	RNN
Дані	Табличні дані	Дані зображень	Послідовні дані
Повторні зв’язки	-	-	+
Поширення параметрів	-	+	+
Просторове відношення	-	+	-
Зникаючий градієнт	+	+	+

1.2. Історія нейронних мереж

Не дивно, що ідея нейронних мереж виникла як модель того, як нейрони в мозку функціонують, що отримала назву “коннекціонізм” та використовує зв'язані між собою ланцюжки для імітації інтелектуальної поведінки. В 1943 нейрофізіолог Уоррен Мак-Каллок і математик Уолтер Піттс зобразили її простою електричною схемою. Дональд Хебб розвинув цю ідею далі у своїй книзі “Організація поведінки” (1949), припустивши, що нейронні шляхи посилюються при кожному наступному використанні, особливо між нейронами, які мають тенденцію спрацьовувати одночасно, запустивши довгий процес до кількісної оцінки складних процесів мозок.

Існують дві основні концепції, які є попередниками нейронних мереж:

- “Threshold Logic” (логіка на порогових елементах) – перетворення безперервного входу в дискретний вихід;
- “Hebbian Learning” (навчання нейронних мереж методом Хебба) – модель навчання, заснована на нейронній пластичності, запропонована Дональдом Хеббом у його книзі “Організація поведінки”, яка часто резюмується фразою: “між клітинами, які збуджуються одночасно, виникає міцний зв'язок” [1].

Вони були запропоновані у 1940-х роках. У 1950-х роках, коли дослідники почали намагатися перетворити ці мережі на обчислювальні системи, перша мережа “Hebbian” була успішно реалізована в Массачусетському технологічному інституті у 1954 році.

Приблизно в цей же час Френк Розенблатт, психолог із Корнелла, працював над відносно простими системами прийняття рішень, присутніх в очах мухи, які лежать в основі та визначають її реакцію втечі. Намагаючись зрозуміти та кількісно оцінити цей процес, він запропонував ідею перцептрона в 1958 році, назвавши його перцептроном Марка I. Це була система з простим

співвідношенням вхід-вихід, змодельована на нейроні Мак-Каллока-Піттса, запропонована в 1943 нейробіологом Уорреном С. та логіком Уолтером Піттсом для пояснення складних процесів прийняття рішень у мозку з використанням лінійного порогового елемента. Нейрон Мак-Каллока-Піттса приймає вхідні дані, приймає середню число і повертає ‘0’, якщо результат нижчий від порогового значення, і ‘1’ в іншому випадку (рис. 1.6).

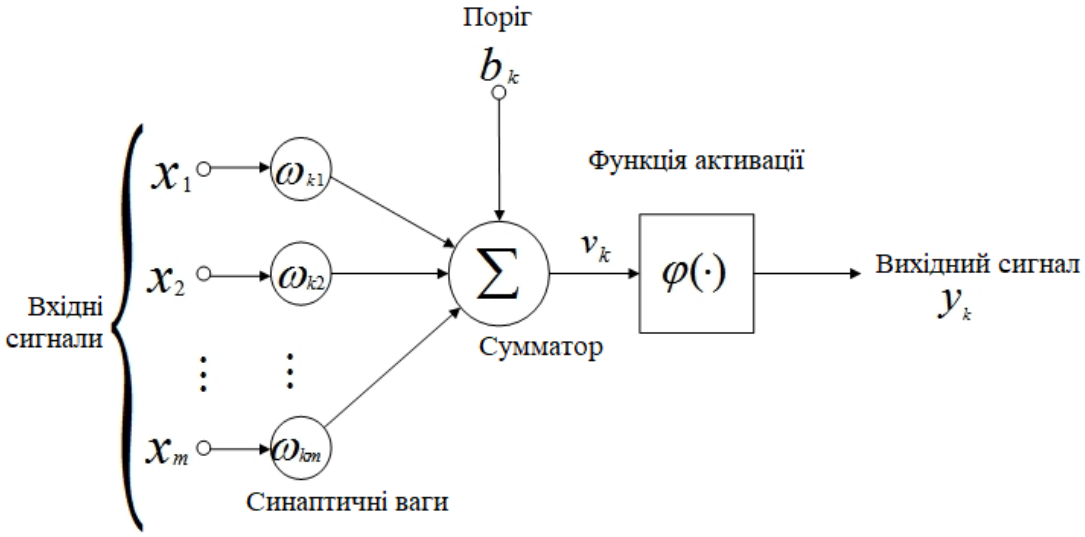

Рисунок 1.6 – Модель нейрона Мак-Каллока-Піттса

Голова особливість Mark I Perceptron (рис. 1.7) у тому, що його ваги “навчатимуться” через вхідні дані, що послідовно передаються, мінімізуючи при цьому різницю між бажаним і фактичним виходом.

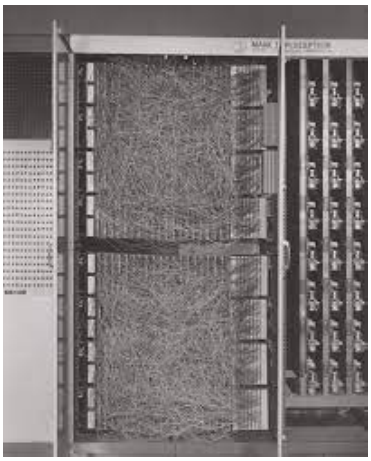

Рисунок 1.7 – Марк I Перцептрон

Головним недоліком цього перцептрона було те, що він міг тільки навчитися ділити класи, що лінійно розділялися, роблячи просту, але нелінійну схему виключної диз’юнкції непереборним бар'єром (рис. 1.8).

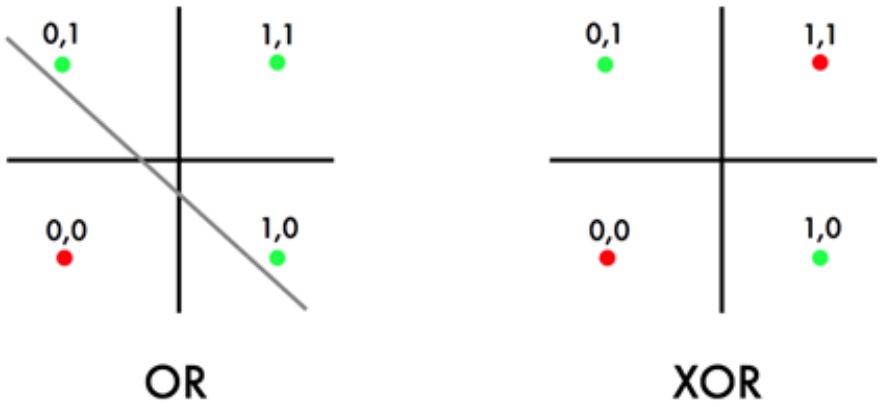

Рисунок 1.8 – Проблема XOR

Незважаючи на безладне і дещо незадовільне використання машинного навчання для кількісної оцінки систем прийняття рішень крім мозку, сьогоднішні штучні нейронні мережі є чимось іншим, ніж просто кілька шарів перцептронів.

Приблизно в цей час все стало швидко розвиватися з нейронними мережами, і в 1959 році в Стенфорді Бернард Відроу і Марсіан Хофф розробили першу нейронну мережу, успішно застосовану до реальної проблеми. Ці системи були названі ADALINE та MADALINE в честь використання у них кількох елементів ADAptive LINear, останній з яких був спеціально розроблений для усунення шуму в телефонних лініях та використовується досі. Головною відмінністю між цими нейронами та перцептронами є дані, які вони повертають як вихідні, які в цьому випадку були зваженими вхідними даними.

Невеличкі досягнення завжди викликали ажіотаж, цей успіх від них нічим не відрізнявся і породив велику зацікавленість навколо можливостей та потенціалу нейронних мереж, хоча тим часом науковці зіштовхувалися з новими проблемами.

Як і у випадку з багатьма “близькими” спробами раніше, ми все ще були далекі до того, щоб оголосити про створення свідомих створених людиною істот. Одна з проблем полягала в непрактично тривалому часі роботи, необхідному для роботи цих мереж, враховуючи, що це були 60-ті роки, а також нездатності вивчати прості логічні схеми XOR.

У 1969 році цьому прийшов кінець з публікацією книги “Перцептрони” Марвіна Мінська, засновника лабораторії штучного інтелекту MTI (Массачусетський технологічний інститут), та Сеймура Паперта, директора лабораторії. У книзі переконливо стверджується, що підхід Розенблатта до нейронних мереж, заснований на єдиному сприйнятті, не може бути ефективно перетворений на багатошарові нейронні мережі. Для оцінки правильних відносних значень ваг нейронів, розподілених по шарах, на основі остаточного результату потрібна велика, якщо не нескінченна кількість ітерацій, а обчислення триватимуть дуже довго.

Мінські у своєму тексті виклав ці та інші проблеми, пов'язані з нейронними мережами, і фактично привів ширшу наукову спільноту і фінансові установи, до висновку, що подальші дослідження в цьому напрямку ні до чого не приведуть. Ефект від цього тексту був потужним і привів до скорочення фінансування настільки, що протягом наступних 10-12 років ніхто з найбільших дослідницьких інститутів того часу, а також дрібніших, не візьметься за проект, який був пов’язаний із приреченими нейронними мережами. Епоха, яку тепер називають “зима штучного інтелекту”, почалася.

Кінець цієї довгої зими наступив цього ж десятиліття у 1982 році в Національній академії наук, коли Джон Хопфілд представив свою доповідь про те, що незабаром стало відомо як “мережа Хопфілда”, а в тому ж році Японія оголосила на американо-японській конференції з кооперативних/конкурентних нейронних мереж. про намір розпочати розробку п'ятого покоління нейронних мереж [2]. Це змусило фінансування знову почати надходити зі скарбниці Америки. Незабаром Американський інститут фізики в 1985 році організував щорічні збори “Нейронні мережі у обчисленнях” [3], за якими пішла перша

Міжнародна конференція з нейронних мереж, організована Інститутом інженерів з електротехніки та електроніки (IEEE) у 1987 році [4].

Однак це було серйозним перевідкриттям концепції, яка вже існувала з 60-х років, яка допомогла нейронним мережам вибратися з передчасної могили. Зворотне поширення – метод, розроблений дослідниками у 60-их роках і безперервно розвивався до “зими штучного інтелекту”, цей метод, заснований на інтуїції, який приписує зменшення значущості кожній події в міру того, як він просувається далі в ланцюжку подій. Першою людиною, яка побачила потенціал у цьому методі для нейронних мереж і вирішила питання про те, як це буде реалізовано на MLP, був Пол Вербос [5], який частково надихнувся його застосуванням у людському розумі та роботою Фрейда щодо зворотного присвоєння значень, написав докторську дисертацію у якій пояснив їх важливість. Однак ця робота не була помічена ніким у співтоваристві, доки Паркер не опублікував звіт про свою роботу у MTI 1985 року. Тільки після того, як Румельхарт, Хінтон та Вільямс повторно винайшли цю техніку і перевидали в ясних та докладних рамках, ця техніка згодом захопила все суспільство. Ті ж автори звернулися до специфічних недоліків, викладених Мінським у його публікації 1969 року у пізнішому тексті.

Таким чином, до 1990-х років нейронні мережі остаточно повернулися, по-справжньому завоювавши увагу світу і, нарешті, зрівнявшись з очікуваннями, а можливо навіть перевершивши їх.

1.3. Аналіз існуючих рішень

1.3.1. Огляд системи “Google Lens”

Google Lens – це технологія на основі штучного інтелекту, яка використовує камеру вашого смартфона та глибоке машинне навчання, щоб не тільки виявляти

об'єкт перед об'єктивом камери, але й класифікувати його та пропонувати такі дії, як сканування, переклад, покупки та багато іншого.

Lens був одним з найбільших анонсів Google у 2017 році та ексклюзивною функцією Google Pixel, коли цей телефон був випущений. З того часу Google Lens з'явився на більшості пристроїв Android – якщо у вас його немає, програма доступна для завантаження в Google Play.

Google Lens дозволяє вам навести телефон на щось, наприклад, на конкретну квітку, а потім запитати Google Assistant, на який об'єкт ви вказуєте. Вам не тільки повідомлять відповідь, але й запропонують варіанти, що базуються на об'єкті, наприклад, найближчих флористів, у разі квітки.

Крім описаних вище варіантів, Google Lens також пропонує такі функції:

- перекладач: ви можете направити свій телефон на текст і, підключивши Google Translate, перекладати текст у реальному часі. Це також працює і в автономному режимі;
- інтелектуальне сканування тексту: ви можете навести камеру свого телефону на текст, потім виділити цей текст у Google Lens і скопіювати його на свій телефон (рис. 1. 9);

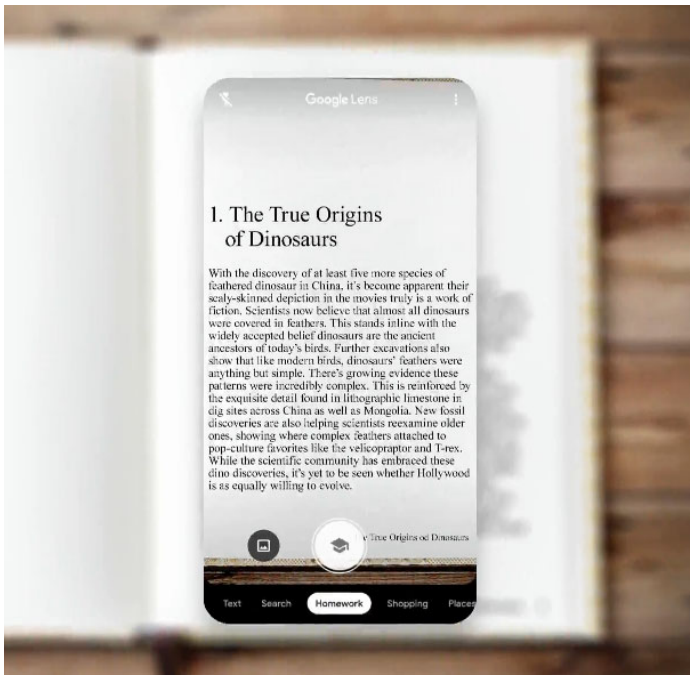

Рисунок 1.9 – Приклад функції інтелектуального сканування тексту

- інтелектуальний пошук тексту: коли ви виділяєте текст у Google Lens, ви також можете знайти його за допомогою пошукової системи Google. Це зручно, наприклад, якщо потрібно знайти визначення слова;
- покупки: якщо ви бачите сукню під час покупок, Google Lens може ідентифікувати цей предмет і аналогічні предмети одягу. Це працює практично з будь-яким предметом, про який ви тільки можете подумати;
- питання про домашнє завдання Google: можете просто переглянути питання і подивитися, що вигадає Google;
- пошук навколо вас: якщо ви наведете камеру об’єкти які вас оточують, Google Lens виявить та ідентифікує їх. Це можуть бути відомості про тварин, пам'ятки або тип їжі.

1.3.2. Огляд системи “TapTapSee”

TapTapSee – це програма для розпізнавання об'єктів яка доступна у магазинах App Store та Google Play. Програма призначена для допомоги сліпим і слабозорим, скануючи об'єкти через камеру свого телефону.

Інтерфейс користувача TapTapSee на сьогоднішній день є найбільш зручним для користувача. Все, що вам потрібно зробити, це торкнутися екрана, і зображення буде скановано незалежно від типу об'єкта. Вам не потрібно буде змінювати категорію певних об'єктів.

Найбільший недолік TapTapSee – це час, необхідний обробки зображення та його сканування. Після сканування зображення обробка зображення може тривати від 15 секунд до хвилини.

Однак нестача швидкості компенсується точністю та дальністю. Додаток також може визначати назви фільмів та відеоігор на основі сканування їх етикеток.

Рисунок 1.10 – Приклад роботи TapTapSee

1.3.3. Аналіз системи “CamFind”

Заснований на чудовій технології розпізнавання зображень, CamFind буде шукати те, що ви бачите, і надавати результати як жодна інша система візуального пошуку, представлена сьогодні на ринку.

Подумайте тільки, якщо ви стоїте на кухні, дивитеся на м’ясо курки і не знаєте, що робити з ним на обід, CamFind може допомогти вам з цим. Сфотографуйте це м’ясо за допомогою CamFind, виконайте пошук і вам буде представлена не тільки інформація про “курку”, але ви також отримаєте список різних рецептів.

Не потрібно сидіти перед вікном браузера і вводити точну послідовність або набір слів для отримання потрібних результатів. Тепер ви можете просто зробити знімок і CamFind зробить свою справу (рис. 1.11).

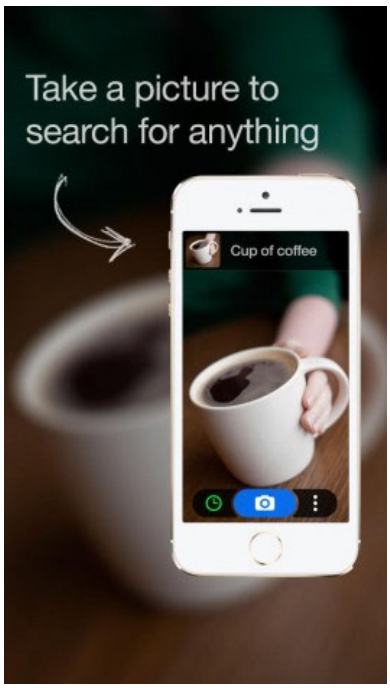

Рисунок 1.11 – Приклад роботи CamFind

CamFind наповнений функціями, які роблять пошук на основі зображень простим та цікавим, дозволяючи вам використовувати пошукові запити, виконані іншими, та дізнатися більше про те, як ефективно робити знімки для отримання точних результатів:

- зображення, пов'язані з тим, що ви зняли на камеру;
- порівняння цін з посиланням на онлайн-продавців;
- пошук адрес та пов'язаними з ними місця;
- розпізнавання DVD та постерів до фільмів;
- завантажте та збережіть раніше зроблені фотографії з вашої галереї;
- сканер QR та штрих-коду;
- голосовий або текстовий пошук також доступний.

1.4. Порівняльний аналіз існуючих програмних рішень

У результаті огляду та аналізу трьох систем, можна провести їх порівняння визначивши переваги та недоліки кожної (табл. 1.2).

Таблиця 1.2 – Порівняння розглянутих систем

	Google Lens	TapTapSee	CamFind
Функціональність	+	-	+
Швидкість	+	-	+
Точність	+	+	-
Зручність інтерфейсу	+	-	+
Кросплатформеність	-	+	+
Відкрите API	-	-	-
Ціна	+	+	+
Розширення функціоналу	-	-	-

Отже, можна зробити висновок проаналізувавши таблицю вище. Всі з розглянутих додатків окрім TapTapSee який призначений для людей з обмеженими можливостями, мають широкий функціонал. Додатки Google Lens та CamFind класифікують об’єкти на зображені досить швидко, але програють по точності TapTapSee. Тільки один з додатків не являється кросплатформеним, тобто їх можна використовувати на системах IOS та Android.

1.5. Висновки до розділу

У цьому розділі ми спробували уявити мінімум понять про нейронні мережі, такі як типи нейронних мереж та різні області їх застосування, щоб мати можливість використовувати їх для інших досліджень.

2. АНАЛІЗ АРХІТЕКТУР ТА ВИДІВ НАВЧАННЯ МЕРЕЖ

2.1. Вибір архітектури для класифікації біооб’єктів на зображеннях

Проаналізувавши таблицю 1.1, можна зробити висновок, що для задачі класифікації зображень найбільше підійдуть згорткові нейронні мережі (CNN). Найяскравішими представниками CNN, є такі моделі як: VGGNet, AlexNet та ResNet.

2.1.1. Аналіз характеристик та принципу будови моделі AlexNet

Модель AlexNet, яка використовувала 8-шарову CNN, виграла конкурс ImageNet Large Scale Visual Recognition Challenge 2012 [6] із феноменально великим відривом.

У першому шарі AlexNet форма вікна згортки становить 11 на 11. Оскільки більшість зображень у ImageNet більш ніж у десять разів вище і ширше, ніж зображення MNIST, відповідно об'єкти даних ImageNet зазвичай займають більше пікселів. Отже, для захоплення об'єкта потрібне більше вікно згортки. Форма вікна згортки у другому шарі зменшується до 5 на 5, а потім до 3 на 3. Крім цього, після першого, другого і п'ятого згорткових шарів мережа додає максимальну кількість шарів об'єднання з формою вікна 3 на 3 і кроком 2.

Після останнього згорткового шару є два повнозв'язні шари з 4096 виходами. Повну структуру мережі AlexNet можна побачити на рисунку 2.1.

AlexNet Network - Structural Details													
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
Total													**62,378,344**

Рисунок 2.1 – Будова мережі AlexNet

Крім того, у AlexNet функцією активації виступає не сигмоїдна, а більш простіша функція активації ReLU. З одного боку, обчислення функції активації ReLU простіше, наприклад, вона не має операції зведення у ступінь, яка присутня у сигмоїдної функції. З іншого боку, функція активації ReLU спрощує навчання моделі за допомогою різних методів ініціалізації параметрів. Це пов'язано з тим, що коли вихідний сигнал сигмовидної функції активації дуже близький до 0 або 1, градієнт цих областей майже дорівнює 0, тому зворотне поширення не може продовжувати оновлювати деякі параметри моделі. Навпаки, градієнт функції активації ReLU в позитивному інтервалі завжди дорівнює 1. Отже, якщо параметри моделі не ініціалізовані належним чином, сигмоїдна функція може отримати градієнт практично рівний 0 в позитивному інтервалі, через це навчання моделі не є ефективним.

Хоча ReLU допомагає з проблемою зникаючого градієнта, через свою необмеженість, вивчені змінні можуть стати невиправдано високими. Щоб запобігти цьому, AlexNet запровадила нормалізацію локального реагування (LRN). Ідея LRN полягає в тому, щоб виконати нормалізацію в околиці пікселів, підсилюючи збуджений нейрон, водночас подавляючи навколишні нейрони.

AlexNet також вирішує проблему перенавчання, використовуючи шари відсіву, коли з'єднання розривається під час навчання з ймовірністю p = 0,5. Хоча це дозволяє уникнути надмірного припасування мережі, допомагаючи їй вийти з

поганих локальних мінімумів, кількість ітерацій, необхідних для збіжності, також подвоюється.

2.1.2. Аналіз характеристик та принципу будови моделі VGGNet

VGGNet виник через необхідність зменшити кількість параметрів у шарах CONV та скоротити час навчання.

Існує кілька варіантів VGGNet (VGG16, VGG19 і т. д.), які відрізняються лише загальною кількістю рівнів у мережі. Структурні деталі мережі VGG16 показані нижче (рис. 2.2).

VGG16 - Structural Details													
#	Input Image			output			Layer	Stride	Kernel		in	out	Param
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
Total													**138,423,208**

Рисунок 2.2 – Структура моделі VGGNet

VGG16 має загалом 138 мільйонів параметрів. Тут важливо відзначити, що всі ядра conv мають розмір 3 на 3, а ядра maxpool мають розмір 2 на 2 з кроком два.

Ідея, що лежить в основі ядер фіксованого розміру, полягає в тому, що всі згорткові ядра змінного розміру, використовувані Alexnet (11 на 11, 5 на 5, 3 на 3), можуть бути відтворені шляхом використання декількох ядер 3 на 3 в якості будівельних блоків. Реплікація визначається рецептивним полем, що охоплюється ядрами.

Розглянемо наступний приклад. Скажімо, ми маємо вхідний шар розміром 5 на 5 і на 1. Реалізація згорткового шару розміром ядра 5 на 5 і кроком 1 призведе до отримання вихідної карти функцій 1 на 1. Таку ж вихідну карту об'єктів можна отримати, реалізувавши два згорткові шари 3 на 3 з кроком 1.

Тепер давайте подивимося на кількість параметрів, які потрібно навчити. Для фільтра згорткового шару 5 на 5 кількість змінних дорівнює 25. З іншого боку, два згорткових шари з розміром ядра 3 на 3 мають всього 3 на 3 і на 2 = 18 змінних (менше на 28%).

Так само ефект одного згорткового шару 7 на 7 (11 на 11) може бути досягнутий шляхом реалізації трьох (п'яти) згорткових шарів 3 на 3 з кроком в один. Це знижує кількість параметрів, що навчаються, на 44,9% (62,8%). Зменшення кількості параметрів, означає більш швидке навчання і більшу стійкість до перенавчання.

2.1.3. Аналіз характеристик та принципу будови моделі ResNet

Існує кілька версій архітектури ResNetXX, де «XX» позначає кількість шарів. Найчастіше використовуються ResNet50 та ResNet101. Так як проблема зникаючого градієнта була вирішена, CNN почали ставати дедалі глибшими. Нижче наведено структурні деталі ResNet18 (рис. 2.3).

ResNet18 - Structural Details														
#	Input Image			output			Layer	Stride	Pad	Kernel		in	out	Param
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
Total														11,511,784

Рисунок 2.3 – Деталі будови ResNet18

Resnet18 має близько 11 мільйонів навчальних параметрів. Він складається з шарів CONV із фільтрами розміром 3 на 3 (як і VGGNet). У мережі використовуються лише два рівні об'єднання: один на початку, а інший наприкінці мережі. Ідентифікаційні з'єднання знаходяться між кожними двома рівнями CONV. Суцільні стрілки показують ярлики ідентичності, де розміри входу та виходу однакові, а пунктирні стрілки являють собою з'єднання проекцій, де розміри різняться.

Архітектура ResNet використовує швидкі з'єднання для вирішення проблеми градієнта, що зникає. Базовим архітектурним блоком ResNet є залишковий блок, який повторюється у всій мережі (рис. 2.4).

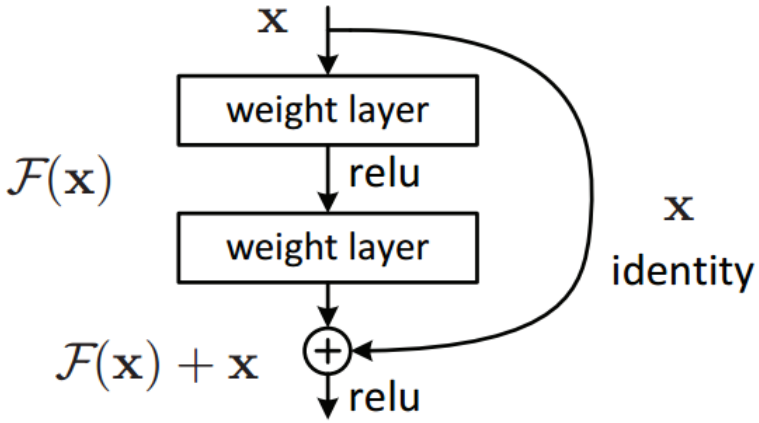

Рисунок 2.4 – Принцип роботи ResNet

Замість того, щоб вивчати відображення з x → F(x), мережа вивчає відображення з x → F(x) + G(x). Коли розмір входу x і виходу F(x) однакова, функція G(x) = x є функцією ідентичності, а з'єднання швидкого доступу називається з'єднанням ідентичності. Ідентичне відображення вивчається шляхом обнулення ваги в проміжному шарі під час навчання, оскільки легше обнулити ваги, ніж підштовхнути їх до одиниці.

Отже, можна провести декілька порівнянь:

AlexNet і ResNet мають близько 60 мільйонів параметрів, але різниця в їхній точності становить близько 10%. Але для навчання ResNet потрібно багато обчислень (приблизно в 10 разів більше, ніж для AlexNet), що означає, що потрібно більше часу та енергії для навчання.

VGGNet не тільки має більше параметрів і FLOP в порівнянні з ResNet, але також має меншу точність. Навчання VGGNet із меншою точністю займає більше часу. Тому, в основі майбутньої моделі буде лежати архітектура ResNet.

2.2. Вибір технології реалізації моделі та середовища

Машинне навчання – складна дисципліна, але, на щастя, є інструменти, що спрощують її реалізацію. Одним із таких ресурсів є фреймворк Google TensorFlow[7].

Це бібліотека з відкритим кодом для чисельних обчислень та машинного навчання, сумісна з мовою Python. Це спрощує процес збору даних, навчання моделей машинного навчання, створення прогнозів та покращення майбутніх результатів.

TensorFlow об'єднує моделі та алгоритми машинного навчання та глибокого навчання. Мова Python надає зручний та комфортний API для створення програм з використанням цієї платформи. Потім ці програми можна запускати на C++.

Цей фреймворк можна використовувати для навчання і запуску глибоких нейронних мереж для класифікації рукописних цифр, розпізнавання зображень, для рекурентних нейронних мереж та для обробки природної мови.

2.2.1. Опис фреймоврка Tensorflow

З самого початку TensorFlow було створено командою Google Brain. Початковою метою було використання нейронних мереж для покращення сервісів Google, таких як Gmail, Фото та пошукової системи.

Завдяки цьому фреймворку дослідники та розробники можуть разом працювати над моделлю штучного інтелекту. Згодом TensorFlow був вперше випущений для широкої публіки наприкінці 2015 року. Проте перша стабільна версія датується 2017 роком [8].

Цей інструмент з вихідним кодом розповсюджується під ліцензією Apache Open Source [9]. Таким чином можна використовувати, змінювати та

розповсюджувати змінену версію в комерційній формі без необхідності платити Google.

Розробники можуть створювати “графи потоків даних”. Це структури для опису того, як дані передаються через графіку чи серію вузлів обробки.

Кожен вузол на графіці є математичною операцією. Кожне з'єднання між вузлами є багатовимірним масивом даних: тензор (рис. 2.5).

Взаємодія з TensorFlow здійснюється через мову Python, яку легко вивчити та використовувати. Ця мова дозволяє легко описати, як абстракції високого рівня можуть бути пов'язані разом.

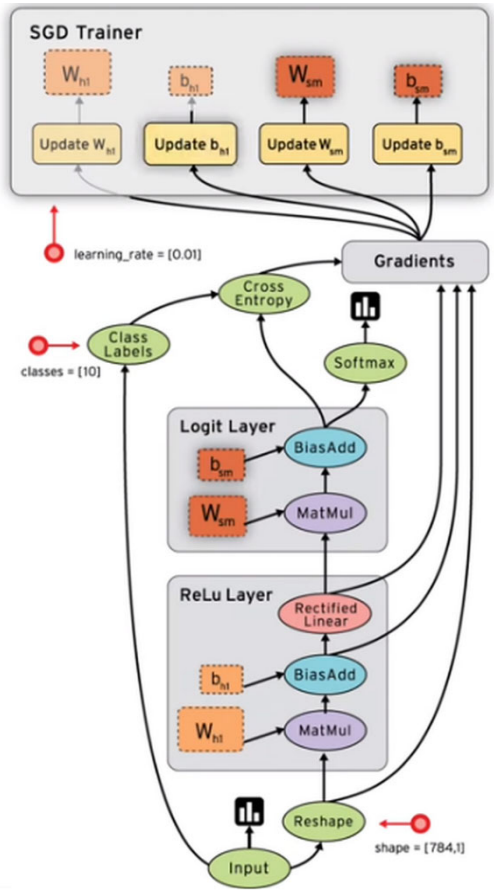

Рисунок 2.5 – Граф потоку даних в Tensorflow

Вузли та тензори TensorFlow є об'єктами Python. Самі програми TensorFlow є програмами Python. Проте самі математичні операції у Python не виконуються. Бібліотеки перетворень доступні у TensorFlow, написані як високопродуктивні

двійкові файли C++. Мова Python просто спрямовує трафік між різними частинами та дозволяє з'єднувати їх один з одним за допомогою абстракцій високого рівня.

Програми TensorFlow можна запускати на локальному комп'ютері, у хмарному кластері, на смартфонах та планшетах iOS або Android, або навіть на центральних та графічних процесорах. Власна хмара Google [10] пропонує запускати TensorFlow на своїх чіпах TensorFlow Processing Unit (TPU) для підвищення прискорення. Моделі, створені TensorFlow, можна розгорнути на будь-якому пристрої для отримання прогнозів.

З виходом TensorFlow 2.0, запущеного у жовтні 2019 року [11], фреймворк зазнав капітального ремонту. Внесені зміни ґрунтуються на відгуках користувачів. Ця нова версія більш ефективна та проста у використанні, зокрема завдяки використанню Keras API для навчання моделей. Новий API полегшує розподілене навчання.

Сумісність із TensorFlow Lite дозволяє розгортати моделі на ширшому спектрі платформ. Єдиний недолік: код, написаний для попередніх версій TensorFlow, необхідно переписати, щоб повною мірою використати нові можливості TensorFlow 2.0.

TensorFlow надає велику кількість переваг для розробки машинного навчання. Один з основних – абстракція. Замість того, щоб зупинятися на деталях реалізації алгоритмів або зв'язку між функціями, розробники можуть зосередитись на загальній логіці програми. Фреймворк дбає про технічні деталі. Також спрощується налагодження та перевірка програм. Наприклад, режим «активного виконання» дозволяє оцінювати та змінювати кожну діаграму операції окремо. Таким чином, вони не є єдиним непрозорим об'єктом, який можна повністю оцінити. Пакет візуалізації TensorBoard, з іншого боку, дозволяє перевіряти, як виконуються діаграми, через інтерактивну веб-панель управління.

Крім того, факт, що за цим проектом стоїть Google, є великою перевагою. Це дозволило прискорити розробку, а також створити безліч пропозицій навколо TensorFlow, спростивши його розгортання та використання.

2.3. Методи навчання нейронних мереж

Характерною рисою нейронних мереж є їхня здатність до навчання (наприклад, розпізнавати зображення, букви, звук…). Але ці знання набуваються не від початку створення мережі.

Більшість нейронних мереж навчаються на прикладах, дотримуючись алгоритму навчання. Під час навчання динаміка мережі змінюється за допомогою ваг відповідно до набору даних, представлених як вхідні, доки не буде отримано бажану поведінку. Навчання може бути контрольованим чи не контрольованим, залежно від наявності або відсутності бажаної відповіді.

2.3.1. Контрольоване навчання нейронної мережі

Для цього типу навчання керівник (вчитель) надає мережі вхідні дані, та відповідні бажані виходи. Мережа повинна коригувати свої ваги, щоб зменшити різницю між бажаним результатом та отриманим результатом, ця процедура повторюється до тих пір, поки критерій продуктивності не буде задовільним.

Алгоритм навчання, що використовується в даній роботі, ґрунтується на зворотному розповсюдженні помилки, тому що останній найкраще підходить для навчання нейронних мереж типу CNN (рис. 2.6).

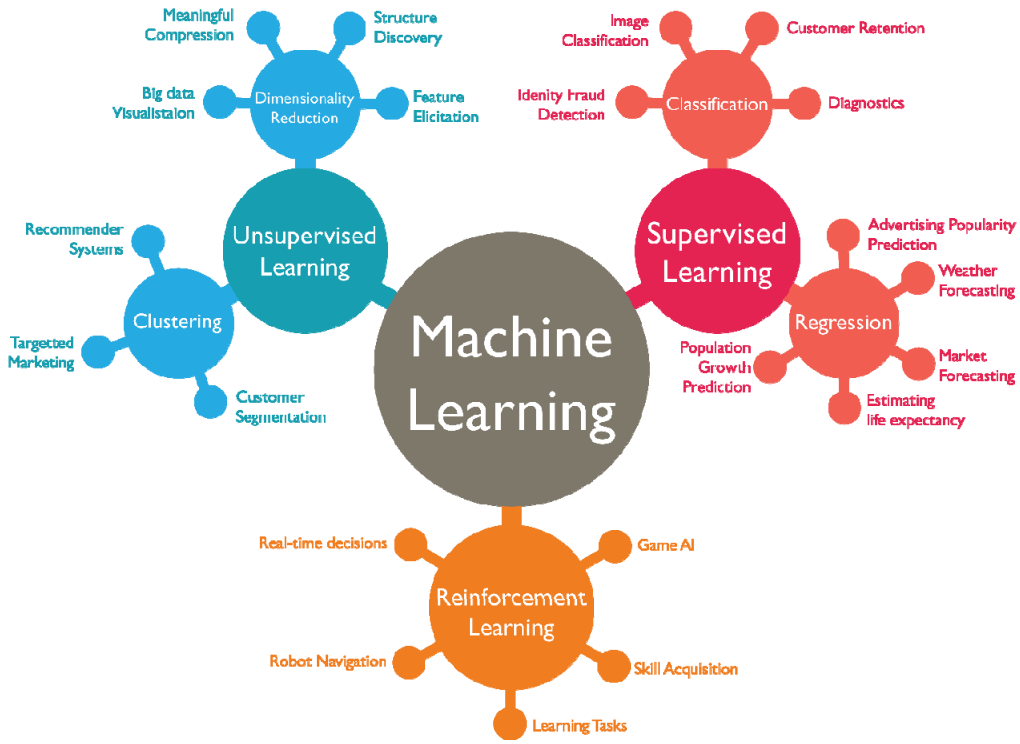

Рисунок 2.6 – Алгоритми навчання нейронних мереж

При інтелектуальному аналізі даних контрольоване навчання можна розділити на два типи завдань – класифікацію та регресію:

- Класифікація використовує алгоритм для точного розподілу тестових даних за певними категоріями. Він розпізнає певні об'єкти в наборі даних і намагається зробити деякі висновки про те, як ці об'єкти мають бути позначені чи визначені. Поширеними алгоритмами класифікації є лінійні класифікатори, методи опорних векторів (SVM), дерева рішень, найближчий сусід і випадковий ліс.
- Регресія використовується для розуміння взаємозв'язку між залежними та незалежними змінними. Цей алгоритм зазвичай використовується для складання прогнозів, наприклад, для доходів від продажу даного бізнесу. Лінійна регресія, логістична регресія та поліноміальна регресія – популярні алгоритми регресії.

2.3.2. Не контрольоване навчання нейронної мережі

При навчанні без учителя жодна інформація про бажаний вихід не надається мережа. Ми представляємо вхід до мережі та дозволяємо їй вільно розвиватися, доки вона не стабілізується, така поведінка відома як "самоорганізація". Нейронна мережа у цьому випадку автоматично виявляє закономірності, які з'являються у наведених прикладах, та змінює ваги сполук так, щоб приклади, що мають однакові характеристики викликали той самий результат.

Моделі навчання без учителя використовуються для вирішення таких завдань – кластеризація та зменшення розмірності. Нижче ми визначимо кожен метод навчання та виділимо загальні алгоритми та підходи для їх ефективного проведення.

- Кластеризація – це метод інтелектуального аналізу даних, який групує немарковані дані на основі їхньої подібності або відмінностей. Алгоритми кластеризації використовуються для обробки необроблених некласифікованих об'єктів даних групи, представлені структурами або шаблонами в інформації. Алгоритми кластеризації можна розділити на кілька типів, зокрема ексклюзивні, ієрархічні, що перекриваються, і ймовірнісні.
- Зменшення розмірності – це метод, який використовується, коли кількість об'єктів або розмірів у цьому наборі даних занадто велика. Це скорочує кількість даних, що вводяться до керованого розміру, при цьому зберігаючи цілісність набору даних в максимально можливому ступені. Він зазвичай використовується на етапі попередньої обробки даних.

Основна відмінність між двома підходами – використання помічених наборів даних. Простіше кажучи, контрольоване навчання використовує позначені вхідні та вихідні дані, а алгоритм неконтрольованого навчання – ні.

При навчанні з учителем алгоритм “навчається” на навчальному наборі даних, ітеративно роблячи прогнози на основі даних та коригуючи їх для отримання правильної відповіді. Хоча моделі навчання з учителем зазвичай більш точні, ніж моделі навчання без вчителя, вони вимагають попереднього втручання людини для належного маркування даних. Наприклад, модель навчання з учителем може передбачити, як довго продовжуватиметься ваша поїздка на роботу, залежно від часу доби, погодних умов тощо. Але спочатку вам потрібно навчити його знати, що дощова погода збільшує час керування.

Моделі навчання без вчителя, навпаки, працюють самі собою, щоб виявити внутрішню структуру немаркованих даних. Зверніть увагу, що вони, як і раніше, вимагають деякого втручання людини для перевірки вихідних змінних. Наприклад, модель навчання без вчителя може визначити, що онлайн-покупці часто купують групи продуктів одночасно. Проте аналітику даних необхідно буде підтвердити, що для механізму рекомендацій має сенс згрупувати дитячий одяг по порядку підгузків, яблучного пюре та стаканчиків.

Інші ключові відмінності між навчанням з учителем та без:

- Мета: при навчанні з учителем ціль полягає в тому, щоб передбачити результати для нових даних. Ви наперед знаєте, яких результатів очікувати. При використанні алгоритму навчання без учителя ціль полягає в тому, щоб отримати знання від великих обсягів нових даних. Саме машинне навчання визначає, що відрізняється чи цікаве у наборі даних.
- Додатки: моделі контрольованого навчання ідеально підходять для виявлення спаму, аналізу настрою, прогнозів погоди та ціноутворення. І навпаки, навчання без вчителя відмінно підходить для виявлення аномалій, механізмів рекомендацій, персоналізації клієнтів та медичної візуалізації.
- Складність: контрольоване навчання – це простий метод машинного навчання, який зазвичай розраховується за допомогою таких програм, як R або Python. При навчанні без вчителя потрібні потужні інструменти

для роботи з великими обсягами некласифікованих даних. Моделі навчання без вчителя складні у обчислювальному відношенні, тому що їм потрібен великий навчальний набір для отримання бажаних результатів.

- Недоліки: навчання моделей контрольованого навчання може зайняти багато часу, а мітки для вхідних та вихідних змінних вимагають спеціальних знань. Тим часом методи навчання без вчителя можуть давати вкрай неточні результати, якщо у не буде втручання людини для перевірки вихідних змінних.

2.4. Поняття функція активації

Функція активації переважно використовується для нелінійної зміни даних. Ця нелінійність дозволяє просторово змінювати їхнє уявлення. Простіше кажучи, функція активації дозволяє змінити спосіб перегляду даних.

Не слід плутати функцію активації з функцією втрат. Функція втрат застосовується до всієї моделі і тому є унікальною, вона дозволяє розрахувати продуктивність моделі. Навпаки, функція активації специфічна кожного шару, вона дозволяє перетворювати дані.

Особливість цієї функції активації у цьому, що вона нелінійна. Ця нелінійність дозволяє змінити уявлення про дані, по-новому подивитись на ці дані. Така зміна уявлення неможлива при лінійному перетворенні.

Кожен нейрон у шарі застосовує функцію активації до даних. Це перетворення відрізнятиметься для кожного нейрона, тому що кожен з них має різну вагу.

2.4.1. Функція активації ReLU

Функція випрямлених лінійних одиниць (рис. 2.7) – найпростіша функція активації, що найбільш широко використовується. Вона вертає x, якщо x більше 0, інакше – 0. Іншими словами, це максимум між x і 0 (1):

$$F(x) = \max(x, 0) \tag{1}$$

де x – вхідні дані.

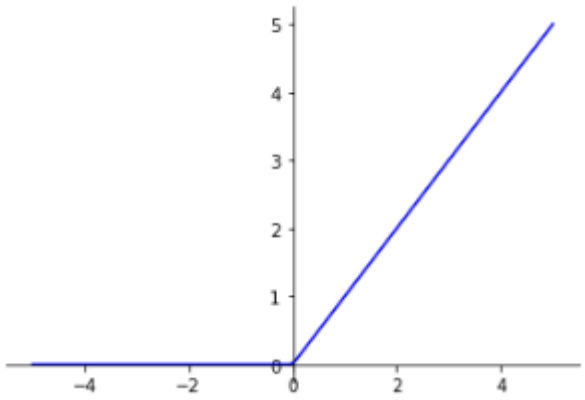

Рисунок 2.7 – Функція активації ReLU

Ця функція дозволяє фільтрувати дані. Вона дозволяє позитивним значенням (x > 0) проходити через наступні рівні нейронної мережі. Вона використовується практично скрізь, але не як останній шар, вона використовується тільки у проміжних шарах.

2.4.2. Функція активації Sigmoid

Сигмоїдна функція (рис. 2.8) повертає значення від 0 до 1, тобто ймовірність. Тому вона широко використовується для двійкової класифікації, коли модель має визначати лише дві мітки.

Таким чином, для класифікації оглядів кіно, чим ближче значення, яке повертає Sigmoid до 1, тим більше модель вважає відгук позитивним. І навпаки, чим більше повернене значення ближче до 0, то відгук вважається негативним (2).

$$F(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \tag{2}$$

де x – вхідні дані.

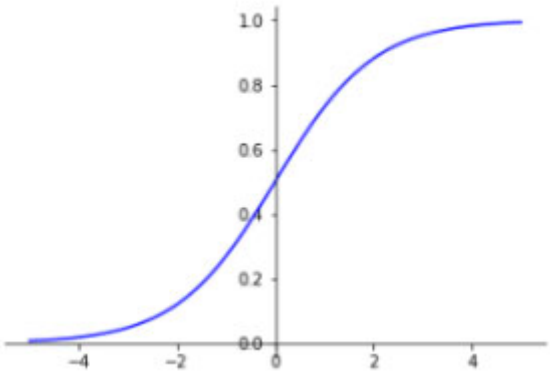

Рисунок 2.8 – Функція активації Sigmoid

2.4.3. Функція активації Softmax

Функція Softmax (рис. 2.9) дозволяє перетворювати дійсний вектор на вектор ймовірності. Вона часто використовується на останньому рівні моделі класифікації, особливо для мультикласових завдань.

У функції Softmax кожен вектор обробляється незалежно. Аргумент осі визначає вхідну вісь, до якої застосовується функція (3).

$$F(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{3}$$

де x – вхідні дані;

n – кількість векторів вхідних даних.

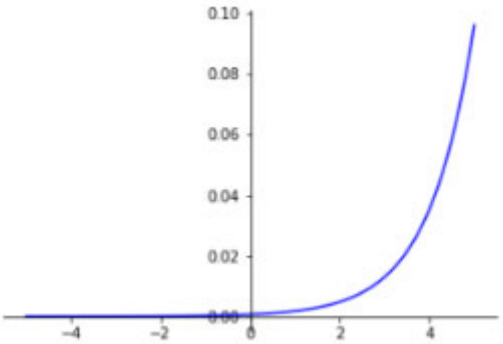

Рисунок 2.9 – Функція активації Softmax

2.5. Поняття перенавчання та недонавчання моделі

Нейронна мережа навчається на представлених прикладах (навчальних іграх). Мета цього навчання – дозволити мережі отримати загальні відомості з цих прикладів і мати можливість згодом застосовувати їх до нових даних.

Навчання глибокої нейронної мережі, яка зможе точно працювати з новими даними, є складним завданням.

Модель із занадто малою ємністю не може вивчити проблему, тоді як модель із занадто великою ємністю може вивчити її надто добре та перевершити навчальний набір даних. В обох випадках виходить модель, яка погано узагальнюється (рис. 2.10).

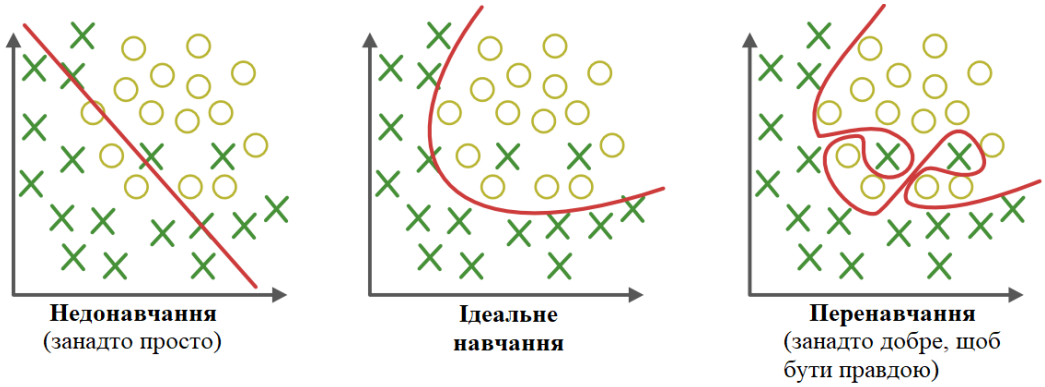

Рисунок 2.10 – Ілюстрація недонавчання та перенавчання

2.5.1. Методи боротьби з недонавчанням

Для боротьби з недонавчанням найкраща стратегія – збільшення складності моделі, збільшивши кількість параметрів у моделі глибокого навчання, або змінивши порядок моделі. Недонавчання пов'язано з тим, що модель простіша, ніж потрібно. Вона не може вловити закономірності у даних. Збільшення складності моделі підвищить ефективність навчання. Також, варто спробувати натренувати модель, збільшивши кількість епох. Переконавшись, що під час тренування втрати поступово зменшуються. В іншому випадку дуже ймовірно, що є якась помилка або збій у самій логіці коду.

Якщо не змішувати дані після кожної епохи, це також може негативно вплинути на продуктивність моделі.

2.5.2. Методи боротьби з перенавчанням

На відміну від недонавчання, існує кілька методів боротьби з перенавчанням, які можна спробувати використати.

Для початку, потрібно збільшити кількість навчальних даних і хоча получити більше даних не завжди можливо, отримання більш репрезентативних даних є надзвичайно корисним. Наявність великого та різноманітного набору даних зазвичай може допомогти підвищити продуктивність моделі. Це допоможе отримати кращу модель, яке буде здатна краще узагальнювати. Це означає, що продуктивність моделі на невидимих даних (тестовий набір даних) буде кращою.

У випадку, коли неможливо отримати більше тренувальних даних, можна спробувати збільшити варіативність даних за допомогою аугментації (англійське слово – augmentation) [12]. Збільшення варіативності означає штучну зміну існуючих даних за допомогою перетворень, що нагадують варіації реальних даних. Найпоширенішими видами аугментації в практичних завданнях є:

1) поворот зображення;
2) зміщення по ширині;
3) зміщення по висоті;
4) зміна яскравості зображення;
5) розтягування зображення;
6) масштабування.

Ще одним методом боротьби з перенавчанням, є рання зупинка [13]: Рання зупинка – це форма регуляризації, що дозволяє уникнути надмірного навчання під час навчання моделі з допомогою ітеративного методу, як-от градієнтний спуск. Під час навчання нейронних мереж ми ітеративно використовуємо градієнти навчальних даних, і намагаємося зробити модель найбільш підходящою для виконання основних функцій реального світу. У певному сенсі, цей метод дозволяє зупинитися в точці найкращої відповідності або біля неї. Таким чином запобігається перенасичення навчальної вибірки та зменшується помилка

узагальнення. Щоб вирішити, коли зупинитися, ми можемо відслідковувати деякі показники, такі як втрати, тестова та валідаційна точності, і залежно від певних умов можна припинити навчання.

Випадання: основна ідея цього методу полягає у випадковому відкиданні одиниць із нейронних мереж під час навчання. Він був представлений у статті: “Dropout: A Simple Way to Prevent Neural Networks from Overfitting (2014)” [14]. Під час навчання випадкові відсіювальні вибірки утворюють велику кількість різних “тонких” мереж. Відмова досягається шляхом побудови матриці шляхом вилучення з розподілу Бернуллі з ймовірністю p (щоб отримати 1) (тобто. ймовірність відмови становить 1 – p), потім виконується множення на елементи з виходами прихованих шарів. На наступному малюнку показано відкидання під час фази навчання (рис. 2.11):

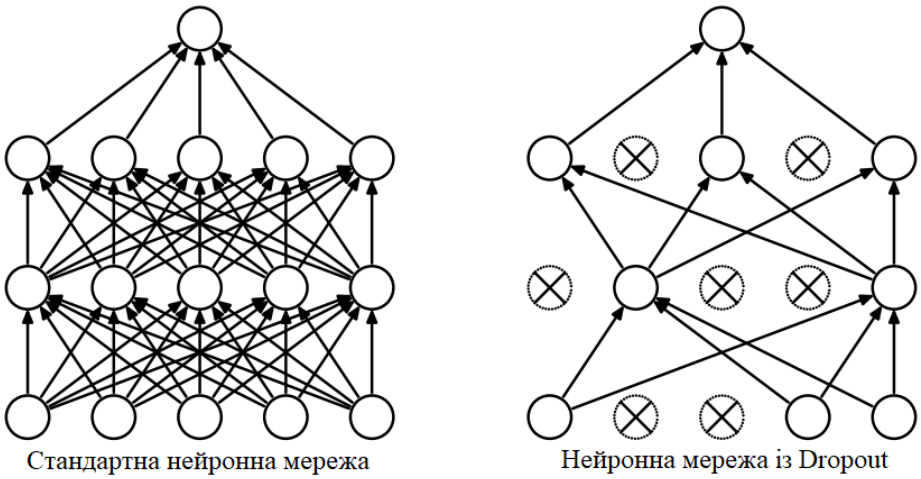

Рисунок 2.11 – Приклад мережі із застосованим методом випадання

Випадання гарантує, що жоден нейрон не виявиться занадто залежним від інших нейронів і натомість дізнається про щось значуще. Випадання може застосовуватися після згорткових, поєднуючих або повнозв'язних шарів.

2.6. Висновки до розділу

У цьому розділі ми побачили, що для навчання нейронної мережі потрібен навчальний набір та алгоритм оптимізації. Ознайомилися із основними функціями активації ReLU, Sigmoid та Softmax. Також, ознайомилися з поняттями перенавчання та недонавчання та методами боротьби із ними.

3. ПРОЕКТУВАННЯ ТА РОЗРОБКА МОДЕЛІ

3.1. Вибір архітектури та платформи для розробки моделі

Для того, щоб реалізувати поставлену задачу було обрано згорткову нейронну мережу, її розробка складається з таких частин:

1) вибір фреймворку для реалізації моделі;
2) збір тренувального набору даних;
3) планування структури моделі та її об’єму;
4) початок тренування моделі;
5) тестування моделі.

Для реалізації спроектованої системи було обрано наступні технології:

1) дистрибутив Anaconda 3;
2) IDE Jupyter Notebook;
3) мова програмування Python;
4) фреймворк TensorFlow

3.1.1. Дистрибутив Anaconda 3

Anaconda – це дистрибутив Python. Дистрибутив Anaconda Python містить диспетчер пакетів conda, який можна використовувати для встановлення пакетів Python conda або conda-forge. Anaconda також включає базове середовище, яке містить основну мову програмування Python і його вбудовані модулі, інтегровані середовища розробки (IDE) Spyder і Jupyter Notebook, а також найбільш поширені бібліотеки або модулі для наукових даних, такі як numpy, scipy, pandas, matplotlib та seaborn. а також багато інших. IDE можна розглядати як програми для роботи з

кодом Python, аналогічно до того, як можна використовувати Libre Office або Microsoft Word для роботи з документом.

Miniconda має той же менеджер пакетів conda, що і Anaconda, але замість універсального базового середовища, упакованого разом з бібліотеками даних, вона, по суті, має порожнє базове середовище. Хоча можна змінити базове середовище, рекомендується створити власне середовище conda, а не змінювати базове середовище в більшості випадків.

І в Miniconda, і в Anaconda середа conda може бути створена з нуля. Теоретично нові користувачі можуть просто встановити Anaconda і використовувати базове середовище conda.

3.1.2. Мова програмування Python

Python – найбільш широко використовувана мова програмування з відкритим кодом серед комп'ютерних наук. Ця мова програмування вийшла на передній план у галузі управління інфраструктурою, аналізу даних та розробки програмного забезпечення. Однією з хороших якостей Python, зокрема є те, що він дозволяє розробникам зосередитись на тому, що вони роблять, а не на тому, як вони це роблять. Це звільнило розробників від обмежень форми, які займали їхній час зі старими мовами. Таким чином, розробка коду на Python виконується швидше, ніж з допомогою інших мов.

Ця мова досить доступна для новачків, багато підручників також доступні для вивчення на спеціалізованих веб-сайтах. На комп'ютерних форумах завжди можна знайти відповіді на свої запитання, тому що ним користуються багато професіоналів.

3.1.3. Фреймворк TensorFlow та його переваги

TensorFlow – це бібліотека з відкритим вихідним кодом для запуску програм машинного навчання та глибокого навчання. Цей інструмент, призначений для створення нейронних мереж, був розроблений Google і широко використовується в галузі штучного інтелекту. Таким чином, як професіонали, так і новачки можуть створювати моделі машинного або глибокого навчання для оптимізації можливостей свого обладнання.

Для того, щоб встановити TensorFlow потрібно скористатися менеджером пакетами “pip”. Після перевірки сумісності відеокарти (карти NVIDIA) [15] можна встановити Tensorflow GPU. TensorFlow сумісний з наступними 64-розрядними системами: Python 3.5-3.8. Ubuntu 16.04 або більше. Windows 7 чи новіший.

PyTorch – це бібліотека машинного навчання, розроблена Facebook. TensorFlow замінює PyTorch з точки зору даних та пропонованих інструментів. PyTorch краще підходить для фундаментальних досліджень, тоді як TensorFlow багатший і ефективніший, але менш простий у використанні. Крім того, PyTorch розвивається, щоб досягти якостей TensorFlow, поєднуючи продуктивність та простоту використання.

Keras – це бібліотека з відкритим кодом, заснована на роботі розробника Google Франсуа Шолле. Він є дуже ефективним додатковим інструментом в додаток до TensorFlow. Keras не є прямим конкурентом, оскільки заснований на трьох інструментах: TensorFlow, Theano та Microsoft Cognitive Toolkit. Таким чином, можна легко перемикатися з одного інтерфейсу на інший.

3.2. Збір набору даних і тренування моделі

Для реалізації моделі знадобиться великий набір даних, а саме зображення з людьми, тваринами, фруктами та рослинами. Існують безліч готових наборів даних, наприклад, ImageNet, COCO, Cifar-100 та Open Images v6.

- ImageNet – це велика візуальна база даних, призначена для використання під час дослідження програмного забезпечення для розпізнавання візуальних об'єктів. Більше 14 мільйонів зображень були вручну анотовані в рамках проекту, щоб вказати, які об'єкти представлені. ImageNet містить понад 20 000 категорій, серед яких типова категорія, така як “повітряна куля” або “полуниця”, складається з кількох сотень зображень.
- COCO – це великомасштабний набір даних для класифікації, сегментації, виявлення ключових точок та субтитрів. Набір даних складається із 328 тисяч зображень.
- CIFAR-100 – це набір даних з анотаціями, який містить у собі 80 мільйонів крихітних зображень. Набір даних складається з 60 000 кольорових зображень (50 000 для навчання та 10 000 для тестування) розміром 32×32 пікселів у 100 класах, згрупованих у 20 суперкласів. У кожного зображення є точна мітка (клас) та груба мітка (суперклас).
- Open Images – це найбільший набір даних анотованих зображень у багатьох відношеннях, призначений для використання при навчанні нових глибоких згорткових нейронних мереж для завдань комп'ютерного зору. Open Images включає 9 мільйонів зображень, анотованих 36 мільйонами міток зображень, 15.8 мільйони обмежуючих рамок, 2.8 мільйона екземплярів із сегментацією і 391 тисяч візуальних взаємозв'язків. Його я і використаю для проведення тренувань моделі для класифікації та ідентифікації біооб’єктів.

Для завантаження набору, я скористався бібліотекою FiftyOne [16], попередньо встановивши її за допомогою менеджера пакетами pip (рис. 3.1).

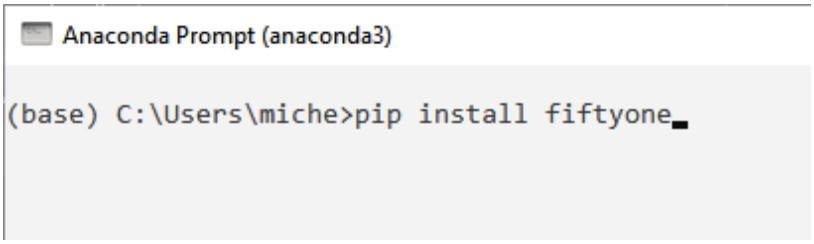

Рисунок 3.1 – Встановлення бібліотеки FiftyOne

У моєму випадку, модель повинна класифікувати та ідентифікувати біооб’єкти на зображенні, тому я завантажив набір даних із типом анотацій “detections”. Для створення збалансованого набору даних, рекомендована кількість зразків на клас повинна становити 1000 зображень [17]. Тому, мною було створений набір даних, що складається із 64 класу біооб’єктів (люди, тварини, рослини тощо), кожен з яких містить у середньому 900 картинок із обмежувальними рамками та мітками (рис. 3.2).

Рисунок 3.2 – Зразок фото із анотаціями

Набір був збережений у форматі Pascal VOC, тобто у кожного зображення є відповідний йому файл у форматі XML, що містить наступну інформацію:

1. ширину та висоту зображення;
2. глибину зображення (в нашому випадку, цей параметр рівний 3, оскільки зображення кольорові);
3. клас відповідний об’єкту на зображені;
4. мітки: “об’єкт всередині”, “об’єкт закритий” та інші;
5. координати обмежувальної рамки.

Далі було створено файл-карта з мітками усіх класів (рис. 3.3), який буде потрібним при проведенні серіалізації даних.

```
item {id: 1 name: 'Person'}
item {id: 5 name: 'Camel'}
item {id: 9 name: 'Starfish'}
item {id: 13 name: 'Crab'}
item {id: 17 name: 'Cat'}
item {id: 21 name: 'Dolphin'}
item {id: 25 name: 'Giraffe'}
item {id: 29 name: 'Fish'}
item {id: 33 name: 'Goat'}
item {id: 37 name: 'Lemon'}
item {id: 41 name: 'Monkey'}
item {id: 45 name: 'Mushroom'}
item {id: 49 name: 'Penguin'}
item {id: 53 name: 'Apple'}
item {id: 57 name: 'Carrot'}
item {id: 61 name: 'Rhinoceros'}
item {id: 2 name: 'Bread'}
item {id: 6 name: 'Watermelon'}
item {id: 10 name: 'Tiger'}
item {id: 14 name: 'Crocodile'}
item {id: 18 name: 'Dog'}
item {id: 22 name: 'Elephant'}
item {id: 26 name: 'Duck'}
item {id: 30 name: 'Flower'}
item {id: 34 name: 'Sparrow'}
item {id: 38 name: 'Leopard'}
item {id: 42 name: 'Moths and butterflies'}
item {id: 46 name: 'Owl'}
item {id: 50 name: 'Pig'}
item {id: 54 name: 'Polar bear'}
item {id: 58 name: 'Rabbit'}
item {id: 62 name: 'Sea lion'}
item {id: 3 name: 'Bull'}
item {id: 7 name: 'Strawberry'}
item {id: 11 name: 'Tomato'}
item {id: 15 name: 'Cucumber'}
item {id: 19 name: 'Horse'}
item {id: 23 name: 'Brown bear'}
item {id: 27 name: 'Eagle'}
item {id: 31 name: 'Fox'}
item {id: 35 name: 'Grape'}
item {id: 39 name: 'Lion'}
item {id: 43 name: 'Mouse'}
item {id: 47 name: 'Panda'}
item {id: 51 name: 'Shark'}
item {id: 55 name: 'Orange'}
item {id: 59 name: 'Raccoon'}
item {id: 4 name: 'Cabbage'}
item {id: 8 name: 'Chicken'}
item {id: 12 name: 'Turtle'}
item {id: 16 name: 'Bird'}
item {id: 20 name: 'Squirrel'}
item {id: 24 name: 'Zebra'}
item {id: 28 name: 'Spider'}
item {id: 32 name: 'Frog'}
item {id: 36 name: 'Kangaroo'}
item {id: 40 name: 'Lizard'}
item {id: 44 name: 'Snake'}
item {id: 48 name: 'Parrot'}
item {id: 52 name: 'Banana'}
item {id: 56 name: 'Pumpkin'}
item {id: 60 name: 'Reptile'}
```

Рисунок 3.3 – Файл-карта з мітками класів

Тепер, після того як ми створили наші анотації та розділили наш набір даних на потрібні підмножини навчання та тестування, настав час перетворити наші анотації у так званий формат TFRecord. Для цього я скористався скриптом “generate_tfrecord.py”, отримані файли я помістив у папку “annotations”.

Для досягнення цілей цієї роботи я не буду тренувати модель з нуля, а використаю одну з попередньо навчених моделей, наданих TensorFlow.

Модель, яку я буду використовувати у своїй роботі, – це модель SSD ResNet50 v1 FPN 640x640, оскільки вона забезпечує відносно хорошу продуктивність та швидкість.

Після завантаження моделі, нам потрібно налаштувати файл “pipeline.config”, що відповідає за настройки тренування, а саме, нас цікавлять наступні параметри:

1. “num_classes” – параметр, що відповідає за кількість класів, в моєму випадку цей параметр рівний 62;
2. “batch_size” – відповідає за швидкість навчання (залежить від потужності відеокарти), мій ноутбук зміг стабільно працювати при значенні 3;
3. “fine_tune_checkpoint_type” – тип моделі яку ми перенавчаємо, у моєму випадку це “detection”;
4. “label_map_path” – шлях до файла-карти;
5. “input_path” – шлях до набору даних в форматі tfrecords;

Щоб запустити навчання, потрібно запустити скрипт “model_main_tf2.py”, для цього потрібно у терміналі відкрити папку зі скриптом, та запустити його (рис. 3.4).

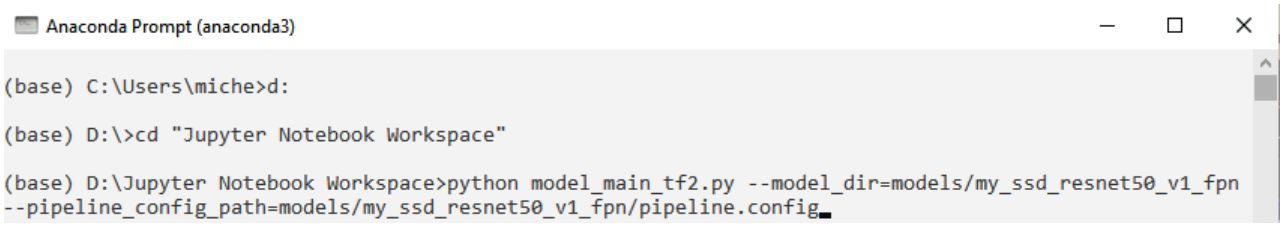

Рисунок 3.4 – Запуск тренування

Після того, як процес навчання розпочато, можна побачити серію роздруківок, подібних до наведених нижче (рис. 3.5):

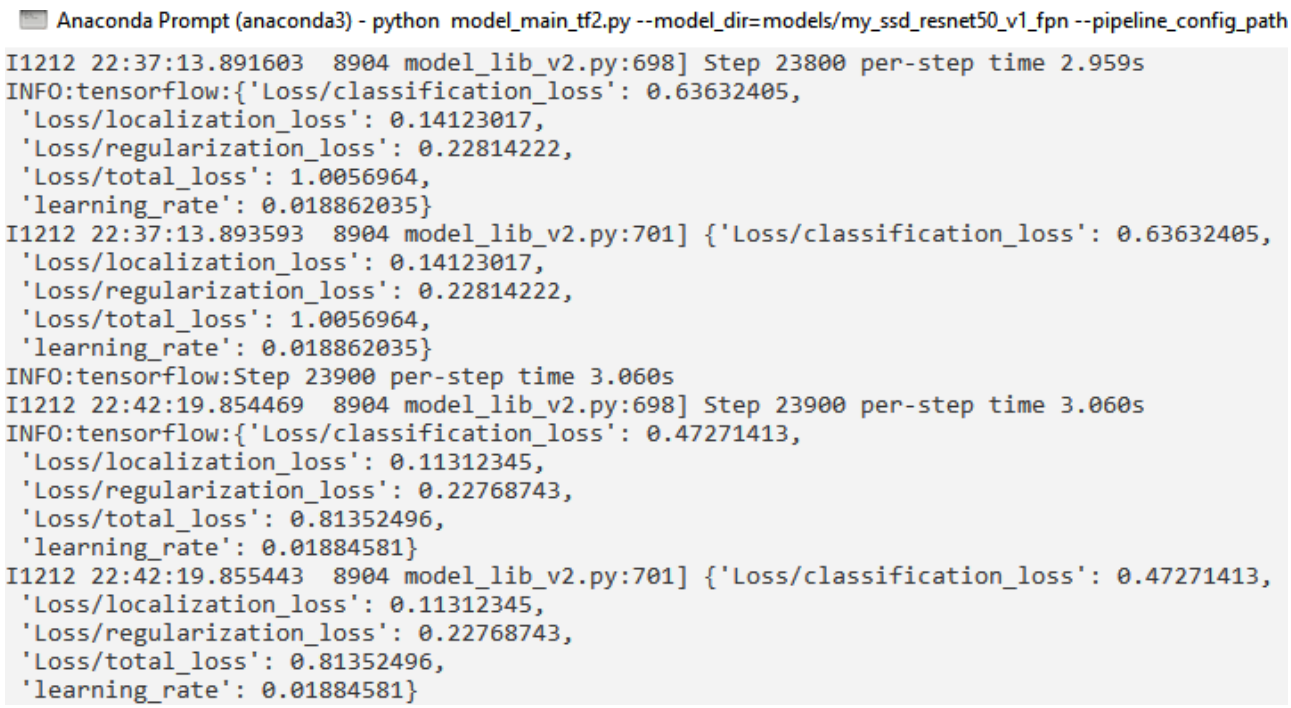

Рисунок 3.5 – Процес навчання моделі

З огляду на рекомендації, доцільно дозволити нашій моделі досягти TotalLoss щонайменше 2 (в ідеалі 1 і нижче), якщо потрібно досягти “точних” результатів виявлення. Очевидно, що нижчий TotalLoss краще, однак дуже низького TotalLoss слід уникати, оскільки модель може в кінцевому підсумку перенавчитися на даному наборі даних, що означає, що вона буде погано працювати при застосуванні до зображень за межами набору даних. Щоб відстежувати TotalLoss, а також ряд інших показників, поки ваша модель тренується, можна подивитися на Monitor Training Job Progress за допомогою TensorBoard.

3.2.1. Вебсервер TensorBoard

TensorBoard – це веб-сервер для перегляду процесу навчання моделі Tensorflow. Для запуску сервера TensorBoard, потрібно вказати папку, в якій

збережені файли журналу, після цього за допомогою команди, яку зображено у лістингу 3.1 можна запустити його.

Лістинг 3.1 – Запуск вебсервера TensorBoard

```
tensorboard --logd=/path/to/logs
```

Далі сервер вкаже URL-адресу, яку нам просто потрібно відкрити в браузері, щоб відкрити відповідну панель керування за замовчанням це “http://localhost:6006”. Коли навчання моделі починається, веб-сторінка періодично зчитує останні файли журналів, створені Tensorflow, для відображення візуального інтерфейсу.

TensorBoard дозволяє візуалізувати безліч речей, а саме візуалізувати еволюцію показників навчання залежно від часу та досліджувати графік компіляції моделі Tensorflow.

Візуальний інтерфейс виглядає так:

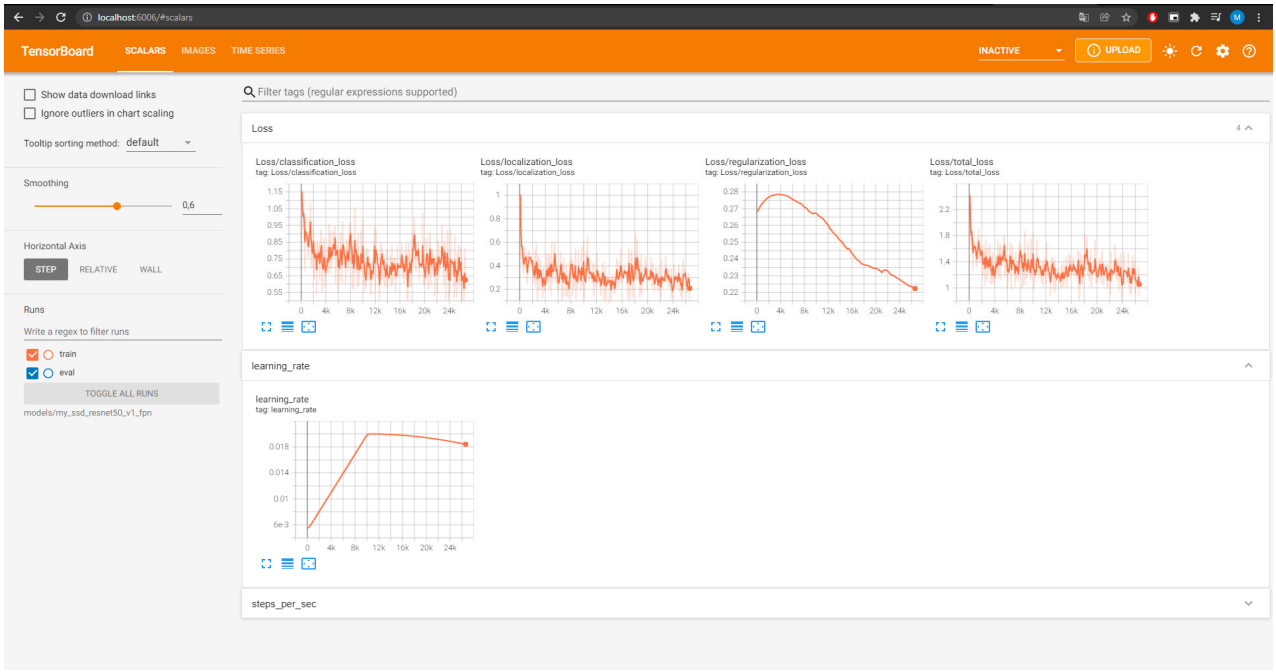

Рисунок 3.6 – Графічний інтерфейс TensorBoard

Вкладка “Images” містить інформацію про зображення, на яких вчиться модель (рис. 3.7). Малюнки здаються неправильними, це все через нормалізацію кольорів.

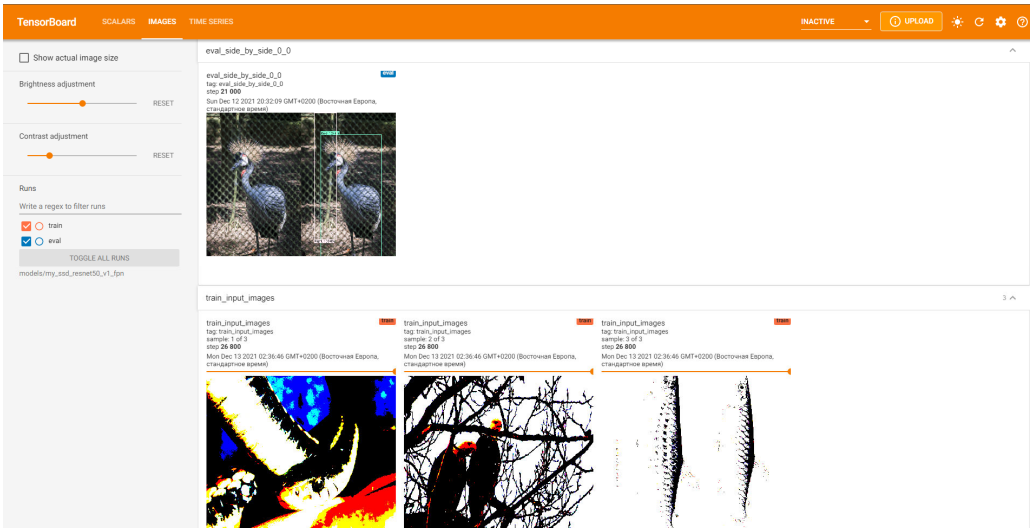

Рисунок 3.7 – Інформація на вкладці “Images”

Під час тренування можна перевірити модель за допомогою команди, яка наведена в лістингу 3.2 та переглянути попередній результат (рис. 3.8).

Лістинг 3.2 – Запуск скрипта для перевірки моделі

```
python model_main_tf2.py
```

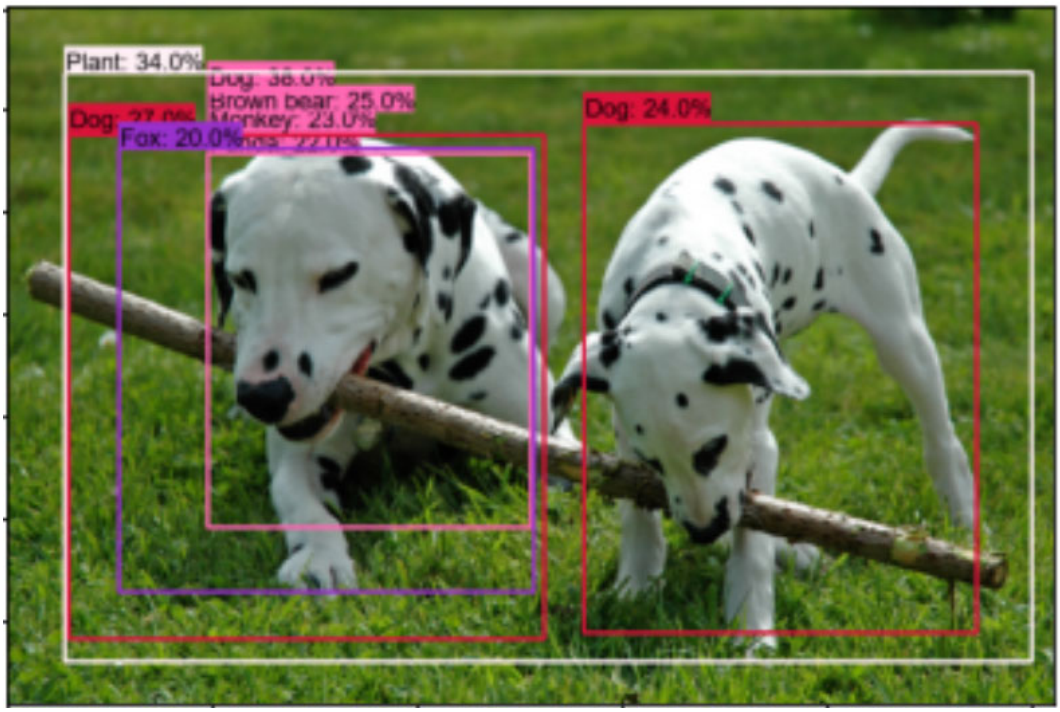

Рисунок 3.8 – Попередня перевірка роботи моделі

Як можна бачити на рисунку 3.8, модель вже добре визначає об’єкти на зображені, хоча з класифікацією цих об’єктів вона ще сумнівається. Тому варто продовжити навчання.

Після того, як навчання моделі було завершено, потрібно витягти нещодавно навчений графік висновку, який пізніше буде використаний для виявлення об’єктів. Це можна зробити наступним чином:

1) завантажити скрипт “exporter_main_v2.py”, він доступний після встановлення TensorFlow Object Detection API;

2) після цього у терміналі запустити цей скрипт, за допомогою команди, яку наведено в лістингу 3.3 та вказати необхідні параметри.

Лістинг 3.3 – Команда для експорту моделі

```
python .\exporter_main_v2.py --input_type image_tensor
--pipeline_config_path .\path\to\model\pipeline.config
--trained_checkpoint_dir .\path\to\model\
--output_directory .\exported-models\my\model
```

Як тільки скрипт завершить свою роботу, модель буде готова до перевірки на тестових даних, які вона ще не бачила (рис. 3.9).

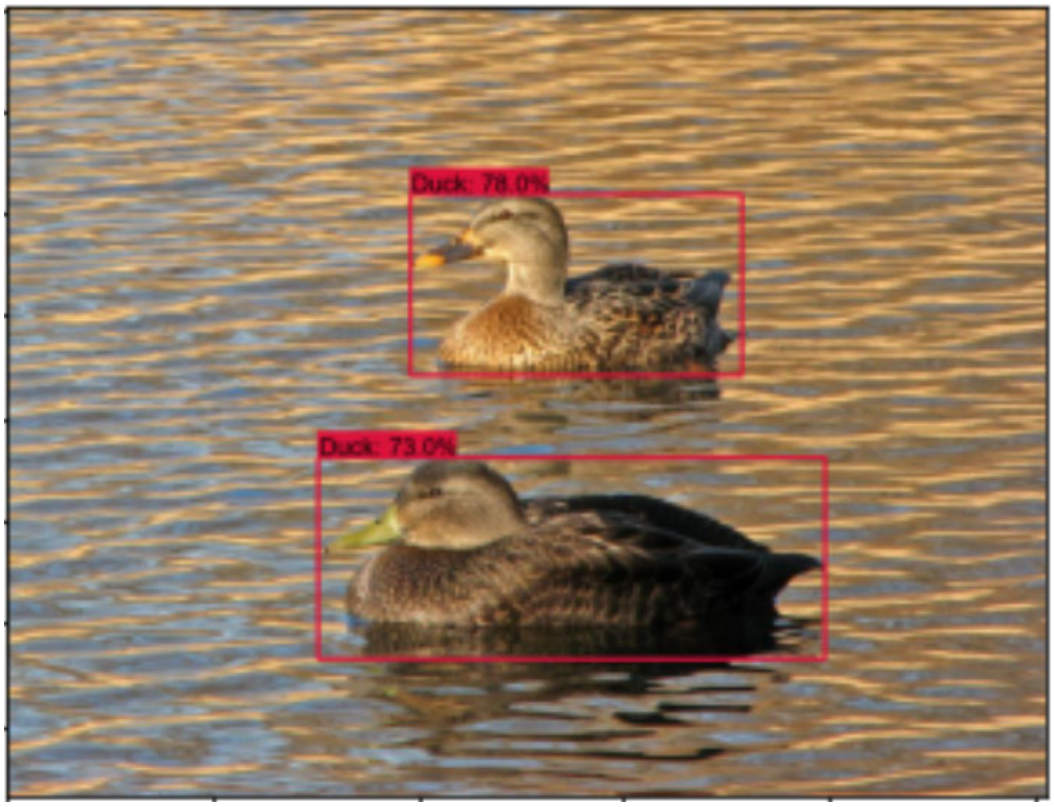

Рисунок 3.9 – Перевірка моделі з новими даними

3.3. Висновки до розділу

В даному розділі було визначено модель, що буде використана в основі для навчання нашої моделі.

Також, визначено набір програм та фреймворків за допомогою яких було створено тренувальний та валідаційний набори даних. Мовою програмування обрано Python, використали дистрибутив Anaconda, а в якості IDE був використаний Jupyter Notebook.

В ході роботи було проведено навчання моделі на навчальних даних, та проведено експеримент, в ході якого визначили, що модель працює та має високий відсоток правильних відповідей.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Розробка програмного забезпечення відбувалося з використанням персональних комп'ютерів. Тому важливим фактором безпеки праці є дотримання основних правил користування комп'ютерною технікою, основних норм та правил охорони праці. Для цього потрібно забезпечити користувачам максимально комфортні та безпечні умови для їх перебування в приміщенні для того, щоб вони якісно та ефективно виконували поставлені завдання.

Вимоги щодо охорони праці, зокрема охорони праці офісних працівників, містять Кодекс законів про працю [18], Закон України "Про охорону праці" [19] та інші нормативно-правові акти.

У відповідності до вимог ст. 153 Кодексу законів про працю України [18] та ст. 6 Закону України "Про охорону праці" [19] на всіх підприємствах, організаціях створюються безпечні і нешкідливі умови праці. Забезпечення відповідних умов праці повинні контролювати власник або уповноважений ним орган.

Будівлі та приміщення, де розміщені робочі місця з комп'ютерами для працівників, повинні відповідати вимогам НПАОП 0.00-7.15-18 "Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями" [20] та "Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин" (ДСанПіН 3.3.2-007-98) [21].

Для внутрішнього оздоблення приміщень з персональними комп'ютерами слід обирати світлі нейтральні кольори стін. Покриття підлоги та поверхня має бути рівною, неслизькою, з антистатичними властивостями.

Робоче місце користувача комп'ютером та іншими подібними пристроями повинно відповідати ергономічним вимогам ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні

ергономічні вимоги з урахуванням характеру та особливостей трудової діяльності [22].

При розміщенні подібних робочих місць необхідно дотримуватись наступних вимог:

- відстань між бічними поверхнями, на яких розміщений обчислювальний прилад, не має бути меншою ніж 1.2 м;
- інтервал між рядами робочих місць має бути не менш 1 м.

Вимоги цих пунктів щодо відстаней та інтервалів беруться з урахуванням особливостей стін та перегородок.

Побудова робочого місця користувача комп'ютерними засобами має забезпечувати зберігання задовільної робочої пози за наступними рекомендованими правилами: підошва – на підлозі або на підставці для ніг стегна – в горизонтальному положенні; передпліччя – вертикально; лікті – під кутом 70°-90° до вертикальної поверхні; зап'ястя зігнуті під кутом не більше 20° відносно горизонтальної поверхні, нахил голови – 20°відносно вертикальної поверхні; монітор персонального комп'ютеру – не менше 60 см від обличчя користувача.

Оскільки інноваційні технології ідуть вперед, деякі галузі потребують роботу з широкоформатними моніторами. У цих випадках відстань у 60см між користувачем та дисплеєм є дуже небезпечною та порушує медичні норми роботи з обчислювальними пристроями. Згідно медичним нормам, оптимальна відстань до монітору повинна складати більше мінімального – від півтора до двох діагоналей. У таких випадках, роботодавець повинен забезпечити безпечну відстань, яка розраховується за наступною формулою:

$$S = L \cdot 2.54 \cdot 1.75 \quad (1)$$

де L – довжина діагоналі монітору у дюймах;

2.54 – коефіцієнт переводу дюймів у сантиметри;

1.75 – середнє арифметичне між 1.5 та двома діагоналями.

Екран обчислювальної машини та клавіатура мають розташовуватись на задовільній відстані від очей користувача, але не ближче 60 см, з урахування розміру алфавітно-цифрових знаків та символів. Клавіатуру необхідно розташовувати на поверхні столу або на спеціальній, регульованій за висотою, робочій площині окремо від столу на відстані 10-30 см від краю, ближчого до працівника. Кут нахилу клавіатури має бути в межах 5°-15°.

Робочий стілець оператора обчислювальної машини повинен бути оснащений наступними елементами: сидінням, спинкою та стаціонарними або змінними підлокітниками. Ширина та глибина сидіння повинні бути не меншими за 40 см. Висота поверхні сидіння має регулюватися в межах 40-50 см, а кут нахилу поверхні – від 15° вперед, до 5° назад. Поверхня сидіння має бути плоскою, передній край – заокругленим, або оздоблена спеціальними ортопедичними засобами. Кут нахилу спинки стільця повинен регулюватися в межах 0°-30° відносно вертикального положення. Відстань від спинки до переднього краю сидіння повинна регулюватись у межах 25-40 см.

4.1.1. Дії працівників в аварійних ситуаціях

Якщо при подання струму на обчислювальну машину або при ввімкненні з’являється спалах, запах або відчувається прямий контакт з струмом, то слід моментально повідомити керівника або головного електрика на підприємстві. Якщо є можливість вимкнути подачу струму самостійно, через вимикач або штепсель, то треба зробити ці маніпуляції.

Виробнича кімната обов’язково повинна бути оснащена наступними приладами для першій допомозі при пожежі: вогнегасником, планом евакуації, табличками з позначкою “Вихід” та номером пожежної служби “101”, порядок дій при виникненні пожежі, пожежні щити та ін. Вся площа виробництва повинна

бути забезпечена технічними знаками безпеки згідно з ДСТУ EN ISO 7010:2019 “Графічні символи. Кольори та знаки безпеки” [23].

У разі загоряння приміщення необхідно терміново дзвонити на гарячу лінію пожежників за номером 101. Поки їдуть пожежники намагатися наявними засобами пожежогасіння самотужки опанувати місце займання, якщо це не сильно загрожує життю.

4.1.2. Надання першої медичної допомоги

Заходи по наданню першої допомоги залежать від того, в якому стані знаходиться потерпілий після звільнення його від електричного струму:

- якщо потерпілий знаходиться в свідомості, а до цього був без або тривалий період знаходився під дією струму, йому необхідно забезпечити повний спокій до появи лікаря або терміново відвести його в лікарню;
- при відсутності свідомості, але з наявністю дихання, потерпілого необхідно покласти рівно, розстебнувши одяг, забезпечити приплив свіжого повітря, дати понюхати нашатирний спирт, побризкати водою, розтерти і зігріти його тіло до появи лікаря;
- якщо потерпілий не дихає або дихає дуже погано (рідко, судорожно з хлипанням) і дихання поступово погіршується, необхідно до появи лікаря робити штучне дихання та непрямий масаж серця;

Поміч потерпілому, яка надається не медичним співробітником, моє надаватися лише до прибуття лікаря.

Послідовність дії при наданні першої допомоги потерпілому при різних травмах прописані в інструкції № 03-ОП “Про надання першої (долікарської) медичної допомоги при нещасних випадках”, яка вивчається робітниками підприємства при проходженні первинного та послідуючих інструктажів з питань охорони праці.

4.2. Підвищення стійкості об’єктів господарської діяльності в умовах надзвичайних ситуацій

4.2.1. Стійкість роботи об’єктів господарської діяльності і фактори, що впливають на їх стійкість

Ефективність економіки держави залежить від того, наскільки окремі галузі господарства здатні стійко працювати не тільки у звичайних умовах, а й в умовах НС мирного та воєнного часу.

Значні руйнування, пожежі та втрати серед населення, викликані наслідками НС, можуть стати причиною різкого скорочення випуску промислової та сільськогосподарської продукції, а отже і зниження економічного потенціалу держави. Виникає потреба завчасного вживання заходів щодо забезпечення стійкої роботи промислових об’єктів на випадок виникнення НС.

Знання можливих НС, характерних для даної місцевості та виробництва, дозволяє диференційовано і цілеспрямовано розробляти та здійснювати заходи, які можуть запобігти аваріям, катастрофам та стихійним лихам або пом’якшити їх наслідки.

Стійкість роботи об’єкта господарської діяльності – це здатність його в умовах НС випускати продукцію у запланованому обсязі та визначеної номенклатури, а у разі слабких та середніх руйнувань або порушення матеріального постачання – відновлювати виробництво власними силами у короткий термін.

На стійкість роботи об’єкта впливають такі фактори:

- захищеність робітників та службовців від уражальних факторів у НС;
- здатність інженерно-технічного комплексу об’єкта (будівель, споруд, обладнання та комунально-енергетичних мереж) протистояти руйнівній дії уражальних факторів аварій, катастроф, стихійного лиха та сучасної зброї;

- надійність постачання об’єкта електроенергією, водою, паливом, комплектуючими та сировиною;
- підготовленість об’єкта до проведення аварійно-рятувальних та відновлюваних робіт;
- оперативність управління виробництвом та здійсненням заходів ЦЗ у НС.

Підвищення стійкості об’єкта досягають проведенням комплексу інженерно-технічних, технологічних, організаційних заходів.

До інженерно-технічних заходів належать роботи, що забезпечують стійкість виробничих будівель і споруд, обладнання та комунально-енергетичних систем.

Технологічні заходи забезпечують підвищення стійкості об’єкта спрощенням технологічного процесу виробництва кінцевої продукції та виключенням або обмеженням розвитку аварій.

Організаційні заходи передбачають розробку ефективних дій керівного складу, служб та формувань ЦЗ, спрямованих на захист виробничого персоналу, проведення рятувальних та інших невідкладних робіт, а також відновлення виробництва.

4.2.2. Шляхи підвищення стійкості об’єктів господарювання господарської діяльності

Основні шляхи підвищення стійкості роботи промислових об’єктів у надзвичайних умовах мирного і воєнного часів:

- забезпечення надійного захисту робітників і службовців від ЗМУ (засобів масового ураження);
- захист виробничих фондів від вражаючих факторів ЗМУ, в тому числі і від вторинних;

- підвищення надійності і оперативності управлінням виробництвом і ЦЗ;
- забезпечення стійкості постачання підприємства електроенергією, газом, водою та інше;
- підготовка об’єкта до проведення відновлювальних робіт.

Підвищення стійкості роботи промислових підприємств в умовах НС мирного і воєнного часів досягається завчасним проведенням комплексу інженерно-технічних, технологічних і організаційних заходів.

Інженерно-технічні заходи (ІТЗ) включають комплекс робіт по підвищенню міцності і надійності будинків, споруд комунально-енергетичних систем, матеріально-технічних запасів.

Технологічні заходи спрямовані на підвищення стійкості виробництва шляхом заміни існуючого технологічного режиму роботи на такий, що виключає можливість виникнення вторинних вражаючих факторів.

Організаційні заходи передбачають розробку і планування дій в умовах НС керівного складу об’єкту, штабу, служб та невоєнізованих формувань ЦЗ по захисту робітників і службовців, проведення рятувальних робіт та відновлення порушеного виробництва.

4.3. Висновки до розділу

1. Розглянуто та розібрано загальні положення безпеки під час роботи з електронно-обчислювальними машинами та приладами вводу-виводу. Розглянутий та сформований перелік вимоги до робочих конструкцій та його додаткових обладнань, до виробничих кімнат та освітлення, де використовують обчислювальні машини, електрична та пожежна безпека при роботі з комп’ютерними установами.

2. Розібрано та сформовано основні правила дій працівників під час НС. Сформульовано основні дії працівників у разі появи аварійних ситуацій та послідовні дії при наданні першої медичної допомоги потерпілим.

ВИСНОВКИ

Минуло 30 років з моменту випуску першої глибокої нейронної мережі Lenet-5, яка використовувалася для розпізнавання рукописних цифр, і за 20 років у нас є моделі, які не тільки класифікують об’єкти, але й можуть сегментувати кожен об’єкт на рівні пікселя та визначити його.

У даній дипломній роботі були розглянуті деякі методи глибокого навчання, які з різним ступенем успішності знайшли практичне застосування у житті. Зокрема, особливу увагу було виділено згортковій нейронній мережі (CNN). CNN складається з групи шарів згортки та підвибірки, та завершується шаром передбаченням, що розпізнає мітку класу для вхідних даних мережі (зазвичай зображення). Архітектура розглянутих у цій роботі моделей, безсумнівно, вплинула вибір методу вирішення завдання виявлення біооб’єктів на зображені, що підкреслює важливість вивчення різних принципів вирішення подібних завдань.

На основі даних, отриманих в процесі аналізу, сформульовано задачу створення системи класифікації та ідентифікації зображень біооб’єктів з використанням відкритих бібліотек Python.

Також, у другій частині дипломної роботи ми детально вивчили, як були побудовані новітні моделі глибокого навчання з використанням базової та передової архітектур CNN. Кожна з цих моделей має свої недоліки та переваги перед іншими, наприклад, модель ResNet має найвищу точність, але вона повільна, її можна використовувати для додатків, де точність є важливим фактором, а швидкість ні.

Коротко описано принципи навчання нейронних мереж. Оглянуто питання підготовки даних до навчання та тестування моделей, що є важливим у процесі вирішення задач машинного навчання.

Варто зазначити, що технології, використані в цьому проекті, не тільки успішно використовуються сьогодні, але й мають перспективне майбутнє. Незважаючи на те, що цей проект можна використовувати тільки для класифікації та ідентифікації біооб'єктів на зображені, цю програму можна використовувати в будь-якій сфері, де потрібне виявлення об'єктів шляхом зміни набору даних.

Що стосується потенційних покращень, було б цікаво застосувати більше методів обробки зображень для збільшення навчальних даних. У цьому проекті було використано випадкове кадрування зображення, яке випадковим чином обрізає зображення. Однак варто спробувати використовувати інші перетворення, такі як обертання або зсув.

Проведено збір тестових даних, а саме картинок для 62 класів із обмежувальними рамками у кількості, у середньому більше ніж 900 штук для кожного класу. Проведено підсумковий експеримент навчання нейронної мережі на тестових даних. Система визначає біооб'єкти з точністю результату до 90%. Також систему навчено виділяти біооб'єкти на фото обмежувальними рамками. Тестову систему було розроблено використовуючи IDE Jupyter Notebook на мові програмування Python 3.7, який було встановлено за допомогою дистрибутива Anaconda. Набір тренувальних даних получений із Open Images v6, було створено за допомогою фреймворка FiftyOne.

Тренування моделі проводилось за допомогою фреймворка TensorFlow Object Detection API. Попередньо навчена модель SSD ResNet50 v1 FPN 640x640, була використана за основу при тренуванні власної.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Löwel. S. and Singer. W. Selection of Intrinsic Horizontal Connections in the Visual Cortex by Correlated Neuronal Activity. (1992) Science 255. pp. 209–212
2. Нейронні мережі. Історія: від 1980-х років до сьогодення [Електронний ресурс] // Режим доступу: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history2.html.
3. Штучні нейронні мережі [Електронний ресурс] // Режим доступу: https://devopedia.org/artificial-neural-network
4. Історія нейронних мереж [Електронний ресурс] // Режим доступу: http://www2.psych.utoronto.ca/users/reingold/courses/ai/cache/neural4.html
5. Метод зворотного поширення [Електронний ресурс] // Режим доступу: https://www.sciencedirect.com/topics/engineering/backpropagation-learning
6. Конкурс ILSVRC2012 [Електронний ресурс] // Режим доступу: image-net.org/challenges/LSVRC/2012/
7. Фреймворк TensorFlow [Електронний ресурс] // Режим доступу: https://www.tensor-flow.org/
8. Перша стабільна версія TensorFlow [Електронний ресурс] // Режим доступу: https://www.wikiwand.com/en/TensorFlow
9. Ліцензія Apache Open Source [Електронний ресурс] // Режим доступу: https://www.apache.org/
10. Сервіс Google Cloud [Електронний ресурс] // Режим доступу: https://cloud.google.com/functions
11. Запуск TensorFlow 2.0 [Електронний ресурс] // Режим доступу: https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html

12. Метод штучного розширення тренувальних даних [Електронний ресурс] // Режим доступу: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/
13. Метод ранньої зупинки [Електронний ресурс] // Режим доступу: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/
14. Метод випадання [Електронний ресурс] // Режим доступу: https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf
15. Сумісні відеокарти [Електронний ресурс] // Режим доступу: https://developer.nvidia.com/cuda-gpus
16. Бібліотека для завантаження тренувальних даних [Електронний ресурс] // Режим доступу: https://voxel51.com/docs/fiftyone/
17. Рекомендації для створення тренувальних даних [Електронний ресурс] // Режим доступу: https://petewarden.com/2017/12/14/how-many-images-do-you-need-to-train-a-neural-network/
18. Про охорону праці [Текст]: Закон України від 14.10.1992 № 2694-XII
19. Кодекс законів про працю України [Текст]: закон України від 10 грудня 1971 року № 322-VIII
20. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [Текст]: наказ Міністерства соціальної політики України від 14.02.2018 № 207
21. ДСанПІН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [Текст]: Постанова Головного державного санітарного лікаря України від 10 грудня 1998 р. № 7
22. ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги [Текст]: Постанова від 01.07.2017 № 1102
23. ДСТУ EN ISO 7010:2019 «Графічні символи. Кольори та знаки безпеки» [Текст]: Наказ Держспоживстандарту України від 24.06.2019 р. № 174

ДОДАТКИ

ДОДАТОК А

(Технічне завдання)

ДОДАТОК Б

(Тези)

ДОДАТОК В

(Лістинг коду)

Вихідний код файлу “load_dataset.ipynb”

```
import fiftyone as fo
import fiftyone.zoo as foz

fo.config.dataset_zoo_dir = "C:/Users/miche/Datasets"

categories = ["Apple", "Banana", "Bird", "Bread",
        "Brown bear", "Bull", "Cabbage", "Camel",
        "Carrot", "Cat", "Chicken",
        "Crab", "Crocodile", "Cucumber", "Dog",
        "Dolphin", "Duck", "Eagle", "Elephant",
        "Fish", "Flower", "Fox", "Frog",
        "Giraffe", "Goat", "Grape",
        "Horse", "Kangaroo", "Lemon", "Leopard",
        "Lion", "Lizard", "Monkey", "Moths and butterflies",
        "Mouse", "Mushroom", "Orange",
        "Owl", "Panda", "Parrot", "Penguin",
        "Person", "Pig", "Plant", "Polar bear",
        "Pumpkin", "Rabbit", "Raccoon", "Reptile",
        "Rhinoceros", "Rose", "Sea lion",
        "Shark", "Snake", "Sparrow",
        "Spider", "Squirrel", "Starfish", "Strawberry",
        "Tiger", "Tomato", "Tortoise",
        "Watermelon", "Zebra"]

for cat in categories:
    dataset = foz.load_zoo_dataset(
```

```
        "open-images-v6",
        label_types=["detections"],
        classes = [f"{cat}"],
        dataset_name=f"open-images-{cat}-v2",
        max_samples=1000
    )

    dataset.export(
      export_dir=f"C:/Users/miche/Datasets/Bioobjects detection dataset/{cat}",
      dataset_type = fo.types.VOCDetectionDataset,
    )
```

Вихідний код файлу “split_dataset.ipynb”

```
import os
import re
from shutil import copyfile
import argparse
import math
import random

root_dir = "C:/Users/miche/Datasets/Bioobjects detection dataset"
tfrecords_dir = "D:/Jupyter Notebook Workspace/annotations"

def iterate_dir(source, dest, ratio, copy_xml):
    source = source.replace('\\', '/')
    dest = dest.replace('\\', '/')
    train_dir = os.path.join(dest, 'train')
    test_dir = os.path.join(dest, 'test')

    if not os.path.exists(train_dir):
        os.makedirs(train_dir)
```

```
    if not os.path.exists(test_dir):
        os.makedirs(test_dir)

    images = [f for f in os.listdir(root_dir)
            if re.search(r'([a-zA-Z0-9\s_\\.\-\(\):])+(?i)(.jpg|.jpeg|.png)$', f)]

    num_images = len(images)
    num_test_images = math.ceil(ratio*num_images)

    for i in range(num_test_images):
        idx = random.randint(0, len(images)-1)
        filename = images[idx]
        copyfile(os.path.join(root_dir, filename),
                os.path.join(test_dir, filename))
        if copy_xml:
            xml_filename = os.path.splitext(filename)[0]+'.xml'
            copyfile(os.path.join(root_dir, xml_filename),
                    os.path.join(test_dir,xml_filename))
        images.remove(images[idx])

    for filename in images:
        copyfile(os.path.join(root_dir, filename),
                os.path.join(train_dir, filename))
        if copy_xml:
            xml_filename = os.path.splitext(filename)[0]+'.xml'
            copyfile(os.path.join(root_dir, xml_filename),
                    os.path.join(train_dir, xml_filename))

def main():
    iterate_dir(root_dir, root_dir, 0.1, True)
main()
```

Вихідний код файлу “generate_tfrecord.ipynb”

```
import os
import glob
import pandas as pd
from shutil import copyfile, move
import io
import xml.etree.ElementTree as ET
import argparse

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow.compat.v1 as tf
from PIL import Image
from object_detection.utils import dataset_util, label_map_util
from collections import namedtuple

label_map = label_map_util.load_labelmap(os.path.join(root_dir, "label_map.txt"))
label_map_dict = label_map_util.get_label_map_dict(label_map)
train_dir = "C:/Users/miche/Datasets/Bioobjects detection dataset/train"
false_dir = "C:/Users/miche/Datasets/Bioobjects detection dataset/train/false"

classes = {"Apple", "Banana", "Bird", "Bread",
        "Brown bear", "Bull", "Cabbage", "Camel",
        "Carrot", "Cat", "Chicken",
        "Crab", "Crocodile", "Cucumber", "Dog",
        "Dolphin", "Duck", "Eagle", "Elephant",
        "Fish", "Flower", "Fox", "Frog",
        "Giraffe", "Goat", "Grape",
        "Horse", "Kangaroo", "Lemon", "Leopard",
        "Lion", "Lizard", "Monkey", "Moths and butterflies",
        "Mouse", "Mushroom", "Orange",
        "Owl", "Panda", "Parrot", "Penguin",
        "Person", "Pig", "Polar bear",
        "Pumpkin", "Rabbit", "Raccoon", "Reptile",
```

```
            "Rhinoceros", "Sea lion",
            "Shark", "Snake", "Sparrow",
            "Spider", "Squirrel", "Starfish", "Strawberry",
            "Tiger", "Tomato", "Turtle",
            "Watermelon", "Zebra"}

def check_same_bbox(objt, xml_list = []):
    for obj in xml_list:
        if (obj[0] == objt[0] and abs(obj[4]-objt[4])<=40 and abs(obj[5]-objt[5])<=40
        and abs(obj[6]-objt[6])<=40 and abs(obj[7]-objt[7])<=40 and obj[3] != objt[3]):
            print(f"{obj[0]}")
            file = obj[0].split('.')
            xml_file = file[0] + '.xml'
            print(xml_file)
            print(f"{obj[3]} == {objt[3]}")

            if not os.path.exists(os.path.join(train_dir, obj[0])):
                continue

            move(os.path.join(train_dir, obj[0]),
                 os.path.join(false_dir, obj[0]))
            move(os.path.join(train_dir, xml_file),
                 os.path.join(false_dir, xml_file))
            return True
        else: continue

    return False

def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        print(xml_file)
        tree = ET.parse(xml_file)
```

```
        root = tree.getroot()
        filename = root.find('filename').text
        width = int(root.find('size').find('width').text)
        height = int(root.find('size').find('height').text)
        for member in root.findall('object'):
            bndbox = member.find('bndbox')
            class_text = member.find('name').text

            if class_text == "Tortoise":
                class_text = "Turtle"

            value = (filename,
                     width,
                     height,
                     class_text,
                     int(float(bndbox.find('xmin').text)),
                     int(float(bndbox.find('ymin').text)),
                     int(float(bndbox.find('xmax').text)),
                     int(float(bndbox.find('ymax').text))
                     )
            if value[3] not in classes:# or check_same_bbox(value, xml_list) == True:
                continue;

            xml_list.append(value)
    column_name = ['filename', 'width', 'height',
                   'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def class_text_to_int(row_label):
    return label_map_dict[row_label]
```

```
def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.keys(), gb.groups)]


def create_tf_example(group, path):
    with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf-8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    for index, row in group.object.iterrows():
        classes_text.append(row['class'].encode('utf-8'))
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
```

```
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }))

    return tf_example

def main():

    writer = tf.python_io.TFRecordWriter(os.path.join(tfrecords_dir, "train.tfrecord"))
    path = os.path.join(os.path.join(root_dir, "train"))
    examples = xml_to_csv(os.path.join(root_dir, "train"))
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())
    writer.close()
    print('Successfully created the TFRecord file: {}'.format(os.path.join(tfrecords_dir,
"train.tfrecord")))
     examples.to_csv(os.path.join(root_dir, "train.csv"), index=None)
     print('Successfully created the CSV file: {}'.format(os.path.join(root_dir, "train.csv")))

main()
```

Вихідний код файлу “object_detection.ipynb”

```
import io
import os
import scipy.misc
import numpy as np
import six
import time
import glob
from IPython.display import display

from six import BytesIO

import matplotlib
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw, ImageFont

import tensorflow as tf
from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util

%matplotlib inline

import time
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils

PATH_TO_MODEL_DIR = 'D:/Jupyter Notebook Workspace/exported-models/my_model'
PATH_TO_SAVED_MODEL = PATH_TO_MODEL_DIR + "/saved_model"

print('Loading model...', end='')
start_time = time.time()

detect_fn = tf.saved_model.load(PATH_TO_SAVED_MODEL)
```

```
end_time = time.time()
elapsed_time = end_time - start_time
print('Done! Took {} seconds'.format(elapsed_time))

PATH_TO_LABELS = 'D:/Jupyter Notebook Workspace/annotations/label_map.txt'
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
                                                                    use_display_name=True)

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')   # Suppress Matplotlib warnings

def load_image_into_numpy_array(path):
    return np.array(Image.open(path))

for image_path in glob.glob('D:/Jupyter Notebook Workspace/Datasets/test/*.jpg'):

    print('Running inference for {}... '.format(image_path), end='')

    image_np = load_image_into_numpy_array(image_path)
    input_tensor = tf.convert_to_tensor(image_np)
    input_tensor = input_tensor[tf.newaxis, ...]

    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                   for key, value in detections.items()}
    detections['num_detections'] = num_detections
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
```

```
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_detections,
        detections['detection_boxes'],
        detections['detection_classes'],
        detections['detection_scores'],
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=3,
        min_score_thresh=.2,
        agnostic_mode=False)

    plt.figure()
    plt.imshow(image_np_with_detections)
    print('Done')
plt.show()
```

ДОДАТОК Г

(CD з програмним кодом)