МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
В.о. завідувача кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент
_____ Є. В. Сіденко
«____»_____202_ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ДОСЛІДЖЕННЯ АРХІТЕКТУР НЕЙРОННИХ МЕРЕЖ ДЛЯ ФІЛЬТРАЦІЇ КОНТЕНТУ

Спеціальність 122 «Комп'ютерні науки»

122 – МКР – 601.21710127

Виконав студент 6-го курсу, групи 601
_____ М. М. Смоленський
«16» лютого 2023 р.

Керівник: канд. техн. наук, доцент
_____Є. В. Сіденко
«16» лютого 2023 р.

Миколаїв – 2023

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**

Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри інтелектуальних інформаційних систем, канд. техн. наук, доцент
_____ Є. В. Сіденко
« » 20 р.

ЗАВДАННЯ

на магістерську кваліфікаційну роботу

Смоленського Микити Михайловича
(прізвище, ім'я, по батькові)

1. Тема магістерської кваліфікаційної роботи «Дослідження архітектур нейронних мереж для фільтрації контенту».
Керівник роботи Сіденко Євген Вікторович, канд. техн. наук, доцент.
Затв. наказом Ректора ЧНУ ім. Петра Могили від «03» листопада 2022 р. № 199
2. Строк подання студентом роботи 16 лютого 2023 р.
3. Вхідні (початкові) дані до роботи: датасет контенту, що підлягає фільтрації, варіації архітектур нейронних мереж. Очікуваний результат: оптимальна архітектура нейронної мережі для фільтрації забороненого контенту.
4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):
   - аналіз предметної області і сучасного стану задачі фільтрації забороненого контенту;
   - огляд існуючих методів фільтрації контенту;
   - проектування системи;
   - розробка і тестування системи.
5. Перелік графічного матеріалу: презентація.
6. Завдання до спеціальної частини: розробити методичні рекомендації стосовно проведення лабораторної роботи на бази групи студентів; провести огляд основних

понять охорони праці, які корелюють з галуззю створення програмного продукту, описати робоче місце і основі засади реагування під час надзвичайних ситуацій.

7. Консультанти:

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис**
Спеціальна частина з охорони праці	Григор'єва Л. І., докт. біол. наук, професор	
Методична частина	Сіденко Є. В., канд. техн. наук, доцент	

Керівник роботи __канд. техн. наук, доцент Сіденко Є. В._____

(наук. ступінь, вчене звання, прізвище та ініціали)

_____

(підпис)

Завдання прийнято до виконання ___Смоленський М. М._____

(прізвище та ініціали)

_____

(підпис)

Дата видачі завдання «__07__» _____листопада____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН
Виконання магістерської кваліфікаційної роботи

Тема: Дослідження архітектур нейронних мереж для фільтрації контенту

№	Найменування роботи	Початок	Закінчення	Примітки
1	Визначення керівника і теми МКР. Подання заяви на затвердження теми МКР	01.09.2022	20.10.2022	
2	Отримання завдання на виконання МКР	21.10.2022	10.11.2022	
3	Складання календарного плану на період виконання МКР	11.11.2022	15.11.2022	
4	Огляд літератури за темою дослідження	16.11.2022	27.11.2022	
5	Проходження переддипломної практики, збір та аналіз матеріалів до МКР	28.11.2022	18.12.2022	
6	Аналіз предметної області та розробка технічного завдання. Моделювання результатів	19.12.2022	12.01.2023	
7	Опис фахової частини МКР, зокрема дослідження публікацій щодо фільтрації контенту, огляд існуючих архітектур штучних нейронних мереж для вирішення поставленої задачі, реалізація обраних технологій з аналізом отриманих результатів	13.01.2023	25.01.2023	
8	Розробка спеціальної частини з охорони праці та методичної частини	26.01.2023	02.02.2023	
9	Попередній захист МКР на засіданні комісії кафедри	03.02.2023	03.02.2023	
10	Корегування роботи за результатами попереднього захисту	04.02.2023	06.02.2023	
11	Остаточне оформлення пояснювальної записки та слайдів доповіді для захисту	07.02.2023	09.02.2023	
12	Подання МКР рецензенту	09.02.2023	10.02.2023	
13	Рецензування МКР	11.02.2023	12.02.2023	
14	Подання МКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.02.2023	16.02.2023	
15	Захист МКР перед екзаменаційною комісією (ЕК)	22.02.2023	23.02.2023	

Розробив студент _____Смоленський М. М._____ _____
(прізвище та ініціали) (*підпис*)

Керівник роботи _____канд. техн. наук, доцент Сіденко Є. В._____ _____
(*наук. ступінь, вчене звання, прізвище та ініціали)* *(підпис)*

«12» листопада 2022 р.

АНОТАЦІЯ

до магістерської кваліфікаційної роботи

студента групи 601 ЧНУ ім. Петра Могили

Смоленського Микити Михайловича

на тему: **“ДОСЛІДЖЕННЯ АРХІТЕКТУР НЕЙРОННИХ МЕРЕЖ ДЛЯ ФІЛЬТРАЦІЇ КОНТЕНТУ”**

Актуальність даного дослідження полягає в необхідності фільтрації контенту з високою точністю за рахунок створення оптимальних варіацій архітектур нейронних мереж. Наявні сьогодні рішення дають низьку точність фільтрації, що призводить до блокування потенційно безпечного контенту. Повна відсутність фільтрації призведе до того, що неповнолітні користувачі та інші вразливі групи отримають до нього доступ, що неприпустимо.

Об’єктом дослідження є процеси фільтрації контенту.

Предметом дослідження є методи і технології побудови нейронних мереж для фільтрації контенту.

Мета роботи полягає в підвищенні точності фільтрації контенту за рахунок розробки архітектури нейронної мережі.

В якості засобу розроблення системи було обрано PyCharm та мову програмування Python. Інструментами розроблення були обрані бібліотеки SciPy та Keras, бібліотека для імпорту та експорту даних Pickle та бібліотека для роботи з алгоритмами нейронних мереж Sckit-learn.

Перший розділ присвячено огляду основних понять предметної області; другий розділ містить в собі аналіз засобів розробки; третій розділ роботи присвячено моделювання майбутньої системи; четвертий розділ присвячено розробці і тестуванню системи; п’ятий розділ — методичний; шостий розділ містить інформацію стосовно охорони праці.

Результатом роботи є побудована архітектура нейронної мережі для фільтрації забороненого контенту.

Загальний обсяг роботи – 109 сторінок. Магістерська кваліфікаційна робота містить 9 рисунків, 2 таблиці і посилання на 46 літературних джерел.

Ключові слова: *нейронна мережа, заборонений контент, фільтрація контенту, архітектура нейронних мереж.*

ABSTRACT

to the master's qualification work by the student of the group 601 of Petro Mohyla Black Sea National University

Smolenskyi Mykyta

“RESEARCH OF NEURAL NETWORK ARCHITECTURES FOR CONTENT FILTERING”

The relevance of this study lies in the need to filter content with high accuracy due to the creation of optimal variations of neural network architectures. The solutions available today provide low filtering accuracy, which leads to the blocking of potentially safe content. A complete lack of filtering will lead to the fact that minor users and other vulnerable groups will gain access to it, which is unacceptable.

The object of research is content filtering processes.

The subject of research is the methods and technologies of building neural networks for content filtering.

The purpose of the work is to increase the accuracy of content filtering by developing a neural network architecture.

PyCharm and the Python programming language were chosen as a system development tool. The SciPy and Keras libraries, the Pickle data import and export library, and the Sckit-learn library for working with neural network algorithms were chosen as development tools.

The first section is dedicated to the overview of the main concepts of the subject area; the second section contains an analysis of development tools; the third section of the work is devoted to the modeling of the future system; the fourth section is devoted to the development and testing of the system; the fifth section is methodical; the sixth section contains information on labor protection.

The result of the work is the constructed neural network architecture for filtering prohibited content.

The total volume of work is 109 pages. The master's thesis contains 9 figures, 2 tables and references to 46 literary sources.

Keywords: neural network, prohibited content, content filtering, neural network architecture.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ANN – Artificial Neural Network

URL – Uniform Resource Locator

DNS – Domain Name System

RL – Reinforcement Learning

ML – Machine Learning

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему:

«ДОСЛІДЖЕННЯ АРХІТЕКТУР НЕЙРОННИХ МЕРЕЖ ДЛЯ ФІЛЬТРАЦІЇ КОНТЕНТУ»

Спеціальність 122 «Комп’ютерні науки»

122 – МКР – 601.21710127

Виконав студент 6-го курсу, групи 601
_____ М. М. Смоленський
«16» лютого 2023 р.

Керівник: канд. техн. наук, доцент
_____Є. В. Сіденко
«16» лютого 2023 р.

Миколаїв – 2023

ВСТУП

Актуальність теми дослідження. Завдяки онлайн-технологіям люди отримали величезні можливості для спілкування, набуття нових навичок, творчості та участі у створенні кращого суспільства. Проте ці технології також можуть створювати нові ризики, пов’язані з конфіденційністю, незаконним контентом, домаганнями, кібербулінгом та зловживанням особистою інформацією.

В сучасному світі все більш активно набирає обертів процес діджиталізації і інтегрування все більшої кількості програмних продуктів в життя рядового члену соціуму. Кожного дня з’являються нові додатки, які допомагають людям генерувати терабайти контенту, такі як TikTok, Instagram, Telegram, Linked In, тощо. Проте, зі збільшенням кількості контенту збільшується і кількість матеріалів, що порушують правила платформи або закони країни, громадянином якої є користувач. Кожного дня в соціальних мережах з’являються фото і відеоматеріали, на яких присутні насильство, паління, алкоголь, зброя, пряма або непряма пропаганда наркотиків, тощо.

Виходячи з усього вищесказаного можна зробити висновок про високу актуальність створення різного роду інструментів для автоматичної фільтрації контенту.

Об’єктом дослідження є процеси фільтрації контенту.

Предметом дослідження є методи і технології побудови нейронних мереж для фільтрації контенту.

Мета роботи полягає в підвищенні точності фільтрації контенту за рахунок розробки архітектури нейронної мережі.

Для досягнення поставленої мети необхідно виконати наступні завдання:

– аналіз предметної області та сучасного стану задачі фільтрації забороненого контенту;

– огляд існуючих методів та підходів для фільтрації контенту;

– проектування інтелектуальної системи фільтрації контенту з використанням обраної архітектури нейронної мережі;

– розробка і тестування системи фільтрації контенту.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Поняття фільтрації

Інтернет-фільтр — це програмне забезпечення, яке обмежує або контролює доступ користувачів Інтернету, особливо коли він використовується для обмеження матеріалів, що передаються через Інтернет, електронною поштою чи іншим способом. Програмне забезпечення для контролю вмісту визначає, який вміст дозволено або заблоковано [1-2].

Такі обмеження можуть бути застосовані на різних рівнях: уряди можуть спробувати забезпечити їх дотримання по всій країні, або, наприклад, постачальники Інтернет-послуг для своїх клієнтів, роботодавці для своїх працівників, школи для студентів, відвідувачі бібліотек, батьки для дітей, які використовують комп’ютери, або окремі користувачі для своїх власні комп'ютери. Зазвичай мотивація полягає в тому, щоб запобігти доступу до вмісту, який власник комп’ютера або інші органи влади можуть вважати небажаними. Якщо контроль вмісту запроваджено без згоди користувачів, це можна описати як форму інтернет-цензури. Деяке програмне забезпечення для контролю вмісту містить функції контролю часу, які дозволяють батькам обмежувати час, який діти можуть витрачати на доступ до Інтернету, ігри чи іншу діяльність за комп’ютером [1, 3, 4].

У деяких країнах таке програмне забезпечення є повсюдним. На Кубі текстові процесори або веб-браузери автоматично вимикаються з попередженням про «урядову безпеку», якщо користувач комп’ютера в підконтрольному уряду Інтернет-кафе вводить певні слова [1].

Термін «модерація вмісту» іноді використовують CNN [2], журнал Playboy [3], San Francisco Chronicle [4] і The New York Times [5]. Однак кілька інших термінів включають «програмне забезпечення для фільтрації вмісту», «проксі-сервер фільтрації», «захищений веб-шлюз», «програмне забезпечення для цензури», «безпека та контроль вмісту», «програмне забезпечення для фільтрації

веб-сайтів», «програмне забезпечення для модерування вмісту» та «вміст Часто використовується "програмне забезпечення" для перехоплення. "Няня" також використовується в маркетингу продуктів і ЗМІ. Галузева дослідницька фірма Gartner використовує "Secure Web Gateway" (SWG) для опису сегмента ринку [6].

Компанії, які виробляють продукти, які вибірково блокують веб-сайти, не називають їх цензурним програмним забезпеченням, віддаючи перевагу використанню таких термінів, як «Інтернет-фільтр» або «URL-фільтр»; в окремому випадку програмного забезпечення також використовується «програмне забезпечення для батьківського контролю». Деякі продукти реєструють усі сайти, які відвідує користувач, і оцінюють їх за типом вмісту для звітування особисто обраному «партнеру з підзвітності», використовуючи термін «відповідальне програмне забезпечення». Інтернет-фільтри, програмне забезпечення для батьківського контролю та/або програмне забезпечення для звітування також можуть бути такимиоб'єднані в один продукт.

Проте критики цього типу програмного забезпечення використовують термін «цензурне програмне забезпечення» вільно: розглянемо, наприклад, проект цензурного програмного забезпечення [7]. Використання терміну «цензурне програмне забезпечення» в редакційних статтях, які критикують виробників програмного забезпечення, є широко поширеним і охоплює багато різних видів і застосувань: Ксенія Джардін у редакційній статті в The New York Times, 9 березня 2006 р. Термін використовувався для обговорення використання програмного забезпечення фільтрації США для придушення вмісту в Китаї; того ж місяця старшокласник використав цей термін, щоб обговорити впровадження такого програмного забезпечення у своєму шкільному окрузі [8, 9].

Загалом, традиційні газети не використовують термін «цензурне програмне забезпечення» у своїх звітах, за винятком вищезгаданих редакційних сторінок, віддаючи перевагу використанню менш суперечливих термінів, таких як «фільтр вмісту», «контроль вмісту» або «веб-фільтрація». «Нью-Йорк Таймс» і «Уолл Стріт Джорнал», схоже, наслідували їхній приклад. З іншого боку, веб-газети, такі як

CNET, використовують цей термін у редакційному та журналістському контексті, наприклад «Windows Live to Get Censorware» [10].

1.2 Методи фільтрації контенту

Фільтри можна реалізувати різними способами: за допомогою програмного забезпечення на вашому персональному комп’ютері, через мережеву інфраструктуру, таку як проксі-сервери, DNS-сервери або брандмауери, які забезпечують доступ до Інтернету. Жодне єдине рішення не може забезпечити повне покриття, тому більшість компаній розгортають комбінацію технологій, щоб досягти адекватного контролю вмісту відповідно до своєї політики.

Фільтр браузера. Рішення для фільтрації вмісту на основі браузера — це найпростіше рішення для фільтрації вмісту, реалізоване за допомогою сторонніх розширень для браузера.

Фільтр електронної пошти. Фільтри електронної пошти впливають на тіло електронної пошти, заголовки електронної пошти (наприклад, відправника та тему) та інформацію, що міститься у вкладеннях електронної пошти, щоб класифікувати, приймати або відхиляти повідомлення. Зазвичай використовується фільтр Байєса, статистичний фільтр. Доступні фільтри як на стороні клієнта, так і на стороні сервера.

Клієнтський фільтр. Цей тип фільтра встановлюється як програмне забезпечення на кожному комп’ютері, який потребує фільтрації [11, 12]. Як правило, будь-хто з правами системного адміністратора може керувати, вимикати або видаляти цей фільтр. Фільтри на стороні клієнта на основі DNS повинні налаштовувати воронку DNS, наприклад Pi-Hole.

Інтернет-провайдери з обмеженим (або відфільтрованим) вмістом. Інтернет-провайдери, що обмежують (або фільтрують) вміст, — це постачальники Інтернет-послуг, які надають доступ лише до певних частин Інтернет-вмісту за згодою або в обов’язковому порядку. Усі, хто підписується на такі послуги,

обмежені. Типи фільтрів можна використовувати для забезпечення державного [13] регулювання [14] або батьківського контролю над абонентами.

Веб-фільтрація. Цей тип фільтра реалізовано як прозорий проксі на транспортному рівні або як веб-проксі на прикладному рівні [15]. Програмне забезпечення для фільтрації може містити функції запобігання втраті даних для фільтрації вихідної та вхідної інформації. Усі користувачі регулюються інституційною політикою доступу. Фільтрування можна налаштувати, тому бібліотека середньої школи району може мати інший профіль фільтрації, ніж бібліотека середньої школи округу.

Фільтрація на основі DNS. Цей тип фільтрації застосовується на рівні DNS і намагається заблокувати пошук доменів, які не відповідають набору політик (батьківський контроль або правила компанії). Деякі безкоштовні загальнодоступні служби DNS пропонують параметри фільтрації як частину своїх послуг. Для цієї мети також можна використовувати воронки DNS, такі як Pi-Hole, але лише для клієнтів.

Фільтр пошукової системи. Багато пошукових систем, таких як Google і Bing, пропонують користувачам можливість увімкнути фільтри безпеки. Коли цей фільтр безпеки активовано, він фільтрує невідповідні посилання з усіх результатів пошуку. Якщо користувач знає фактичну URL-адресу сайту, що містить непристойний вміст або вміст для дорослих, Вони можуть отримати доступ до цього вмісту без використання пошукової системи. Деякі інтернет-провайдери пропонують адаптовані до дітей версії своїх механізмів, дозволяючи лише сайти, орієнтовані на дітей [16].

1.3 Поняття нейронної мережі

Нейронна мережа - система, призначена для обробки інформації, структура і принцип дії якої певною мірою моделюються за функціонуванням фрагментів реальної (біологічної) нервової системи. Схеми штучних нейронів, що входять до

складу мережі, і (певною мірою) її структура базуються на біологічних передумовах. Проте схеми зв'язку нейронів у нейронній мережі вибираються довільно і не є моделлю реальних нейронних структур [1].

Відмінною рисою нейронної мережі як ІТ-інструменту є можливість вирішення практичних задач без їх попередньої математичної формалізації. Додаткова перевага полягає в тому, що немає необхідності посилатися на будь-які теоретичні припущення щодо проблеми, що вирішується при використанні мережі [1].

Найсуттєвішою особливістю нейронної мережі є її здатність навчатися на прикладах і здатність автоматично узагальнювати набуті знання (генералізація) [1].

Іноді назву «штучні нейронні мережі» використовують для опису міждисциплінарної галузі знань, що стосується побудови, навчання та тестування можливостей цього типу мережі.

Типи нейронних мереж. Загальною рисою всіх нейронних мереж є те, що їх структура складається з нейронів, з’єднаних між собою синапсами. Синапси пов’язані з вагами, тобто числовими значеннями, інтерпретація яких залежить від моделі.

Односпрямовані мережі. Односторонні мережі — це нейронні мережі, в яких немає зворотного зв’язку, тобто один шаблон або сигнал проходить через кожен нейрон рівно один раз за його цикл. Найпростішою нейронною мережею є персептрон з одним порогом, розроблений МакКалохом і Піттсом у 1943 році.

У більш просунутих рішеннях використовуються функції переходу. Найпопулярнішим класом функцій, які використовуються в нейронних мережах, є сигмоїдальні функції, наприклад, гіперболічний тангенс. Мережа, побудована з нейронів, оснащених функцією нелінійного переходу, здатна до нелінійного розділення вхідних шаблонів. Отже, це універсальний класифікатор.

Алгоритми градієнтного спуску, включаючи алгоритм зворотного поширення, використовуються для навчання багатошарових персептронів.

Односпрямовані мережі поділяються на одношарові, двошарові та багатошарові. Одношарові мережі можуть вирішити лише вузький клас завдань. Двошарові та багатошарові мережі можуть вирішувати набагато ширший клас і є еквівалентними в цьому відношенні, але для них використовуються різні алгоритми навчання (вони простіші для багаторівневих мереж).

Рекурсивні мережі. Рекурсивна мережа - це мережа, в якій зв'язки між нейронами є графом із циклами. Серед різноманітності рекурсивних моделей штучних нейронних мереж можна виділити наступні:

– мережа Хопфілда - система щільно взаємопов'язаних нейронів (один з одним, але без зворотних зв'язків), що реалізують динаміку, що гарантує конвергенцію до бажаних патернів;

– машина Больцмана – стохастична модифікація мережі Хопфілда, розроблена Джеффом Хінтоном і Террі Сейновським; ця модифікація дозволила навчити прихованих нейронів і усунути паразитні патерни за рахунок збільшення часу моделювання;

– мережі Хопфілда та машини Больцмана використовуються як контекстно-адресована пам’ять, для розпізнавання зображень, розпізнавання мови та для вирішення проблем мінімізації (наприклад, проблема комівояжера).

Самоорганізуючі карти. Самоорганізуючі карти (SOM), також відомі як мережі Кохонена, — це мережі нейронів із пов’язаними координатами на лінії, площині чи будь-якому n-вимірному просторі.

Навчання цього типу мережі полягає в зміні координат нейронів таким чином, щоб вони прагнули до шаблону, який відповідає структурі аналізованих даних. Таким чином, мережі «розтягуються» навколо наборів даних, підлаштовуючи під них свою структуру.

Ці мережі використовуються для класифікації шаблонів, наприклад безперервних звуків мови, тексту, музики. Одним із найцікавіших застосувань є створення сітки навколо комп’ютерної моделі сканованого об’єкта.

Також популярними моделями є опорні векторні машини (SVM), мережі на основі радіальних базисних функцій (радіальні мережі, RBF) і нейронні мережі протипоширення. Відносно новою моделлю є мережі на основі імпульсних нейронів.

Matlab. Matlab — це інтерактивне середовище, де виконуються інженерні розрахунки. Він оснащений власною мовою програмування, яка дозволяє писати повнофункціональні програми. З точки зору графіки, є можливість малювати дво- і тривимірні діаграми, а також візуалізувати результати розрахунків у вигляді статичних малюнків і відображати їх анімацію. Дані вимірювань можна завантажити для обробки із зовнішнього пристрою через порти. Спеціалізовані функції (так званий інструментарій), якими оснащений Matlab, були створені для: матричних обчислень, нейронних мереж, обробки зображень, обробки сигналів, нечіткої логіки, статистики, перетворення форми сигналу, ідентифікації системи, оптимізації, обчислень з фіксованою комою, фінансових аналіз, збір даних. Для створення нейронної мережі в Matlab використовуються функції бібліотеки Deep Learning Toolbox і Fuzzy Logic Toolbox [2].

Нейронні мережі статистики. Statistica Neural Networks – серед інструментів, доступних для проектування та моделювання нейронних мереж, це найбільш технологічно просунута програма [30]. Унікальністю цієї програми є висока ефективність роботи та наявність багатьох рідкісних інструментів. Також для користувача, який професійно не займається проектуванням мереж, побудова та робота програми не становить труднощів. Для менш досвідчених користувачів доступний майстер, т.зв «Автоматичний конструктор», який допомагає користувачеві на кожному етапі побудови нейронної мережі. Професіонали знайдуть тут великий набір типів нейронних мереж і алгоритмів, які навчають ці мережі [2].

Використання. Сьогодні немає сумніву, що штучні нейронні мережі не є хорошою моделлю мозку [потрібна виноска], хоча їх різні форми виявляють риси, характерні для біологічних нейронних систем: здатність узагальнювати знання,

оновлювати за рахунок раніше вивчених шаблонів, давати неправильні відповіді після переповнення [потрібна виноска]. Незважаючи на спрощену структуру, штучні нейронні мережі іноді використовуються для моделювання захворювань мозку.

Штучні нейронні мережі використовуються для розпізнавання та класифікації шаблонів (призначення шаблонів до категорій), прогнозування часових рядів, статистичного аналізу даних, усунення шумів і стиснення зображень і звуку, а також у питаннях керування та автоматизації.

Журнал BYTE перераховує такі способи використання цих мереж, зокрема:

- діагностика електронних систем;
- психіатричні експертизи;
- прогнози фондового ринку;
- прогнозування продажів;
- розвідка нафти;
- інтерпретація біологічних досліджень;
- прогнози цін;
- аналіз медичних досліджень;
- планування капітального ремонту машин;
- планування прогресу навчання;
- аналіз виробничих проблем;
- оптимізація комерційної діяльності;
- спектральний аналіз;
- оптимізація утилізації відходів;
- вибір сировини;
- вибір об'єктів для проведення криміналістичного дослідження;
- підбір працівників;
- управління виробничими процесами [3].

Зараз найпопулярніші використання нейронних мереж:

- у програмах розпізнавання рукописного тексту (OCR);

– в аеропортах, щоб перевірити, чи містить рентгенівський багаж небезпечні вантажі;

– для синтезу мовлення.

У галузі технічних наук штучні нейронні мережі використовуються наприклад для [4]:

– апроксимація, прогнозування, прогнозування вихідних даних на основі вхідних даних без необхідності явного визначення зв’язку між ними;

– класифікація та розпізнавання;

– зіставлення даних – нейронні мережі дозволяють автоматизувати процеси міркувань і допомагають виявити значні зв’язки між даними;

– аналіз даних, тобто пошук зв'язків між даними.

1.4 Розпізнавання зображень

Розпізнавання шаблонів — це автоматичне розпізнавання шаблонів у даних. Він має застосування в аналізі статистичних даних, обробці сигналів, аналізі зображень, пошуку інформації, біоінформатиці, стисненні даних, комп’ютерній графіці та машинному навчанні. Розпізнавання образів бере свій початок у статистиці та інженерії; завдяки підвищеній доступності великих даних і новій величезній обчислювальній потужності деякі сучасні методи розпізнавання образів включають використання машинного навчання. Однак цю діяльність можна розглядати як два аспекти однієї області застосування, і разом вони зазнали значного розвитку за останні кілька десятиліть.

У багатьох випадках системи розпізнавання образів навчаються на позначених «навчальних» даних, але коли позначені дані недоступні, для виявлення раніше невідомих шаблонів можна використовувати інші алгоритми. KDD і інтелектуальний аналіз даних зосереджені на неконтрольованих методах і більш тісному зв’язку з бізнесом. Розпізнавання образів більше зосереджено на сигналі, враховуючи отримання та обробку сигналу. Він бере свій початок у

інженерії, і цей термін популярний у контексті комп'ютерного зору: провідна конференція з комп'ютерного бачення називається Конференцією з комп'ютерного зору та розпізнавання образів.

У машинному навчанні розпізнавання образів — це призначення міток заданим вхідним значенням. У статистиці дискримінантний аналіз був введений у 1936 році з тією ж метою. Прикладом розпізнавання шаблонів є класифікація, яка намагається призначити кожне вхідне значення одному з даного набору класів (наприклад, визначити, чи є дана електронна пошта «спамом»). або «не спам»). Однак розпізнавання образів є більш загальною проблемою, яка також включає інші типи міркувань. Іншими прикладами є регресія, яка призначає кожному входу вихід із дійсним значенням; тегування послідовності, яке призначає клас кожному члену послідовності значень (наприклад, тегування частини мови, яке призначає частину -мовлення до кожного слова у вхідному реченні); синтаксичний аналізатор призначає дерево синтаксичного аналізу вхідному реченню, описуючи синтаксичну структуру речення.

Алгоритми розпізнавання образів, як правило, розроблені для надання правдоподібних відповідей на всі можливі вхідні дані та виконання «найбільш імовірного» збігу вхідних даних, беручи до уваги їхню статистичну варіацію. Це на відміну від алгоритмів зіставлення шаблонів, які шукають точні збіги з уже існуючими шаблонами у вхідних даних. Поширеним прикладом алгоритму зіставлення за зразком є зіставлення регулярного виразу, яке шукає шаблони заданого типу в текстових даних і включено до функції пошуку багатьох текстових редакторів і текстових процесорів.

1.5 Аналіз досліджень та публікацій

Якщо розглядати методи фільтрації контенту в широкому сенсі — все зводиться до етичної сторони питання, чи слід вводити ті чи інші фільтри та як це відобразиться на сучасному соціумі, окремих користувачах мережі інтернет та

окремих інтернет-ресурсах та спільнотах, які на поточний час подібного роду систем не мають.

Проте, при простій декомпозиції можна зробити висновок, що задача зводиться до розпізнавання образів.

Першими літературними джерелами, що знаходяться в мережі за запитом «розпізнавання образів» є такі публікації:

1. Методи розпізнавання образів. / В. М. Заяць, Р. М. Камінський [1];
2. Дэвід А. Форсайт, Джин Понс. [Computer Vision: A Modern Approach] [2];
3. Джордж Стокман, Лінда Шапіро. [Computer Vision] [3];
4. В. Н. Вапнік, А. Я. Червоненкіс Теорія розпізнавання образів [4].

На сьогоднішній день дані дослідження є достатньо вичерпними, щоб розуміти суть області [4] — є однією з найстаріших публікацій стосовно теми і, фактично, стоїть біля витоків даної сфери діяльності людини. Інші — більш нові і мають відношення вже до практичної сторони питання, розглядаючи вирішення реальних задач.

Не дивлячись на те, що в тематиці розпізнавання зображень є велика кількість різного роду публікацій і досліджень — використання розпізнавання зображень для фільтрації контенту є досить недослідженою ділянкою знань. Великі компанії використовують власні системи фільтрації контенту, які підпадають під поняття «корпоративної таємниці», що робить неможливим їх дослідження.

1.6 Постановка задачі

В сучасному світі люди кожної хвили створюють терабайти контенту різного характеру і направленостей, починаючи від навчальних роликів на платформі YouTube і закінчуючи відверто провокативними постами, наприклад, на платформі Twitter.

Зазвичай великі інтернет-конгломерати фільтрують свій контент власноруч, проте, все ще існують маленькі форуми та сайти закритих спільнот, які не мають ресурсів для реалізації подібних рішень.

Загальна проблематика питання полягає в:

а) абсолютній необхідності впровадження системи фільтрації контенту на багатьох інтернет-ресурсах;

б) питання підходу до реалізації системи фільтрації контенту.

У випадку даної роботи основна увага приділена саме другій проблемі. Система фільтрації контенту створюється на базі нейронної мережі і проблема підходу приймає форму проблеми створення максимально підходящої архітектури нейронної мережі, яка в подальшому зможе видавати максимально можливу точність при вхідних даних.

Тобто, резюмуючи, можна сказати, що основна проблема поставленої задачі полягає в створенні архітектури системи, яка матиме можливість з максимальною точністю розпізнавати «заборонені» об’єкти на фото, методом емпіричних досліджень.

Актуальність теми дослідження. Завдяки онлайн-технологіям люди отримали величезні можливості для спілкування, набуття нових навичок, творчості та участі у створенні кращого суспільства. Проте ці технології також можуть створювати нові ризики, пов’язані з конфіденційністю, незаконним контентом, домаганнями, кібербулінгом та зловживанням особистою інформацією.

В сучасному світі все більш активно набирає обертів процес діджиталізації і інтегрування все більшої кількості програмних продуктів в життя рядового члену соціуму. Кожного дня з’являються нові додатки, які допомагають людям генерувати терабайти контенту, такі як TikTok, Instagram, Telegram, Linked In, тощо. Проте, зі збільшенням кількості контенту збільшується і кількість матеріалів, що порушують правила платформи або закони країни, громадянином якої є користувач. Кожного дня в соціальних мережах з’являються фото і відеоматеріали, на яких присутні

насильство, паління, алкоголь, зброя, пряма або непряма пропаганда наркотиків, тощо.

Виходячи з усього вищесказаного можна зробити висновок про високу актуальність створення різного роду інструментів для автоматичної фільтрації контенту.

Об’єктом дослідження є процеси фільтрації контенту.

Предметом дослідження є методи і технології побудови нейронних мереж для фільтрації контенту.

Мета роботи полягає в підвищенні точності фільтрації контенту за рахунок розробки архітектури нейронної мережі.

Для досягнення поставленої мети необхідно виконати наступні завдання:

– аналіз предметної області та сучасного стану задачі фільтрації забороненого контенту;

– огляд існуючих методів та підходів для фільтрації контенту;

– проектування інтелектуальної системи фільтрації контенту з використанням обраної архітектури нейронної мережі;

– розробка і тестування системи фільтрації контенту.

Висновки до розділу 1

Перший розділ роботи присвячено огляду основних аспектів предметної області, таких як нейронні мережі, парадигми навчання нейронних мереж, поняття фільтрації контенту.

На основі усієї отриманої в ході аналізу інформації було поставлено задачу на роботу, яка полягає в аналізі архітектур нейронних мереж для фільтрації контенту.

2 МЕТОДИ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ РОЗРОБКИ НЕЙРОННИХ МЕРЕЖ ДЛЯ ФІЛЬТРАЦІЇ КОНТЕНТУ

2.1 Парадигми навчання нейронних мереж

Існує дві парадигми вивчення нейронних мереж – з учителем і без учителя. У першому випадку вхідний вектор має готові відповіді, у другому – нейронна мережа самонавчається. Кожен вид навчання має свій набір завдань, і в цілому вони не перетинаються. В даний час винайдено і запатентовано велику кількість архітектур нейронних мереж і методів їх навчання. Але головне — для навчання з учителем це «алгоритм зворотного поширення», а для навчання без учителя — алгоритм Хебба та Кохонена. Ці парадигми значною мірою збігаються з біологічними реаліями, такими як – діти навчаються з учителем чи без нього?

Окрім того, нещодавно з’явилася нова третя парадигма — навчання з підкріпленням.

2.1.1 Навчання з учителем

Кероване навчання (SL) — це парадигма машинного навчання для проблем, у яких доступні дані складаються з позначених прикладів, тобто кожна точка даних містить ознаки (коваріати) і пов’язану мітку. Метою алгоритмів керованого навчання є вивчення функції, яка відображає вектори ознак (вхідні дані) у мітки (вихідні) на основі прикладів пар введення-виведення [1]. Він виводить функцію з позначених навчальних даних, що складаються з набору навчальних прикладів [2]. У керованому навчанні кожен приклад є парою, що складається з вхідного об’єкта (зазвичай вектора) і бажаного вихідного значення (також називається контрольним сигналом). Алгоритм навчання під наглядом аналізує навчальні дані та створює передбачувану функцію, яку можна використовувати для відображення нових прикладів. Оптимальний сценарій дозволить алгоритму правильно визначити мітки

класу для невидимих екземплярів. Це вимагає, щоб алгоритм навчання узагальнював навчальні дані на невидимі ситуації «розумним» способом (див. індуктивне зміщення). Ця статистична якість алгоритму вимірюється через так звану помилку узагальнення.

Кроки для виконання. Щоб розв'язати задану проблему навчання під наглядом, необхідно виконати наступні кроки:

– визначте вид навчальних прикладів. Перш ніж робити щось інше, користувач повинен вирішити, який тип даних використовуватиметься як навчальний набір. У випадку аналізу почерку, наприклад, це може бути один рукописний символ, ціле рукописне слово, ціле речення рукописного тексту або, можливо, повний абзац рукописного тексту;

– зберіть навчальний набір. Навчальний набір має відповідати реальному використанню функції. Таким чином, збирається набір вхідних об'єктів, а також збираються відповідні вихідні дані або від людей-експертів, або від вимірювань;

– визначте представлення вхідних ознак вивченої функції. Точність вивченої функції сильно залежить від того, як представлено вхідний об'єкт. Як правило, вхідний об'єкт перетворюється на вектор ознак, який містить ряд ознак, що описують об'єкт. Кількість ознак не повинна бути занадто великою, через прокляття розмірності; але має містити достатньо інформації для точного прогнозування результату;

– визначте структуру вивченої функції та відповідний алгоритм навчання. Наприклад, інженер може вибрати використання опорних векторних машин або дерев рішень;

– завершіть дизайн. Запустіть алгоритм навчання на зібраному навчальному наборі. Деякі контрольовані алгоритми навчання вимагають від користувача визначення певних контрольних параметрів. Ці параметри можна налаштувати шляхом оптимізації продуктивності на підмножині (так званому наборі перевірки) навчального набору або через перехресну перевірку;

– оцініть точність вивченої функції. Після налаштування параметрів і навчання продуктивність результуючої функції повинна бути виміряна на тестовому наборі, який є окремим від навчального набору.

Алгоритми. Найбільш широко використовувані алгоритми навчання:

– опорно-векторні машини;
– лінійна регресія;
– логістична регресія;
– наївний Байєс;
– лінійний дискримінантний аналіз;
– дерева рішень;
– алгоритм K-найближчого сусіда;
– нейронні мережі (багатошаровий персептрон);
– навчання подібності;
– підходи та алгоритми;
– аналітичне навчання;
– штучна нейронна мережа;
– зворотне поширення;
– підвищення (мета-алгоритм);
– Байєсівська статистика;
– міркування на основі випадків;
– навчання дерева рішень;
– індуктивне логічне програмування;
– регресія процесу Гаусса;
– генетичне програмування
– груповий метод обробки даних;
– оцінки ядра;
– навчальні автомати;
– навчання систем класифікації;

– мінімальна довжина повідомлення (дерева рішень, графіки рішень тощо);
– мультилінійне підпросторове навчання;
– наївний байєсівський класифікатор;
– класифікатор максимальної ентропії;
– умовне випадкове поле;
– алгоритм найближчого сусіда;
– ймовірно, приблизно правильне навчання (PAC);
– правила Ripple down, методологія отримання знань;
– алгоритми символічного машинного навчання;
– підсимволічні алгоритми машинного навчання
– опорно-векторні машини;
– машини мінімальної складності (MCM);
– випадкові ліси;
– ансамблі класифікаторів;
– порядкова класифікація
– попередня обробка даних;
– обробка незбалансованих наборів даних;
– статистичне реляційне навчання;
– Proaftn, багатокритеріальний алгоритм класифікації.

2.1.2 Навчання без учителя

Неконтрольоване навчання – це тип алгоритму, який вивчає закономірності з немаркованих даних. Мета полягає в тому, щоб за допомогою мімікрії, яка є важливим способом навчання людей, машина була змушена побудувати стисле уявлення про свій світ, а потім генерувати з нього образний вміст.

На відміну від навчання під наглядом, де дані позначаються тегами експерта, напр. помічені як «куля» або «риба», неконтрольовані методи виявляють самоорганізацію, яка фіксує шаблони як щільність ймовірності [1] або комбінацію

переваг нейронних особливостей, закодованих у вагах і активаціях машини. Інші рівні в спектрі нагляду – це навчання з підкріпленням, коли машині дається лише числова оцінка продуктивності як орієнтир, і напівконтрольоване навчання, де невелика частина даних позначена тегами.

Завдання проти методів. Схильність до використання контрольованих і неконтрольованих методів. Назви завдань, що перетинають межі кола, є навмисним. Це показує, що класичний розподіл творчих завдань (ліворуч) із застосуванням неконтрольованих методів розмитий у сучасних схемах навчання.

Завдання нейронної мережі часто класифікують як дискримінаційні (розпізнавання) або генеративні (уява). Часто, але не завжди, дискримінаційні завдання використовують контрольовані методи, а генеративні – безконтрольні; однак поділ дуже туманний. Наприклад, розпізнавання об’єктів надає перевагу навчанню під наглядом, але неконтрольоване навчання також може об’єднувати об’єкти в групи. Крім того, у міру прогресу деякі завдання використовують обидва методи, а деякі завдання змінюються один на інший. Наприклад, розпізнавання зображень спочатку проходило під інтенсивним наглядом, але стало гібридом із використанням попереднього навчання без нагляду, а потім знову перейшло до нагляду з появою відсіву, повторного навчання та адаптивного навчання.

Навчання. Під час фази навчання неконтрольована мережа намагається імітувати дані, які вона надає, і використовує помилку в імітованих результатах, щоб виправити себе (тобто виправити свої ваги та зміщення). Іноді помилка виражається як низька ймовірність того, що виводиться помилковий вихід, або це може бути виражено як нестабільний стан високої енергії в мережі.

На відміну від домінуючого використання зворотного розповсюдження в контрольованих методах, неконтрольоване навчання також використовує інші методи, зокрема: правило навчання Хопфілда, правило навчання Больцмана, контрастну дивергенцію, сплячий сон, варіаційний висновок, максимальну правдоподібність, максимальну апостеріорність, вибірку Гіббса та помилки реконструкції зворотного поширення або перепараметризації прихованого стану.

Конкретні мережі. Тут ми висвітлюємо деякі характеристики вибраних мереж. Подробиці кожного наведено в порівняльній таблиці нижче.

Мережа Хопфілда. Феромагнетизм надихнув мережі Хопфілда. Нейрон відповідає домену заліза з бінарними магнітними моментами Up і Down, а нейронні зв'язки відповідають впливу доменів один на одного. Симетричні з'єднання дозволяють формувати глобальну енергію. Під час висновку мережа оновлює кожен стан за допомогою стандартної функції кроку активації. Симетричні ваги та правильні енергетичні функції гарантують збіжність до стабільної моделі активації. Асиметричні ваги важко аналізувати. Мережі Хопфілда використовуються як адресна пам'ять (CAM).

Машина Больцмана. Це стохастичні мережі Хопфілда. Значення їхнього стану вибирається з цього PDF-файлу таким чином: припустимо, що двійковий нейрон спрацьовує з імовірністю Бернуллі p(1) = 1/3 і зупиняється з p(0) = 2/3. Вимірюється з нього, беручи РІВНОМІРНО розподілене випадкове число y і підключаючи його до оберненої інтегральної функції розподілу, яка в даному випадку є ступінчастою функцією з порогом 2/3. Обернена функція = { 0, якщо x <= 2/3, 1, якщо x > 2/3 }

Мережа глибокої віри. Ця мережа, представлена Хінтоном, є гібридом RBM і Sigmoid Belief Network. Верхні 2 шари - це RBM, а другий шар вниз утворює сигмоподібну мережу переконань. Його тренують за методом складеного RBM, а потім відкидають ваги розпізнавання під верхнім RBM. Станом на 2009 рік оптимальною глибиною є 3-4 шари [3].

Машина Гельмгольца. Це перші джерела натхнення для варіаційних автоматичних кодерів. Це 2 мережі, об'єднані в одну: прямі ваги забезпечують розпізнавання, а зворотні ваги реалізують уяву. Можливо, це перша мережа, яка зробила і те, і інше. Гельмгольц не працював у сфері машинного навчання, але він надихнув на думку про «механізм статистичного висновку, функцією якого є висновок про ймовірні причини сенсорного введення» (3). стохастичний бінарний нейрон видає ймовірність того, що його стан дорівнює 0 або 1. Вхідні дані зазвичай

не вважаються рівнем, але в режимі генерації машини Гельмгольца рівень даних отримує вхідні дані від середнього рівня, має для цього окремі ваги, тому це вважається шаром. Отже, ця мережа має 3 шари.

Варіаційний автокодувальник. Вони натхненні машинами Гельмгольца та поєднують мережу ймовірностей із нейронними мережами. Autoencoder — це 3-рівнева мережа CAM, де середній рівень має бути деяким внутрішнім представленням шаблонів введення. Нейронна мережа кодера — це розподіл ймовірностей qφ(z задано x), а мережа декодера — pθ(x задано z). Ваги називаються фі і тета, а не W і V, як у Гельмгольца, — косметична різниця. Ці 2 мережі тут можна повноцінно з’єднати, або використовувати іншу схему NN.

Hebbian Learning, ART, SOM. Класичним прикладом неконтрольованого навчання у вивченні нейронних мереж є принцип Дональда Гебба, тобто нейрони, які працюють разом, з’єднуються разом [4]. У геббівському навчанні зв'язок зміцнюється незалежно від помилки, але є виключно функцією збігу між потенціалами дії між двома нейронами [5]. Подібна версія, яка змінює синаптичні ваги, враховує час між потенціалами дії (пластичність, залежна від часу спайку або STDP). Вважається, що Hebbian Learning лежить в основі ряду когнітивних функцій, таких як розпізнавання образів і експериментальне навчання.

Серед моделей нейронних мереж, самоорганізуюча карта (SOM) і теорія адаптивного резонансу (ART) зазвичай використовуються в алгоритмах неконтрольованого навчання. SOM — це топографічна організація, у якій сусідні місця на карті представляють вхідні дані з подібними властивостями. Модель ART дозволяє змінювати кількість кластерів залежно від розміру проблеми та дозволяє користувачеві контролювати ступінь схожості між членами одних і тих же кластерів за допомогою визначеної користувачем константи, яка називається параметром пильності. Мережі ART використовуються для багатьох завдань розпізнавання образів, таких як автоматичне розпізнавання цілей і обробка сейсмічних сигналів [6].

Імовірнісні методи. Двома основними методами, які використовуються в неконтрольованому навчанні, є аналіз основних компонентів і кластерний аналіз. Кластерний аналіз використовується в неконтрольованому навчанні для групування або сегментування наборів даних зі спільними атрибутами з метою екстраполяції алгоритмічних зв'язків [7]. Кластерний аналіз — це розділ машинного навчання, який групує дані, які не були позначені, класифіковані чи категоризовані. Замість того, щоб реагувати на відгуки, кластерний аналіз визначає спільні риси в даних і реагує на наявність або відсутність таких спільних рис у кожній новій частині даних. Цей підхід допомагає виявити аномальні точки даних, які не підходять до жодної групи.

Основним застосуванням неконтрольованого навчання є оцінка щільності в статистиці [8], хоча неконтрольоване навчання охоплює багато інших областей, що включають узагальнення та пояснення характеристик даних. Його можна порівняти з навчанням під наглядом, сказавши, що в той час як навчання під наглядом має на меті вивести умовний розподіл ймовірностей, обумовлений міткою вхідних даних; неконтрольоване навчання має на меті зробити висновок про апріорний розподіл ймовірностей.

Метод моментів. Одним із статистичних підходів до неконтрольованого навчання є метод моментів. У методі моментів невідомі параметри (що представляють інтерес) у моделі пов'язані з моментами однієї або кількох випадкових змінних, і, таким чином, ці невідомі параметри можуть бути оцінені з урахуванням моментів. Моменти зазвичай оцінюються за зразками емпіричним шляхом. Основними моментами є моменти першого та другого порядку. Для випадкового вектора моментом першого порядку є вектор середнього значення, а моментом другого порядку є коваріаційна матриця (коли середнє дорівнює нулю). Моменти вищих порядків зазвичай представляють за допомогою тензорів, які є узагальненням матриць вищих порядків як багатовимірних масивів.

Зокрема, показано ефективність методу моментів у навчанні параметрів моделей латентних змінних. Моделі латентних змінних — це статистичні моделі,

де крім спостережуваних змінних також існує набір латентних змінних, які не спостерігаються. Дуже практичним прикладом моделей латентних змінних у машинному навчанні є тематичне моделювання, яке є статистичною моделлю для генерації слів (спостережуваних змінних) у документі на основі теми (латентної змінної) документа. У тематичному моделюванні слова в документі генеруються відповідно до різних статистичних параметрів, коли тема документа змінюється. Показано, що метод моментів (методи тензорної декомпозиції) послідовно відновлює параметри великого класу моделей латентної змінної за деяких припущень [11].

Алгоритм очікування-максимізації (EM) також є одним із найбільш практичних методів вивчення моделей латентних змінних. Однак він може застрягти в локальних оптимумах, і немає гарантії, що алгоритм зійдеться до справжніх невідомих параметрів моделі. Навпаки, для методу моментів глобальна збіжність гарантується за певних умов.

2.1.3 Навчання з підкріпленням

Навчання з підкріпленням (RL) — це область машинного навчання, яка пов’язана з тим, як інтелектуальні агенти повинні виконувати дії в середовищі, щоб максимізувати поняття сукупної винагороди. Навчання з підкріпленням є однією з трьох основних парадигм машинного навчання, поряд із навчанням під наглядом і навчанням без нагляду.

Навчання з підкріпленням відрізняється від навчання під наглядом тим, що не потрібно подавати позначені пари входу/виходу, а також не потрібно явно виправляти субоптимальні дії. Натомість увага зосереджена на пошуку балансу між дослідженням (незвіданих територій) і експлуатацією (сучасних знань) [1].

Середовище зазвичай описується у формі марковського процесу прийняття рішень (MDP), оскільки багато алгоритмів навчання з підкріпленням для цього контексту використовують методи динамічного програмування [2]. Основна

відмінність між класичними методами динамічного програмування та алгоритмами навчання з підкріпленням полягає в тому, що останні не передбачають знання точної математичної моделі MDP, і вони націлені на великі MDP, де точні методи стають неможливими.

Завдяки своїй загальності навчання з підкріпленням вивчається в багатьох дисциплінах, таких як теорія ігор, теорія управління, дослідження операцій, теорія інформації, оптимізація на основі моделювання, мультиагентні системи, ройовий інтелект і статистика. У літературі з дослідження операцій і контролю навчання з підкріпленням називається наближеним динамічним програмуванням або нейродинамічним програмуванням. Проблеми навчання з підкріпленням також вивчалися в теорії оптимального керування, яка пов’язана здебільшого з існуванням і характеристикою оптимальних рішень, а також алгоритмами для їх точного обчислення, і менше з навчанням або апроксимацією, особливо за відсутності математичну модель середовища. В економіці та теорії ігор навчання з підкріпленням може бути використане для пояснення того, як може виникнути рівновага в умовах обмеженої раціональності.

Розвідка. Компроміс між розвідкою та експлуатацією був найбільш ретельно вивчений через проблему багаторукого бандита та для MDP кінцевого простору станів у Burnetas and Katehakis (1997) [8].

Навчання з підкріпленням вимагає розумних механізмів дослідження; випадковий вибір дій без посилання на оцінений розподіл ймовірностей показує низьку продуктивність. Випадок (малих) кінцевих MDP відносно добре вивчений. Однак через відсутність алгоритмів, які добре масштабуються з кількістю станів (або масштабуються до проблем із нескінченними просторами станів), прості методи дослідження є найбільш практичними.

Одним із таких методів є e-greedy, де $0<e<1$ є параметром, що контролює кількість розвідки та експлуатації. З імовірністю 1-e обирається експлуатація, і агент обирає дію, яка, на його думку, має найкращий довгостроковий ефект (зв’язки між діями розриваються рівномірно випадковим чином). Як альтернатива,

з ймовірністю e вибирається дослідження, а дія вибирається рівномірно навмання. e зазвичай є фіксованим параметром, але його можна регулювати відповідно до розкладу (що змушує агента поступово досліджувати), або адаптивно на основі евристики [9].

Алгоритми контролю навчання. Навіть якщо питання розвідки знехтувати і навіть якщо стан можна було спостерігати (припускається далі), залишається проблема використання минулого досвіду, щоб з’ясувати, які дії призводять до вищих сукупних винагород.

Груба сила. Підхід грубої сили передбачає два кроки:

- Для кожної можливої політики зразок повертається під час її дотримання
- Виберіть поліс із найбільшою очікуваною прибутковістю

Проблема полягає в тому, що кількість політик може бути великою або навіть нескінченною. Інший полягає в тому, що дисперсія доходів може бути великою, що вимагає багатьох вибірок для точної оцінки прибутку кожного поліса.

Ці проблеми можна вирішити, якщо ми припустимо певну структуру та дозволимо зразкам, створеним з однієї політики, впливати на оцінки, зроблені для інших. Двома основними підходами для досягнення цього є оцінка функції вартості та прямий пошук політики.

2.2 Вибір мови програмування

Python - мова програмування загального призначення високого рівня [2], з великим пакетом стандартних бібліотек [3], керівною ідеєю якої є читабельність і ясність вихідного коду. Його синтаксис чіткий і лаконічний [4, 5].

Python підтримує різні парадигми програмування: об'єктно-орієнтоване, імперативне і меншою мірою функціональне. Він має повністю динамічну систему типів і автоматичне керування пам’яттю, подібно до Perl, Ruby, Scheme або Tcl. Як і інші динамічні мови, вона часто використовується як мова сценаріїв. Інтерпретатори Python доступні для багатьох операційних систем.

Python розробляється як проект з відкритим вихідним кодом, яким керує Python Software Foundation, яка є некомерційною організацією. Стандартною реалізацією мови є CPython (написаний на C), але є й інші, такі як Jython (написаний на Java), CLPython, написаний на Common Lisp, IronPython (для .NET) і PyPy

Розвиток мови. Python був створений на початку 1990-х років Гвідо ван Россумом - як наступник мови ABC, створеної в Centrum voor Wiskunde en Informatica (CWI - Центр математики та інформатики в Амстердамі). Ван Россум є основним розробником Python, хоча багато з цього було зроблено іншими людьми. Через ключову роль ван Россума у прийнятті важливих дизайнерських рішень його часто називали «доброзичливим диктатором на все життя» (BDFL).

Назва мови походить не від тварини, а від комедійного серіалу BBC «Летючий цирк Монті Пайтона» 1970-х років. Дизайнер, будучи фанатом серії та шукаючи коротку, унікальну та дещо загадкову назву, знайшов цю назву чудовою [6].

Версія 1.2 була останньою, випущеною CWI. З 1995 року Ван Россум продовжував працювати над Python у Корпорації національних дослідницьких ініціатив (CNRI) у Рестоні, штат Вірджинія, де він випустив кілька версій Python, аж до 1.6 включно. У 2000 році ван Россум і команда розробників ядра Python перейшли на BeOpen.com, щоб створити команду BeOpen PythonLabs. Першою і єдиною версією, випущеною BeOpen.com, був Python 2.0.

Після випуску версії 1.6 і ван Россума, який залишив CNRI, щоб почати комерційну розробку, було визнано дуже бажаним, щоб Python використовувався з програмним забезпеченням під ліцензією GPL. CNRI та Фонд вільного програмного забезпечення (FSF) доклали спільних зусиль, щоб відповідно змінити ліцензію Python. Версія 1.6.1 була по суті ідентичною версії 1.6, за винятком кількох незначних змін і ліцензії, яка зробила пізніші версії сумісними з GPL. Python 2.1 походить від 1.6.1 і 2.0.

Після випуску Python 2.0 компанією BeOpen.com Гвідо ван Россум та інші розробники PythonLabs перейшли до Digital Creations. Уся інтелектуальна

власність, додана з того часу, починаючи з Python 2.1 (включаючи альфа- та бета-версії), належить Python Software Foundation (PSF), некомерційній організації, створеній за моделлю Apache Software Foundation.

Філософія Python. Python реалізує кілька парадигм одночасно. Подібно до C++ і на відміну від Smalltalk, він не передбачає застосування одного стилю програмування, дозволяючи використовувати різні. У Python можливі об'єктно-орієнтоване програмування, структурне програмування та функціональне програмування. Типи перевіряються динамічно, а для керування пам'яттю використовується збір сміття.

Хоча його популяризація підкреслює його відмінності від Perl, Python багато в чому схожий. Однак розробники Python відмовилися від складного синтаксису Perl на користь більш економного та, на їхню думку, більш читабельного синтаксису. Хоча схожий на Perl, Python іноді класифікують як мову сценаріїв, він використовується для створення великих проектів, таких як сервер додатків Zope, система обміну файлами Mojo Nation або навіть програмне забезпечення класу ERP - Odoo.

Типи та структури даних. У Python значення, а не змінні, мають тип, тому Python є динамічно типізованою мовою, як і Lisp, на відміну від Java. На відміну від багатьох мов, значення передаються не за значенням і не за посиланням, а за допомогою присвоєння [7].

Порівняно з іншими мовами програмування, Python досить жорстко типізований. Він не є ні таким дозвільним, як Perl, ні таким обмежувальним, як OCaml. Правила синтаксису Python дозволяють виражати поняття без написання додаткового коду. Для числових типів визначено автоматичне перетворення, тому можна, наприклад, помножити комплексне число на довге ціле без приведення. Однак, на відміну від Perl, немає автоматичного перетворення між рядками та числами, наприклад; число не є дійсним аргументом для рядкової операції.

Python пропонує широкий спектр основних типів даних, включаючи числові типи (ціле, з плаваючою точкою, комплекс) і колекції.

Колекції. Деякі з перерахованих вище типів є колекціями.

Списки, кортежі та рядки є послідовностями, і тому забезпечують ряд загальних операцій (наприклад, ви можете перебирати рядки так само, як і елементи списку, або вказувати елементи за допомогою індексів). Списки - це масиви зі змінною кількістю елементів (з можливістю їх видалення, додавання і заміни), а кортежі - масиви з фіксованою кількістю елементів (без можливості видалення, додавання і заміни).

Інші типи — це невпорядковані колекції: словники (тип dict) — відомі в інших мовах як карти або асоціативні масиви, і набори, представлені двома типами: змінний набір і незмінний заморожений набір. Словникові ключі та елементи набору повинні бути т.зв. об’єкти, які можна хешувати (мають метод __hash__()) – що зазвичай означає, що вони мають бути незмінними (не змінними); наприклад, ключ словника не може бути списком або змінним набором (типу набору) - він може бути кортежем або незмінним набором (типу frozenset), доки він містить лише незмінні елементи.

Ython підтримує загальний набір рядкових операцій. Рядки Python незмінні. Будь-яка операція, яка змінить вміст рядка (наприклад, перетворення малих літер на великі), поверне новий рядок, залишивши вихідний рядок без змін.

Слід додати, що основні колекції в стандартній реалізації Python на C оптимізовані для швидкості пошуку, сортування тощо.

Все є предметом. Система типів Python тісно пов'язана з системою класів. Хоча вбудовані типи насправді не є класами, клас може успадковувати будь-який тип. Таким чином, ви можете успадковувати класи від рядків або словників і навіть від цілих чисел. Крім того, можливе множинне успадкування.

Мова дозволяє глибоко досліджувати типи та класи. Типи можна читати та порівнювати - як і в Smalltalk, типи (опис типів) також є типом. Атрибути об'єктів можна отримати як словник.

У Python немає інкапсуляції, як у C++ або Java, але є механізми для досягнення подібного ефекту. У той же час Python значно полегшує самоаналіз

об’єктів, тому правильне використання атрибутів об’єкта залишається за програмістом.

Крім того, кожна функція, клас і модуль можуть бути задокументовані у вихідному коді. Хоча він не має розширених функцій, подібних до javadoc, він доступний під час виконання, а отже, в інтерактивному режимі.

Емуляція типів, перевантаження операторів, виклик як функцій. Python дозволяє налаштовувати властивості даного класу в широкому діапазоні. Реалізуючи відповідні методи, ви можете змусити об’єкти даного класу поводитися як колекції, числа і навіть функції.

Синтаксис. Читання коду займає в рази більше часу, ніж його написання, а читабельну програму легше зрозуміти та розробити. Python розроблено для забезпечення максимально можливої читабельності вихідного коду. Він має простий макет тексту, використовує відступи або англійські слова там, де в інших мовах використовується пунктуація, і має набагато менше синтаксичних конструкцій, ніж багато структурованих мов, таких як C, Perl і Pascal.

Для зручності читання в Python існує лише два типи циклів: for, який виконує ітерацію по елементах списку (як у Perl foreach), і while, який повторюється, доки не буде виконано логічну умову. Python не має стилю C для синтаксису, do...while або стилю Perl до синтаксису, хоча, звичайно, ви можете отримати їхні складні еквіваленти.

Починаючи з версії 2.5, Python має умовний оператор, аналогічний умові ? value1 : value2 відоме з C і Java. Синтаксис: значення1 якщо умова інше значення2.

Структура за відступом. Відмінною рисою Python від інших мов є використання відступів для ізоляції блоків коду. Це унікальна функція серед широко використовуваних мов програмування, оскільки її першими помічають програмісти, які не використовують Python.

У мовах програмування, які отримують блочну структуру з Algol (не обов’язково безпосередньо) - наприклад, Pascal, C або Perl - блоки коду позначаються фігурними дужками або ключовими словами (C і Perl

використовують {}, Pascal використовують початок і кінець). Однак у всіх цих мовах програмісти традиційно використовують відступи для виділення блоків у навколишньому коді.

Стандартна бібліотека. Python має велику стандартну бібліотеку, яка дозволяє використовувати його для багатьох завдань. Автори мови дотримуються політики т. зв Батареї включені, тобто надається якомога більше інструментів разом із інсталяційним пакетом. Модулі стандартної бібліотеки можна доповнити модулями, написаними на C або Python. Стандартна бібліотека особливо добре підходить для розробки веб-додатків, оскільки підтримує велику кількість стандартних форматів і протоколів (наприклад, MIME, HTTP). Також включені модулі для створення GUI (на основі Tcl/Tk), обробки регулярних виразів, навіть простого веб-сервера з підтримкою CGI.

Більшість стандартної бібліотеки доступна на всіх платформах, тому навіть великі програми часто можуть працювати без змін на Unix, Windows, Macintosh та інших платформах. На відміну, наприклад, від Java, набір доступних функцій не обмежується частиною, спільною для різних платформ; наприклад, os.fork() доступний на Unix, але не на Windows [9].

Стандарти для «зовнішніх» бібліотек. Як і у випадку з іншими мовами, було розроблено низку стандартів для створення допоміжних API, наприклад, драйверів реляційної бази даних. Завдяки повністю динамічній системі типів немає необхідності включати «базовий інтерфейс» у стандартну бібліотеку, як це має місце, наприклад, у JDBC. Розробник зовнішньої бібліотеки просто повинен переконатися, що модулі, функції та класи, які він створює, мають відповідні атрибути.

Інші особливості. Інтерпретатор Python також має інтерактивний режим, у якому вирази можна вводити з терміналу з миттєвим результатом. За припущеннями творців Python, це робиться для полегшення навчання програмуванню, оскільки дозволяє випробувати фрагменти коду з негайним ефектом. Стандартна оболонка, однак, не дуже зручна і не має багатьох функцій

(наприклад, відсутність завершення TAB) - IPython, який є частиною пакету SciPy, не має цих недоліків.

Разом з Python бібліотека unittest також поширюється для модульних тестів, що дозволяє створювати вичерпні тести коректності створеного програмного забезпечення [10].

2.3 Вибір додаткового інструментарію

Scikit-image (раніше scikits.image) — це бібліотека зображень з відкритим кодом для мови програмування Python [12]. Він включає алгоритми для сегментації, геометричної трансформації, маніпулювання простором кольорів, аналізу, фільтрації, морфології, виявлення ознак тощо. Він призначений для взаємодії з чисельними та науковими бібліотеками Python NumPy та SciPy.

Проект Scikit-image розпочався як scikits.image, створений Стефаном ван дер Вальтом. Його назва походить від ідеї, що це «SciKit» (набір інструментів SciPy), розроблений і розповсюджений окремо сторонніми розширеннями SciPy [4]. Оригінальна кодова база пізніше була значно переписана іншими розробниками. У листопаді 2012 р. у різних наукових статтях наукові знання та наукові знання описувалися як «ретельно переглянуті та популяризовані» [5]. Scikit-image також дуже активний у Google Summer of Code [6].

Scikit-image в основному написаний на Python, а деякі основні алгоритми написані на Cython, який має кращу продуктивність.

Keras — це бібліотека програмного забезпечення з відкритим кодом, яка надає інтерфейс Python для штучних нейронних мереж. Keras діє як інтерфейс для бібліотеки TensorFlow.

До версії 2.3 Keras підтримував різноманітні серверні програми, включаючи TensorFlow, Microsoft Cognitive Toolkit, Theano та PlaidML [1-3]. Починаючи з версії 2.4, підтримується лише TensorFlow. Він розроблений для швидкого експериментування з глибокими нейронними мережами з акцентом на зручності

використання, модульності та масштабованості. Вона була розроблена в рамках дослідницької роботи проекту ONEIROS (Open Neuroelectronic Intelligent Robot Operating System) [4], головним автором і супроводжувачем якого є Франсуа Шолле, інженер Google. Шолле також є автором моделі глибокої нейронної мережі Xception [5].

Деталі. Keras включає кілька реалізацій загальних будівельних блоків нейронної мережі, таких як шари, цілі, функції активації, оптимізатори та різні інструменти, які допомагають із зображеннями та текстовими даними для спрощення кодування, необхідного для написання глибокого коду нейронної мережі. Код розміщено на GitHub, а форуми підтримки спільноти містять сторінку проблем GitHub і канал Slack.

Окрім стандартних нейронних мереж, Keras також підтримує згорточні нейронні мережі та рекурентні нейронні мережі. Він підтримує інші загальні рівні корисності, такі як відключення, нормалізація пакетів і агрегація [6].

Keras дозволяє користувачам створювати глибокі моделі на смартфонах (iOS і Android), в Інтернеті або на віртуальній машині Java [2]. Це також дозволяє розподілене навчання моделей глибокого навчання на кластерах графічних процесорів (GPU) і тензорних процесорів (TPU) [7].

TensorFlow — це безкоштовна бібліотека програмного забезпечення з відкритим кодом для машинного навчання та штучного інтелекту. Його можна використовувати для багатьох завдань, але він зосереджений на навчанні та висновках глибоких нейронних мереж [5, 6].

TensorFlow розроблено командою Google Brain для наукового та виробничого використання в Google [7-9]. Початкова версія була випущена в 2015 році під ліцензією Apache 2.0 [1, 10]. У вересні 2019 року Google випустив оновлену версію TensorFlow під назвою TensorFlow 2.0 [11].

TensorFlow доступний кількома мовами програмування, включаючи Python, JavaScript, C++ і Java [12]. Ця гнучкість дозволяє використовувати різні програми в різних секторах.

Самодиференціація. Самодиференціювання — це процес автоматичного обчислення вектора градієнта для кожного параметра моделі. За допомогою цієї функції TensorFlow може автоматично обчислювати градієнт параметрів у моделі, що дуже корисно для таких алгоритмів, як зворотне поширення, які потребують градієнтів для оптимізації продуктивності [33]. Для цього фреймворк повинен стежити за порядком операцій, які виконуються над вхідними тензорами в моделі, а потім обчислювати градієнти щодо відповідних параметрів [33].

Наполегливе виконання. TensorFlow включає режим «своєчасного виконання», що означає, що операції оцінюються негайно, а не додаються до обчислювального графіка для виконання пізніше [33]. Може поступово перевіряти код, що виконується, за допомогою відладчика, оскільки дані заповнюються в кожному рядку коду, а не пізніше в графі обчислень [34]. Завдяки своїй покроковій прозорості цю парадигму виконання вважають легшою для налагодження [34].

Розподіл. TensorFlow надає API для розподілу обчислень між кількома пристроями з різними стратегіями розподілу [35]. Ці розподілені обчислення часто прискорюють навчання та оцінку моделей TensorFlow і є звичайною практикою в ШІ [35, 36].

Втрата. Для навчання та оцінки моделей TensorFlow надає набір функцій втрат (також відомих як функції вартості) [34]. Деякі популярні приклади включають середню квадратичну помилку (MSE) і двійкову крос-ентропію (BCE) [37]. Ці функції втрат обчислюють «помилку» або «різницю» між виходом моделі та очікуваним виходом (ширше, різницею між двома тензорами). Для різних наборів даних і моделей різні втрати використовуються для визначення пріоритетів певних аспектів продуктивності.

Індекс. Щоб оцінити продуктивність моделей машинного навчання, TensorFlow надає API-доступ до загальних показників. Приклади включають різноманітні метрики точності (бінарні, категоричні, розріджені категоричні), а також інші метрики, такі як точність, відкликання та перетин через об’єднання (IoU) [38].

TF.nn. TensorFlow.nn — це модуль для виконання примітивних операцій нейронної мережі над моделями [39]. Деякі з цих операцій включають варіації згорток (1/2/3D, Atrous, по глибині), функції активації (Softmax, RELU, GELU, Sigmoid тощо) та їх варіації, а також інші операції Tensor (max-pooling, bias-add тощо) [39].

Оптимізатори. TensorFlow пропонує набір оптимізаторів для навчання нейронних мереж, включаючи ADAM, ADAGRAD і Stochastic Gradient Descent (SGD) [40]. Під час навчання моделі різні оптимізатори пропонують різні режими налаштування параметрів, що часто впливає на конвергенцію та продуктивність моделі.

Keras — це бібліотека з відкритим кодом для машинного навчання та нейронних мереж, написана на Python. Він розроблений як інтерфейс вищого рівня абстракції для інших подібних бібліотек нижчого рівня та підтримує бібліотеки TensorFlow, Microsoft Cognitive Toolkit (CNTK) і Theano [1] як серверні. Розроблений для швидкого створення прототипів глибоких нейронних мереж, він зосереджений на простоті використання, модульності та розширюваності. Його було розроблено в рамках дослідницького проекту ONEIROS [2], а його провідним автором є Франсуа Шолле з Google [3].

У 2017 році команда TensorFlow вирішила офіційно підтримати Keras [4]. Шолле пояснив, що Keras був розроблений як інтерфейс, а не як окрема бібліотека. Він пропонує серію модулів, які дозволяють розробляти глибокі нейронні мережі незалежно від використовуваної серверної частини, за допомогою загальної та інтуїтивно зрозумілої мови[4]. Microsoft додала бекенд до CNTK, починаючи з CNTK версії 2.0 [5, 6].

Висновки до розділу 2

Другий розділ цілком присвячено аналітичному огляду інструментальних засобів розробки майбутнього програмного продукту.

В ході розділу проаналізовано основні особливості мови програмування Python та таких додаткових інструментів як Tensorflow та Keras.

Уся отримана інформація є основою для розуміння порядку роботи з технологіями, які в подальшому будуть використовуватися для розробки програмного продукту.

3 МОДЕЛЮВАННЯ ТА ДОСЛІДЖЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Проектування внутрішньої будови системи

Уніфікована мова моделювання (UML) - напівформальна мова, що використовується для моделювання різних типів систем, створена Грейді Бучем, Джеймсом Рамбо та Іваром Якобсоном, зараз розробляється Object Management Group [1].

Він використовується для моделювання області проблеми (опис-моделювання фрагмента існуючої реальності - наприклад, моделювання того, що робить відділ у компанії) - коли використовується для аналізу та моделювання реальності, яку ще належить створити - в основному створює моделі систем IT. UML здебільшого використовується з його графічним представленням – його елементам призначаються відповідні символи, пов’язані один з одним на діаграмах.

UML офіційно визначено Object Management Group (OMG) у т.зв Метамодель UML – Meta-Object Facility (MOF). Подібно до інших специфікацій, заснованих на Meta-Object Facility, метамодель UML і моделі UML можуть бути серіалізовані (написані) мовою XML Metadata Interchange (XMI) на основі стандарту XML. Незважаючи на те, що UML було розроблено для визначення, візуалізації, конструювання та документування програмно-інтенсивних систем, він не обмежується програмним моделюванням. UML також використовується для моделювання бізнес-процесів, системної інженерії та представлення організаційних структур. Мова моделювання систем (SysML) — це мова моделювання для певних тем системної інженерії, визначена як профіль UML 2.0. UML також може використовувати мову обмежень об’єктів (OCL), спочатку розроблену IBM, для розробки формальних обмежень [41].

Методи. UML не є методом як таким, але був розроблений для сумісності з провідними методами розробки об’єктно-орієнтованого програмного забезпечення (наприклад, OMT, Booch, Objectory). У міру розвитку UML деякі з цих методів було

оновлено для використання нової нотації (наприклад, OMT). Нові методи також були розроблені на основі існуючого UML. Найвідомішим є Rational Unified Process (RUP). Існує багато інших методів на основі UML, таких як метод абстракції, метод розробки динамічних систем та інші, призначені для надання більш конкретних рішень або досягнення інших цілей.

Діаграми. Для UML версії 2.2 є 14 основних діаграм і 3 абстрактні діаграми (структури, поведінка та взаємодії). На жаль, є деякі неоднозначності щодо польського перекладу використаних діаграм, наприклад, діаграма часу перекладається як діаграма часу, часових залежностей, планування, умов часу або діаграма часового ходу.

Діаграма класів - статична структурна діаграма в UML, що показує структуру системи в об'єктних моделях шляхом ілюстрації структури класів і зв'язків між ними.

Діаграма класів показує класи (типи) об’єктів у програмі, на відміну від діаграми об’єктів, яка показує лише екземпляри об’єктів та їхні залежності, що існують у певний момент часу [42].

Діаграма класів показує певну частину структури системи. Діаграми класів використовуються для моделювання статичних аспектів перспективи проектування. Це тісно пов’язане з моделюванням словника системи, співпраці чи схем. Діаграми класів дозволяють формалізувати дані та специфікації методів. Вони також можуть діяти як графічний засіб для відображення деталей реалізації класу.

Класи. Клас у UML-моделі об’єктно-орієнтованої програми представлений прямокутником із назвою класу всередині нього. Відокремлена частина прямокутника під назвою класу може містити атрибути класу, тобто методи (функції), властивості (properties) або поля (змінні). Кожен атрибут відображається принаймні як ім’я, за бажанням також із типом, значенням та іншими характеристиками.

Методи класу можуть бути в окремій частині прямокутника. Кожен метод відображається принаймні як ім’я, а також із його параметрами та типом повернення.

Атрибути (змінні та властивості) і методи також можуть бути позначені видимістю (значення їх імен) наступним чином [43]:

– "+" для public – публічний, глобальний доступ;

– "#" для protected – захищений, доступ для похідних класів (в результаті узагальнення);

– "–" для private – приватний, доступний лише в межах класу (зі статичним атрибутом) або об’єкта (зі звичайним атрибутом);

– "~" для пакета – пакет, доступний у даному пакеті, проекті.

Відносини (асоціації). Залежність. Залежність – найслабший семантичний зв’язок між класами, коли один клас використовує інший. На діаграмі класів він позначений пунктирною лінією, що закінчується стрілкою, що вказує напрямок залежності.

Асоціація. Асоціація вказує на сильніший зв’язок між об’єктами заданих класів (наприклад, компанія наймає працівників). На схемі асоціація позначена лінією, яка може закінчуватися стрілкою з відкритою головкою (вказує напрямок асоціації класів). Назви ознак (наприклад, зайнятий, наймаючий) разом із множинністю розміщуються в цільовій точці асоціації. Назва асоціації вказується посередині (наприклад, наймає).

Узагальнення. Узагальнення або успадкування - зв'язок, що описує класи та підкласи (загальні класи та специфічні класи або батьки та діти). На діаграмі узагальнення позначається відкритим трикутником, що представляє стрілку (вказує від похідного класу до базового класу).

Агрегація. Агрегація являє собою відношення ціле-частина, тобто велике ціле розбивається на елементи. Це означає, що часткові елементи можуть належати до більшого цілого, але вони також можуть існувати без нього (наприклад, колеса

та автомобіль). На діаграмі агрегація представлена лінією, яка закінчується порожнім ромбом.

Композиція. Композиція є більш сильною формою агрегації. У відношенні композиції частини належать тільки одному цілому, а час їх життя загальний - разом з цілим руйнуються і частини. Здебільшого це договірне питання, залежно від даної системи. Наприклад, автомобільний двигун може існувати окремо в одній системі (проста агрегація), а в іншій він може бути знищений разом з автомобілем (композиція). На схемі композиція зображена лінією, що закінчується заповненим ромбом.

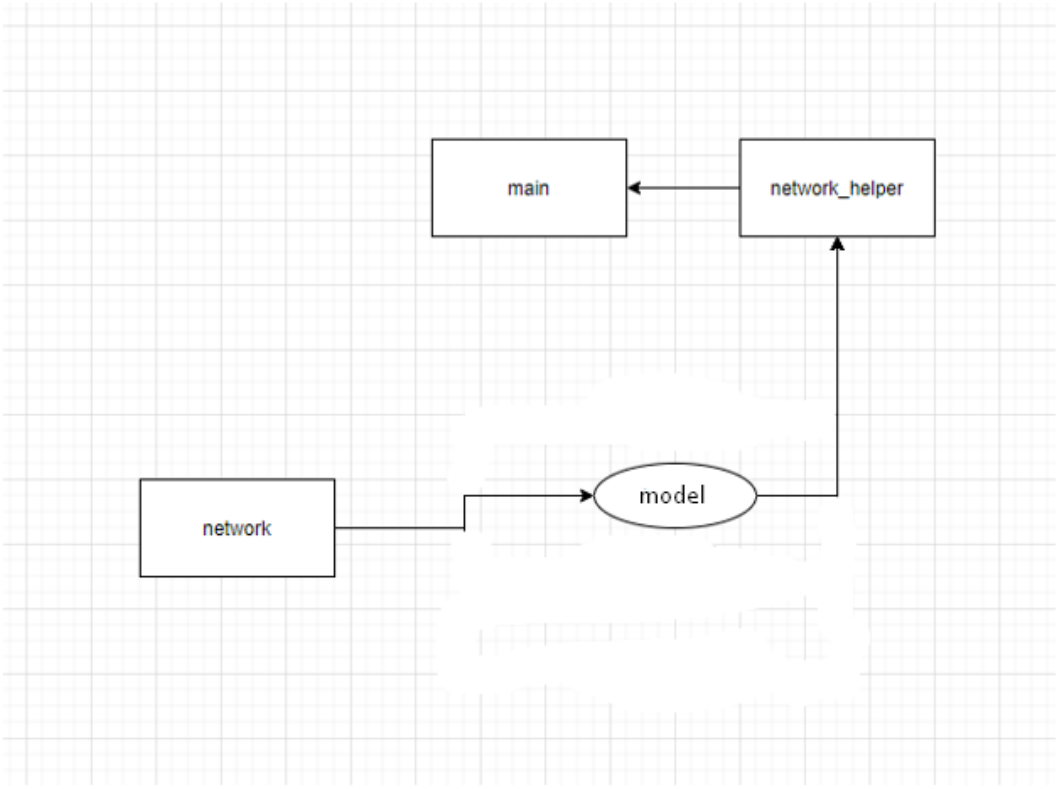

Рисунок 3.1 — Діаграма класів

3.2 Аналіз варіантів діяльності

Діаграма варіантів використання - графічне представлення варіантів використання, акторів і зв'язків між ними, що виникають у даній предметній області.

Діаграма варіантів використання в UML використовується для моделювання функціональності системи. Зазвичай він створюється на ранніх етапах моделювання. Ця діаграма є лише оглядом можливих дій у системі, деталі їх перебігу моделюються за допомогою інших методів (наприклад, діаграм стану чи діяльності).

Діаграма варіантів використання показує послуги, які система надає акторам, але без вказівки на конкретні технічні рішення.

Цілі використання діаграми варіантів використання:

– визначення та документування вимог,

– можливість аналізу області застосування, предметної області,

– дозволяють розробити майбутній дизайн системи,

– це доступна та зрозуміла платформа для співпраці та спілкування розробників систем, інвесторів та власників,

– вони є свого роду угодою, контрактом між акціонерами щодо обсягу та функціональності майбутньої системи,

– вони є основою для перевірки функціонування системи на подальших етапах її життєвого циклу.

Понятійні категорії та графічне позначення. Діаграма варіантів використання складається з таких концептуальних категорій:

– випадки використання;

– актори;

– стосунки.

Випадок використання. Випадок використання – специфікація серії дій та їх варіантів, які система (або інша сутність) може виконувати через взаємодію з акторами цієї системи.

Таким чином, варіант використання - це комплексна діяльність, що виконується в спроектованій системі як наслідок певної діяльності актора. Обсяг даної діяльності визначається всіма взаємопов’язаними випадками використання.

Одиничний варіант використання є представником узгодженої одиниці функціональності, яку забезпечує система.

Ім'я прецеденту - це стисла команда для виконання функції в проектованій системі, найчастіше це імперативна фраза. Відповідно до стандарту UML, він представлений еліпсом з міткою всередині.

Актор. Актор – узгоджений набір ролей, які виконують користувачі варіантів використання під час взаємодії з цим варіантом використання.

Ми розрізняємо особистих і безособових акторів. Персональним актором може бути особа, команда, відділ, установа, організація, асоціація організацій або віртуальна організація. Імена особистих акторів часто покриваються назвами функцій, які вони виконують в організації, проекті чи підприємстві, або назвою посади, яку вони займають. З іншого боку, безособовим актором може бути зовнішня система (підсистеми, бази даних), пристрій або час.

Ім’я дійової особи — іменник (або називний термін) в однині. Визначаючи акторів, ми повинні пам'ятати, що вони відображають ролі цих об'єктів, а не окремих об'єктів реального світу.

Актор використовує один або кілька варіантів використання в системі, що проектується. Взаємодія акторів із варіантами використання складається з їх ініціювання, надання даних, отримання даних і використання функціональних можливостей, реалізованих варіантом використання.

Відношення - смисловий зв'язок між елементами моделі.

Кожен актор, який знаходиться на діаграмі варіантів використання, має бути безпосередньо пов’язаний принаймні з одним варіантом використання. Подібним чином кожен варіант використання повинен використовуватися принаймні одним актором (часто непрямими посиланнями).

Відповідно до стандарту UML можна виділити чотири типи сполук:

– асоціація;
– узагальнення;
– залежність;

– впровадження.

Назва сполуки не включена в діаграму варіантів використання. На малюнку вище показано, що бронювання поїздки залежить від клієнта. На даному етапі також не уточнюється, як здійснюються перелічені дії та які системні ресурси вони повинні будуть використовувати.

Спрямована асоціація. Спрямована асоціація - спрямована асоціація успадковує всі ознаки асоціації, але додатково вказує напрямок навігації. Використовується, коли ми хочемо показати ініціатора взаємодії (наприклад, актор «Клієнт» є ініціатором випадку використання «Купити продукт»).

Узагальнення. Узагальнення полягає в тому, що певний варіант використання може бути окремим варіантом іншого, вже існуючого варіанту використання. Позначення являє собою суцільну лінію, що закінчується порожнім трикутником, що вказує від спеціалізованого до більш загального випадку використання. Це нагадує ідею підкласів з об'єктно-орієнтованого підходу. На практиці це може бути як зручним, так і ефективним способом перенесення типової поведінки, обмежень і припущень конкретних випадків використання в загальний варіант використання.

На рисунку 3.2 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

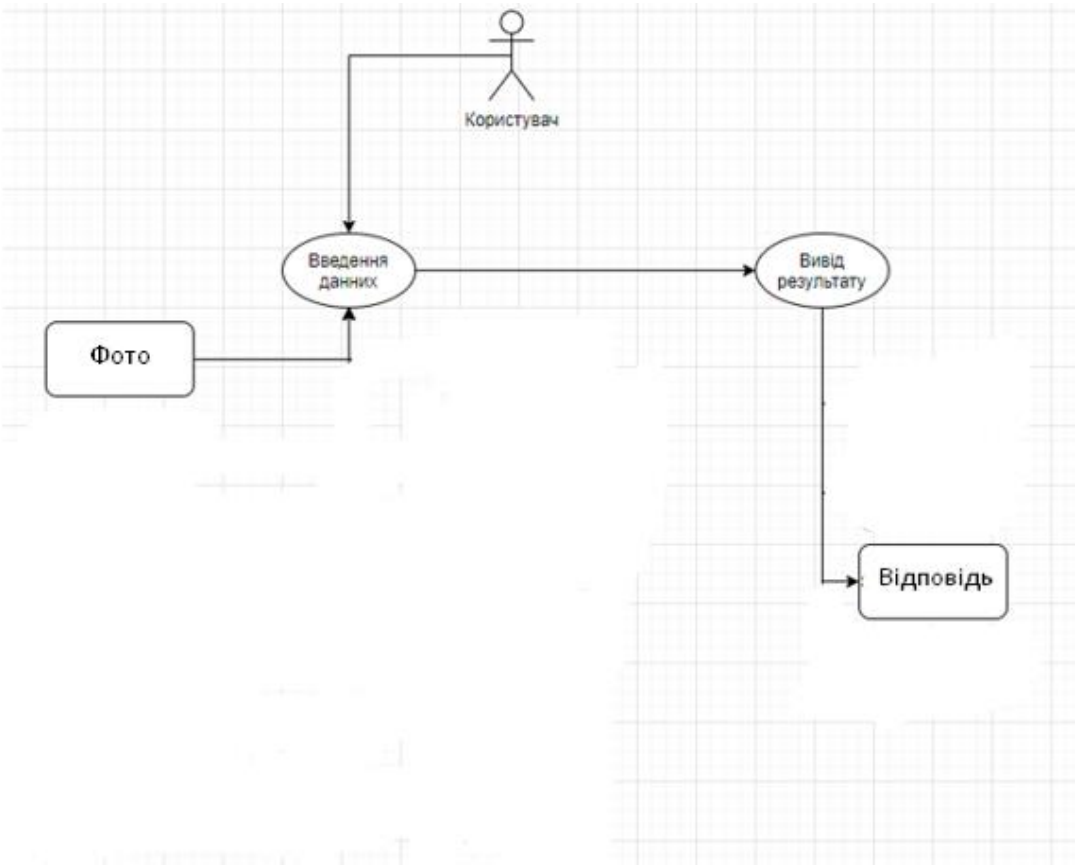

Рисунок 3.2 — Діаграма варіантів використання

3.3 Проектування архітектури нейронної мережі

В першу чергу при проектуванні архітектури нейронної мережі на базі готових шарів бібліотеки слід проаналізувати наявні варіанти шарів.

Для зменшення об’єму інформації буде викладено інформацію лише про використані шари.

3.3.1 Conv2D

Шар 2D згортки (наприклад, просторова згортка над зображеннями).

Цей рівень створює ядро згортки, яке згортається разом із вхідними даними шару для створення тензора виходів. Якщо use_bias має значення True, вектор зміщення створюється та додається до виходів. Нарешті, якщо активація не None, вона також застосовується до виходів.

Використовуючи цей шар як перший шар у моделі, надається аргумент ключового слова input_shape (кортеж цілих чисел або none, не включає вісь зразка), наприклад. input_shape=(128, 128, 3) для зображень 128x128 RGB у data_format="channels_last". Можна використовувати None, якщо розмір має змінні значення.

Аргументи:

– filters: ціле число, розмірність вихідного простору (тобто кількість вихідних фільтрів у згортці);

– kernel_size: ціле число або кортеж/список із 2 цілих чисел, що визначає висоту та ширину вікна двовимірної згортки. Може бути єдиним цілим числом для визначення того самого значення для всіх просторових вимірів.

– strides: ціле число або кортеж/список із 2 цілих чисел, що визначає кроки згортки по висоті та ширині. Може бути єдиним цілим числом для визначення того самого значення для всіх просторових вимірів. Вказування будь-якого значення кроку != 1 несумісне з визначенням будь-якого значення dilation_rate != 1;

– padding: один із «дійсний» або «те саме» (незалежно від регістру). "дійсний" означає відсутність доповнення. "same" призводить до доповнення нулями рівномірно ліворуч/праворуч або вгору/вниз від введення. Якщо padding="same" і strides=1, вивід має такий самий розмір, як і вхід;

– data_format: рядок, один із каналів_останній (за замовчуванням) або каналів_перший. Упорядкування розмірів у вхідних даних. channels_last відповідає входам із формою (batch_size, height, width, channels), тоді як channels_first відповідає входам із формою (batch_size, channels, height, width). За замовчуванням значення image_data_format міститься у вашому конфігураційному файлі Keras за адресою ~/.keras/keras.json. Якщо ви ніколи не встановлюєте його, то це буде канали_останній. Зауважте, що формат channels_first наразі не підтримується TensorFlow на ЦП;

– dilation_rate: ціле число або кортеж/список із 2 цілих чисел, що вказує швидкість розширення для використання для розширеної згортки. Може бути

єдиним цілим числом для визначення того самого значення для всіх просторових вимірів. Наразі вказівка будь-якого значення dilation_rate != 1 несумісна з вказівкою будь-якого значення кроку != 1;

– groups: позитивне ціле число, що вказує кількість груп, на які вхідний сигнал розділено вздовж осі каналу. Кожна група згортається окремо за допомогою фільтрів / фільтрів груп. Результатом є конкатенація всіх результатів груп уздовж осі каналу. Вхідні канали та фільтри мають бути поділені на групи;

– activation: функція активації для використання. Якщо ви нічого не вкажете, активація не застосовуватиметься;

– use_bias: логічне значення, чи використовує шар вектор зсуву;

– kernel_initializer: Ініціалізатор для матриці ваг ядра. За замовчуванням glorot_uniform;

– bias_initializer: Ініціалізатор для вектора зсуву. За замовчуванням "нулі";

– kernel_regularizer: функція регуляризатора, застосована до матриці ваг ядра;

– bias_regularizer: функція регуляризатора, застосована до вектора зсуву;

– activity_regularizer: функція регуляризатора, застосована до результату шару (його «активація»);

– kernel_constraint: функція обмежень, застосована до матриці ядра;

– bias_constraint: функція обмеження, застосована до вектора зміщення.

3.3.2 MaxPooling2D

Операція максимального об’єднання для двовимірних просторових даних.

Зменшує дискретизацію вхідного сигналу вздовж його просторових розмірів (висота та ширина), беручи максимальне значення у вікні введення (розміру, визначеному параметром pool_size) для кожного каналу вхідного сигналу. Вікно зсувається кроками вздовж кожного виміру.

Отриманий результат, якщо використовується «дійсний» параметр доповнення, має просторову форму (кількість рядків або стовпців): output_shape = math.floor((input_shape - pool_size) / strides) + 1 (коли input_shape >= pool_size)

Результуюча вихідна форма при використанні «того самого» параметра відступу: output_shape = math.floor((input_shape - 1) / strides) + 1.

Аргументи наприклад, для strides=(1, 1) і padding="valid":

– pool_size: ціле число або кортеж із 2 цілих чисел, розмір вікна, над яким можна взяти максимальний розмір. (2, 2) прийматиме максимальне значення у вікні об’єднання 2x2. Якщо вказано лише одне ціле число, для обох розмірів використовуватиметься однакова довжина вікна;

– strides: ціле число, кортеж із 2 цілих чисел або жодного. Великі кроки. Визначає, як далеко переміщується вікно об’єднання для кожного кроку об’єднання. Якщо немає, за замовчуванням використовуватиметься pool_size;

– padding: один із «дійсний» або «те саме» (незалежно від регістру). "дійсний" означає відсутність доповнення. "same" призводить до рівномірного відступу ліворуч/праворуч або вгору/вниз вхідних даних, щоб вихідні дані мали той самий розмір висоти/ширини, що й вхідні дані;

– data_format: рядок, один із каналів_останній (за замовчуванням) або каналів_перший. Упорядкування розмірів у вхідних даних. channels_last відповідає входам із формою (пакет, висота, ширина, канали), тоді як channels_first відповідає входам із формою (пакет, канали, висота, ширина). За замовчуванням значення image_data_format міститься у вашому конфігураційному файлі Keras за адресою ~/.keras/keras.json. Якщо ви ніколи не встановлюєте його, то це буде "channels_last".

3.3.3 Flatten

Згладжує введення. Не впливає на розмір партії.

Примітка. Якщо вхідні дані мають форму (пакет,) без осі ознак, тоді зведення додає додатковий розмір каналу, а вихідна форма має вигляд (пакет, 1).

Аргументи:

– data_format: рядок, один із каналів_останній (за замовчуванням) або каналів_перший. Упорядкування розмірів у вхідних даних. channels_last відповідає входам із формою (пакет, ..., канали), тоді як channels_first відповідає входам із формою (пакет, канали, ...). За замовчуванням значення image_data_format міститься у вашому конфігураційному файлі Keras за адресою ~/.keras/keras.json. Якщо ви ніколи не встановлюєте його, то це буде "channels_last".

3.3.4 Dense

Просто звичайний щільно зв’язаний шар NN.

Dense реалізує операцію: output = activation(dot(input, kernel) + bias), де activation — це поелементна функція активації, яка передається як аргумент активації, kernel — це матриця ваг, створена шаром, а bias — це вектор зміщення, створений за шаром (застосовно, лише якщо use_bias має значення True). Усе це атрибути Dense.

Примітка. Якщо вхід до рівня має ранг більше 2, тоді Dense обчислює скалярний добуток між входами та ядром уздовж останньої осі входів і осі 0 ядра (за допомогою tf.tensordot). Наприклад, якщо вхідні дані мають розміри (batch_size, d0, d1), ми створюємо ядро з формою (d1, одиниці), і ядро працює вздовж осі 2 вхідних даних на кожному субтензорі форми (1, 1 , d1) (існують такі підтензори batch_size * d0). Вихід у цьому випадку матиме форму (batch_size, d0, одиниці).

Крім того, атрибути шару не можуть бути змінені після того, як рівень було викликано один раз (крім атрибута trainable). Коли передається популярний kwarg input_shape, keras створить вхідний шар для вставки перед поточним шаром. Це можна розглядати як еквівалент явного визначення InputLayer.

Аргументи:

– units: додатне ціле число, розмірність вихідного простору;

– activation: функція активації для використання. Якщо ви нічого не вказуєте, активація не застосовується (тобто «лінійна» активація: a(x) = x);

– use_bias: логічне значення, чи використовує шар вектор зсуву;

– kernel_initializer: Ініціалізатор для матриці ваг ядра;

– bias_initializer: Ініціалізатор для вектора зміщення;

– kernel_regularizer: функція регуляризатора, застосована до матриці ваг ядра;

– bias_regularizer: функція регуляризатора, застосована до вектора зсуву;

– activity_regularizer: функція регуляризатора, застосована до виводу шару (його «активація»);

– kernel_constraint: функція обмежень, застосована до матриці ваг ядра;

– bias_constraint: функція обмеження, застосована до вектора зсуву.

3.3.6 Dropout

Рівень Dropout випадково встановлює одиниці введення на 0 із частотою швидкості на кожному кроці під час навчання, що допомагає запобігти переобладнанню. Вхідні дані, для яких не встановлено значення 0, масштабуються на 1/(1 - швидкість), щоб сума всіх вхідних даних не змінювалася.

Рівень Dropout застосовується лише тоді, коли для навчання встановлено значення True, щоб жодні значення не пропускалися під час висновку. Якщо використовується model.fit, навчання буде відповідно встановлено на True автоматично, а в інших контекстах ви можете встановити kwarg явно на True під час виклику шару.

(Це відрізняється від налаштування trainable=False для шару Dropout. trainable не впливає на поведінку шару, оскільки Dropout не має жодних змінних/ваг, які можна заморозити під час навчання.)

Аргументи:

- rate: плаває між 0 і 1. Частка вхідних одиниць, які потрібно викинути;

- noise_shape: одновимірний цілочисельний тензор, що представляє форму бінарної маски вилучення, яка буде помножена на вхідні дані. Наприклад, якщо ваші вхідні дані мають форму (batch_size, timesteps, features) і ви хочете, щоб маска випадання була однаковою для всіх часових кроків, ви можете використати noise_shape=(batch_size, 1, features);
- seed: ціле число Python для використання як випадкове початкове число.

3.3.7 UpSampling2D

Рівень підвищення дискретизації для 2D входів.

Повторює рядки та стовпці даних за розміром [0] і розміром [1] відповідно.

Аргументи:

- size: Int або кортеж із 2 цілих чисел. Коефіцієнти підвищення дискретизації для рядків і стовпців;
- data_format: рядок, один із каналів_останній (за замовчуванням) або каналів_перший. Упорядкування розмірів у вхідних даних. channels_last відповідає входам із формою (batch_size, height, width, channels), тоді як channels_first відповідає входам із формою (batch_size, channels, height, width). За замовчуванням значення image_data_format міститься у вашому конфігураційному файлі Keras за адресою ~/.keras/keras.json. Якщо ви ніколи не встановлюєте його, то це буде "channels_last";
- interpolation: рядок, один із «площа», «бікубічний», «білінійний», «гауссовий», «lanczos3», «lanczos5», «mitchellcubic», «nearest».

3.3.8 AveragePooling2D

Операція об’єднання середніх даних для просторових даних.

Зменшує дискретизацію вхідного сигналу вздовж його просторових розмірів (висота та ширина), беручи середнє значення для вхідного вікна (розміру, визначеного pool_size) для кожного каналу вхідного сигналу. Вікно зсувається кроками вздовж кожного виміру.

Отриманий результат під час використання «дійсного» параметра доповнення має форму (кількість рядків або стовпців): output_shape = math.floor((input_shape - pool_size) / strides) + 1 (коли input_shape >= pool_size)

Результуюча вихідна форма при використанні «того самого» параметра відступу: output_shape = math.floor((input_shape - 1) / strides) + 1

Аргументи:

- pool_size: ціле число або кортеж із 2 цілих чисел, коефіцієнти, за якими зменшується масштаб (вертикальний, горизонтальний). (2, 2) вдвічі зменшить вхідні дані в обох просторових вимірах. Якщо вказано лише одне ціле число, для обох розмірів використовуватиметься однакова довжина вікна;

- strides: ціле число, кортеж із 2 цілих чисел або жодного. Великі кроки. Якщо немає, за замовчуванням використовуватиметься pool_size;

- padding: один із «дійсний» або «те саме» (незалежно від регістру). "дійсний" означає відсутність доповнення. "same" призводить до рівномірного відступу вліво/вправо або вгору/вниз від вхідних даних, щоб вихідні дані мали той самий розмір висоти/ширини, що й вхідні дані;

- data_format: рядок, один із каналів_останній (за замовчуванням) або каналів_перший. Упорядкування розмірів у вхідних даних. channels_last відповідає входам із формою (пакет, висота, ширина, канали), тоді як channels_first відповідає входам із формою (пакет, канали, висота, ширина). За замовчуванням значення image_data_format міститься у вашому конфігураційному файлі Keras за адресою ~/.keras/keras.json. Якщо ви ніколи не встановлюєте його, то це буде "channels_last".

3.3.9 MobileNetV2

Створює екземпляр архітектури MobileNetV2.

MobileNetV2 дуже схожий на оригінальний MobileNet, за винятком того, що він використовує перевернуті залишкові блоки з функціями вузьких місць. Він має значно меншу кількість параметрів, ніж оригінальний MobileNet. MobileNets підтримує будь-який вхідний розмір, більший за 32 x 32, причому більші розміри зображень пропонують кращу продуктивність.

MobileNetV2: інвертовані залишки та лінійні вузькі місця (CVPR 2018)

Ця функція повертає модель класифікації зображень Keras, необов'язково завантажену з ваговими коефіцієнтами, попередньо навченими на ImageNet.

Примітка: кожна програма Keras передбачає певний вид попередньої обробки вхідних даних. Для MobileNetV2 викличте tf.keras.applications.mobilenet_v2.preprocess_input у ваших вхідних даних, перш ніж передати їх у модель. mobilenet_v2.preprocess_input масштабує вхідні пікселі від -1 до 1.

Аргументи:

- input_shape: необов'язковий кортеж форми, який потрібно вказати, якщо ви хочете використовувати модель із роздільною здатністю вхідного зображення, яка не є (224, 224, 3). Він повинен мати рівно 3 вхідних канали (224, 224, 3). Ви також можете пропустити цей параметр, якщо хочете вивести input_shape з input_tensor. Якщо ви вирішите включити як input_tensor, так і input_shape, тоді буде використано input_shape, якщо вони збігаються, якщо фігури не збігаються, ми видамо помилку. наприклад (160, 160, 3) буде одним дійсним значенням;

- alpha: Float, більше нуля, контролює ширину мережі. Це відоме як множник ширини в документі MobileNetV2, але назва збережена для узгодженості з програмами. Модель MobileNetV1 у Keras. Якщо альфа < 1,0, пропорційно зменшується кількість фільтрів у кожному шарі. Якщо альфа > 1,0, пропорційно збільшується кількість фільтрів у кожному шарі. Якщо alpha = 1,0, на кожному шарі використовується стандартна кількість фільтрів із паперу;

- include_top: логічне значення, чи включати повністю підключений рівень у верхній частині мережі. За замовчуванням значення True;

- weights: Рядок, одне з Немає (випадкова ініціалізація), 'imagenet' (попереднє навчання на ImageNet) або шлях до файлу ваг, який потрібно завантажити;

- input_tensor: необов’язковий тензор Keras (тобто вихід із layers.Input()) для використання як вхідних даних зображення для моделі;

- pooling: Рядок, необов'язковий режим об'єднання для вилучення функцій, коли include_top має значення False. None означає, що вихід моделі буде 4D-тензорним виходом останнього згорткового блоку. avg означає, що глобальне середнє об’єднання буде застосовано до виходу останнього згорткового блоку, і, отже, виходом моделі буде 2D-тензор. max означає, що буде застосовано глобальне максимальне об’єднання;

- classes: необов’язкова ціла кількість класів для класифікації зображень, яка вказується, лише якщо include_top має значення True і не вказано аргумент ваги;

- classifier_activation: str або callable. Функція активації для використання на «верхньому» шарі. Ігнорується, якщо include_top=True. Установіть classifier_activation=None, щоб повернути логіти «верхнього» шару. Під час завантаження попередньо підготовлених ваг, classifier_activation може мати значення None або "softmax";

- **kwargs: лише для зворотної сумісності.

3.3.10 Збір моделі

В ході аналізу варіантів архітектур нейронної мережі було проаналізовано 3 різні варіанти, серед яких середня точність виявлення забороненого контенту варіювалась в межах 85-90%.

Проте, одна з обраних архіетктур має середню точність на великих тестових відрізках 96%, чого більш ніж достатньо для виконання задачі покращення існуючих рішень.

Нижче представлено шари, які увійшли в обрану оптимальну модель.

– Conv2D(32, (3, 3), activation='relu', input_shape=[200, 200, 3]);
– MaxPooling2D();
– Conv2D(64, (2, 2), activation='relu');
– MaxPooling2D();
– Conv2D(128, (2, 2), activation='relu');
– MaxPooling2D();
– Conv2D(256, (2, 2), activation='relu');
– MaxPooling2D();
– Flatten();
– Dense(100, activation='relu');
– Dense(10, activation='softmax').

В представленій вище архітектурі вказано усі вхідні параметри мережі, включаючи функції активації, розміри, тощо.

Висновки до розділу 3

В ході написання третього розділу роботи було проведено проектування майбутнього програмного продукту та проектування архітектури майбутньої нейронної мережі.

Результати проектування відображені у вигляді UML–діаграм класів і варіантів використання, представлених в тексті розділу.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ЗАСТОСУНКУ

4.1 Числові характеристики нейронної мережі

Основні числові характеристики нейронних мереж виявляються на етапі навчання і підготовки до нього.

На етапі підготовки до навчання виділяють 2 основні характеристики:

- розміри датасетів;
- кількість епох навчання.

В даному випадку датасети розділені у відношенні 75% на 25%, тобто 75% датасету виділяється на навчання мережі і 25% на тестування. Кількість епох рівна 100.

Після навчання додаються нові характеристики, такі як точність.

Середня точність в даному випадку становить 0,9613 або 96,13%.

Таблиця 3.1 — Порівняння моделей

Назва	Швидкість навчання	Точність	Втрати	MSE
Наркотики	235 хвилин	0,9613	0,0387	0,0112
Паління	140 хвилин	0,8702	0,1208	0,0376
Зброя	190 хвилин	0,7908	0,2092	0,0606

4.2 Розробка графічного інтерфейсу користувача

Графічний інтерфейс користувача GUI — це форма інтерфейсу користувача, яка дозволяє користувачам взаємодіяти з електронними пристроями за допомогою графічних піктограм і аудіоіндикаторів, таких як первинна нотація, замість текстових інтерфейсів користувача, міток введених команд або текстової навігації.

Графічні інтерфейси користувача були введені у відповідь на сприйману стрімку криву навчання CLI (інтерфейсів командного рядка) [3-5], які вимагають введення команд на клавіатурі комп’ютера.

Дії в GUI зазвичай виконуються через пряме маніпулювання графічними елементами [6-8]. Крім комп’ютерів, GUI використовуються в багатьох кишенькових мобільних пристроях, таких як MP3-плеєри, портативні медіа-плеєри, ігрові пристрої, смартфони та невеликі побутові, офісні та промислові засоби керування. Термін GUI, як правило, не застосовується до інших типів інтерфейсів з нижчою роздільною здатністю дисплея, таких як відеоігри (де перевага віддається HUD (дисплею з головою) [9], або не включає плоскі екрани, такі як об’ємні дисплеї [10], оскільки термін обмежується сферою застосування двовимірних екранів, здатних описувати загальну інформацію, відповідно до традиції досліджень інформатики в дослідницькому центрі Xerox Palo Alto.

Розробка візуальної композиції та тимчасової поведінки графічного інтерфейсу є важливою частиною програмування прикладних програм у сфері взаємодії людини з комп’ютером. Його мета полягає в тому, щоб підвищити ефективність і простоту використання для основного логічного дизайну збереженої програми, дисципліни дизайну під назвою юзабіліті. Методи дизайну, орієнтованого на користувача, використовуються, щоб гарантувати, що візуальна мова, введена в дизайн, добре адаптована до завдань.

Функції видимого графічного інтерфейсу програми іноді називають chrome або GUI (вимовляється «липкий») [6-8]. Як правило, користувачі взаємодіють з інформацією, маніпулюючи візуальними віджетами, які дозволяють взаємодіяти відповідно до типу даних, які вони зберігають. Віджети добре розробленого інтерфейсу підібрані для підтримки дій, необхідних для досягнення цілей користувачів. Модель–подання–контролер дозволяє створювати гнучкі структури, в яких інтерфейс не залежить від функцій програми та опосередковано пов’язаний із ними, тому графічний інтерфейс можна легко налаштувати. Це дозволяє користувачам вибирати або створювати іншу оболонку за бажанням і полегшує

роботу дизайнера щодо зміни інтерфейсу в міру розвитку потреб користувача. Хороший дизайн графічного інтерфейсу більше стосується користувачів, а менше – архітектури системи. Великі віджети, такі як вікна, зазвичай забезпечують рамку або контейнер для основного вмісту презентації, такого як веб-сторінка, повідомлення електронної пошти або малюнок. Менші зазвичай діють як інструмент введення користувачами.

Графічний інтерфейс користувача може бути розроблений відповідно до вимог вертикального ринку як графічний інтерфейс користувача для конкретної програми. Приклади включають банкомати (ATM), сенсорні екрани торгових точок (POS) у ресторанах [14], каси самообслуговування, які використовуються в роздрібних магазинах, самостійне оформлення квитків і реєстрація авіакомпаній, інформаційні кіоски в громадських місцях, наприклад залізничний вокзал або музей, а також монітори або екрани керування у вбудованому промисловому додатку, який використовує операційну систему реального часу (RTOS).

Стільникові телефони та кишенькові ігрові системи також використовують графічний інтерфейс із сенсорним екраном для спеціальних програм. Новіші автомобілі використовують GUI у своїх навігаційних системах і мультимедійних центрах або комбінаціях навігаційних мультимедійних центрів.

При розробці графічного інтерфейсу користувача було використано PySimpleGUI.

PySimpleGUI — це бібліотека Python, яка містить tkinter, Qt (pyside2), wxPython і Remi (для підтримки браузера), що дозволяє дуже швидко та легко освоїти графічний інтерфейс програмування. PySimpleGUI за замовчуванням використовує tkinter, але користувач може перейти на іншу підтримувану бібліотеку GUI, просто змінивши один рядок.

PySimpleGUI можна встановити за допомогою «pip install pysimplegui» або скопіювавши файл «pysimplegui.py» зі сховища github

PySimpleGUI ліцензовано за GNULGPL 3.0

Проект PySimpleGUI структуровано як стартап-компанію, але має ліцензію на відкритий код. Це забезпечило більшу спритність і, що важливіше, здатність інтенсивно працювати над баченням, не відволікаючись. Проект не приймає пул-запити.

4 стовпи проекту PySimpleGUI включають всю систему PySimpleGUI. 1. Основне програмне забезпечення 2. Розширена документація, довідка про виклики та вбудована документація через рядки документів 3. 300+ демонстраційних програм, які дадуть негайний старт у проекті. 4. Безкоштовна підтримка клієнтів.

Графічний інтерфейс користувача, розроблений в ході створення програмного рішення, представляє собою одне графічне вікно, яке має декілька різних станів.

Перший стан вікна користувач бачить при старті програми (рис. 4.1).

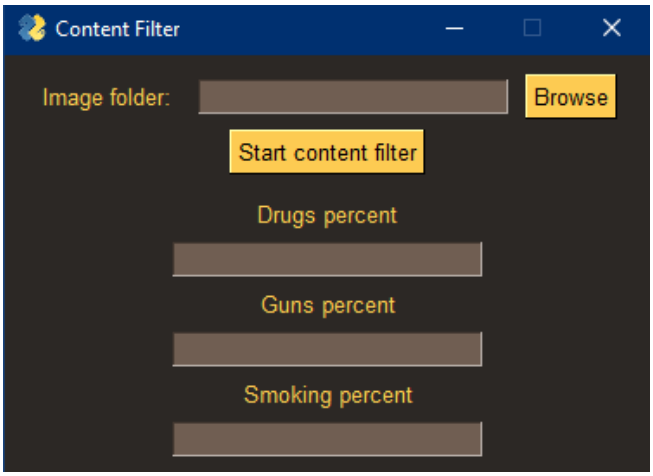

Рисунок 4.1 — Стартовий стан вікна

Після вибору зображення система переходить у другий стан, відображаючи обране зображення (рис. 4.2).

Рисунок 4.2 — Другий стан графічного інтерфейсу

Після натискання кнопки початку фільтрації та її закінчення система переходить в третій стан, а саме — стан показу результатів (рис. 4.3).

Рисунок 4.3 — Фінальний стан вікна

Графічний інтерфейс створено таким чином, щоб після першого використання користувач міг обрати наступне зображення і працювати з ним.

4.3 Тестування системи

Для тестування розробленої системи було обрано метод чорного ящика. Тестування чорного ящика — це метод тестування програмного забезпечення, який перевіряє функціональність програми, не заглядаючи в її внутрішні структури чи роботи. Цей метод тестування можна застосувати практично на кожному рівні

тестування програмного забезпечення: одиничному, інтеграційному, системному та приймальному. Його іноді називають тестуванням на основі специфікації.

Спеціальні знання коду програми, внутрішньої структури та знання програмування загалом не потрібні. Тестер знає, що програмне забезпечення має робити, але не знає, як воно це робить. Наприклад, тестувальник знає, що конкретні вхідні дані повертають певний незмінний вихід, але не знає, як програмне забезпечення виробляє вихідні дані в першу чергу.

Тестові випадки будуються на основі специфікацій та вимог, тобто того, що програма повинна робити. Тестові випадки, як правило, походять із зовнішніх описів програмного забезпечення, включаючи специфікації, вимоги та параметри конструкції. Хоча використовувані тести мають переважно функціональний характер, можуть використовуватися й нефункціональні тести. Конструктор тестів вибирає як дійсні, так і недійсні вхідні дані та визначає правильний вихід, часто за допомогою тестового оракула або попереднього результату, який, як відомо, є хорошим, без будь-якого знання внутрішньої структури тестового об’єкта.

Тестування для моделі фільтрації представлено в таблиці 3.1.

Таблиця 3.2 — Тест кейси для моделі фільтрації

№ п.п.	Опис	Очікуваний результат	Отриманий результат
1	Введення даних, що вибиваються з ряду	Виведення результату, який не відповідає дійсності, проте є точним з огляду на введені данні	Виведення результату, який не відповідає дійсності, проте є точним з огляду на введені данні
2	Введення текстових даних замість зображень	Відповідне повідомлення про	Відповідне повідомлення про

		помилку в роботі програми	помилку в роботі програми
3	Введення занадто великого зображення	Виведення результату, який не відповідає дійсності, проте є точним з огляду на введені данні	Виведення результату, який не відповідає дійсності, проте є точним з огляду на введені данні
4	Введення порожніх полів	Відповідне повідомлення про помилку в роботі програми	Відповідне повідомлення про помилку в роботі програми
55	Введення адекватних вхідних даних	Виведення результату	Виведення результату
6	Введення усіх полів порожніми	Відповідне повідомлення про помилку в роботі програми	Відповідне повідомлення про помилку в роботі програми

Тестування показує повну функціональну вірність, відсутність неточностей і багів в написаній системі.

Висновки до розділу 4

Четвертий розділ присвячено реалізації спроектованого програмного продукту та тестування отриманого результату.

В ході написання розділу було створено графічний інтерфейс системи та проведено тестування чорного ящика, в ході якого виявлено повну функціональність системи.

ВИСНОВКИ

Мета роботи полягає в підвищенні точності фільтрації контенту за рахунок розробки архітектури нейронної мережі.

В ході написання першого розділу роботи було виявлено значення основних понять, таких як фільтрація контенту та нейронні мережі. Було проаналізовано методи фільтрації контенту, основні поняття нейронних мереж, поставлено задачу на розробку.

Другий розділ роботи присвячено огляду інструментальних засобів розробки системи, таких як мова програмування і фреймворки, що дозволяють працювати з нейронними мережами.

Третій розділ роботи містить в собі інформацію стосовно моделювання системи, таку як: аналіз варіантів використання системи, на основі якого побудовано системи; графічне представлення внутрішньої будови системи, яке дозволяє зробити висновок про схему системи.

Четвертий розділ присвячено розробці програмного продукту і містить інформацію стосовно проектування архітектури нейронної мережі, розробки графічного інтерфейсу користувача та тестування системи, націлене на виявлення неточностей в роботі і перевірку роботоздатності в цілому.

В ході п’ятого розділу розроблено методичні матеріали для проведення практичної роботи для студентів.

В ході шостого розділу розглянуто основні вимоги до охорони праці та підтверджено придатність наявних умов.

В результаті проведеної роботи отримано повнофункціональну, навчену модель нейронної мережі, призначену для фільтрації забороненого контенту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методи розпізнавання образів : Навч. посіб. для студ. / В. М. Заяць, Р. М. Камінський; Нац. ун-т "Львів. політехніка". - Л., 2004. - 173 с. - Бібліогр.: 21 назв.

2. Дэвід А. Форсайт, Джин Понс. [Computer Vision: A Modern Approach]. — М. : «Вільямс», 2004. — 928 с. — ISBN 0-13-085198-1.

3. Джордж Стокман, Лінда Шапіро. [Computer Vision]. — М. : Біном. Лаборатория знаний, 2006. — 752 с. — ISBN 5947743841.

4. В. Н. Вапнік, А. Я. Червоненкіс Теорія розпізнавання образів М.: Наука, 1974. — 416 с.

5. Haykin (2008) Neural Networks and Learning Machines, 3rd edition

6. Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization in the Brain". Psychological Review. 65 (6): 386–408. CiteSeerX 10.1.1.588.3775. doi:10.1037/h0042519. PMID 13602029.

7. Werbos, P.J. (1975). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.

8. Rosenblatt, Frank (1957). "The Perceptron—a perceiving and recognizing automaton". Report 85-460-1. Cornell Aeronautical Laboratory.

9. Olazaran, Mikel (1996). "A Sociological Study of the Official History of the Perceptrons Controversy". Social Studies of Science. 26 (3): 611–659. doi:10.1177/030631296026003005. JSTOR 285702. S2CID 16786738.

10. Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". Neural Networks. 61: 85–117. arXiv:1404.7828. doi:10.1016/j.neunet.2014.09.003. PMID 25462637. S2CID 11715509.

11. Ivakhnenko, A. G. (1973). Cybernetic Predicting Devices. CCM Information Corporation.

12. Ivakhnenko, A. G.; Grigor′evich Lapa, Valentin (1967). Cybernetics and forecasting techniques. American Elsevier Pub. Co.

13. Schmidhuber, Jürgen (2015). "Deep Learning". Scholarpedia. 10 (11): 85–117. Bibcode:2015SchpJ..1032832S. doi:10.4249/scholarpedia.32832.

14. Dreyfus, Stuart E. (1 September 1990). "Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure". Journal of Guidance, Control, and Dynamics. 13 (5): 926–928. Bibcode:1990JGCD...13..926D. doi:10.2514/3.25422. ISSN 0731-5090.

15. Mizutani, E.; Dreyfus, S.E.; Nishio, K. (2000). "On derivation of MLP backpropagation from the Kelley-Bryson optimal-control gradient formula and its application". Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium. IEEE: 167–172 vol.2. doi:10.1109/ijcnn.2000.857892. ISBN 0-7695-0619-4. S2CID 351146.

16. Kelley, Henry J. (1960). "Gradient theory of optimal flight paths". ARS Journal. 30 (10): 947–954. doi:10.2514/8.5282.

17. "A gradient method for optimizing multi-stage allocation processes". Proceedings of the Harvard Univ. Symposium on digital computers and their applications. April 1961.

18. Minsky, Marvin; Papert, Seymour (1969). Perceptrons: An Introduction to Computational Geometry. MIT Press. ISBN 978-0-262-63022-1.

19. Linnainmaa, Seppo (1970). The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors (Masters) (in Finnish). University of Helsinki. pp. 6–7.

20. Linnainmaa, Seppo (1976). "Taylor expansion of the accumulated rounding error". BIT Numerical Mathematics. 16 (2): 146–160. doi:10.1007/bf01931367. S2CID 122357351.

21. Dreyfus, Stuart (1973). "The computational solution of optimal control problems with time lag". IEEE Transactions on Automatic Control. 18 (4): 383–385. doi:10.1109/tac.1973.1100330.

22. Werbos, Paul (1982). "Applications of advances in nonlinear sensitivity analysis" (PDF). System modeling and optimization. Springer. pp. 762–770.

23. Mead, Carver A.; Ismail, Mohammed (8 May 1989). Analog VLSI Implementation of Neural Systems (PDF). The Kluwer International Series in Engineering and Computer Science. 80. Norwell, MA: Kluwer Academic Publishers. doi:10.1007/978-1-4613-1639-8. ISBN 978-1-4613-1639-8.

24. David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams , "Learning representations by back-propagating errors ," Nature', 323, pages 533–536 1986.

25. J. Weng, N. Ahuja and T. S. Huang, "Cresceptron: a self-organizing neural network which grows adaptively," Proc. International Joint Conference on Neural Networks, Baltimore, Maryland, vol I, pp. 576–581, June 1992.

26. J. Weng, N. Ahuja and T. S. Huang, "Learning recognition and segmentation of 3-D objects from 2-D images," Proc. 4th International Conf. Computer Vision, Berlin, Germany, pp. 121–128, May 1993.

27. J. Weng, N. Ahuja and T. S. Huang, "Learning recognition and segmentation using the Cresceptron," International Journal of Computer Vision, vol. 25, no. 2, pp. 105–139, Nov. 1997.

28. J. Schmidhuber., "Learning complex, extended sequences using the principle of history compression," Neural Computation, 4, pp. 234–242, 1992.

29. Domingos, Pedro (22 September 2015). The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World. chapter 4: Basic Books. ISBN 978-0465065707.

30. Smolensky, P. (1986). "Information processing in dynamical systems: Foundations of harmony theory.". In D. E. Rumelhart; J. L. McClelland; PDP Research Group (eds.). Parallel Distributed Processing: Explorations in the Microstructure of Cognition. 1. pp. 194–281. ISBN 978-0-262-68053-0.

31. Ng, Andrew; Dean, Jeff (2012). "Building High-level Features Using Large Scale Unsupervised Learning". arXiv:1112.6209 [cs.LG].

32. Ian Goodfellow and Yoshua Bengio and Aaron Courville (2016). Deep Learning. MIT Press.

33. Cireşan, Dan Claudiu; Meier, Ueli; Gambardella, Luca Maria; Schmidhuber, Jürgen (21 September 2010). "Deep, Big, Simple Neural Nets for Handwritten Digit Recognition". Neural Computation. 22 (12): 3207–3220. arXiv:1003.0358. doi:10.1162/neco_a_00052. ISSN 0899-7667. PMID 20858131. S2CID 1918673.

34. Dominik Scherer, Andreas C. Müller, and Sven Behnke: "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition," In 20th International Conference Artificial Neural Networks (ICANN), pp. 92–101, 2010. doi:10.1007/978-3-642-15825-4_10.

35. 2012 Kurzweil AI Interview Archived 31 August 2018 at the Wayback Machine with Jürgen Schmidhuber on the eight competitions won by his Deep Learning team 2009–2012

36. "How bio-inspired deep learning keeps winning competitions | KurzweilAI". www.kurzweilai.net. Archived from the original on 31 August 2018. Retrieved 16 June 2017.

37. Graves, Alex; and Schmidhuber, Jürgen; Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks, in Bengio, Yoshua; Schuurmans, Dale; Lafferty, John; Williams, Chris K. I.; and Culotta, Aron (eds.), Advances in Neural Information Processing Systems 22 (NIPS'22), 7–10 December 2009, Vancouver, BC, Neural Information Processing Systems (NIPS) Foundation, 2009, pp. 545–552.

38. Graves, A.; Liwicki, M.; Fernandez, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. (2009). "A Novel Connectionist System for Improved Unconstrained Handwriting Recognition" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (5): 855–868. CiteSeerX 10.1.1.139.4502. doi:10.1109/tpami.2008.137. PMID 19299860. S2CID 14635907.

39. Graves, Alex; Schmidhuber, Jürgen (2009). Bengio, Yoshua; Schuurmans, Dale; Lafferty, John; Williams, Chris editor-K. I.; Culotta, Aron (eds.). "Offline

Handwriting Recognition with Multidimensional Recurrent Neural Networks". Neural Information Processing Systems (NIPS) Foundation. Curran Associates, Inc: 545–552.

40. Graves, A.; Liwicki, M.; Fernández, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. (May 2009). "A Novel Connectionist System for Unconstrained Handwriting Recognition". IEEE Transactions on Pattern Analysis and Machine Intelligence. 31 (5): 855–868. CiteSeerX 10.1.1.139.4502. doi:10.1109/tpami.2008.137. ISSN 0162-8828. PMID 19299860. S2CID 14635907.

41. Охорона праці в галузі : методичні вказівки / Укладач к.т.н., доц. каф. ТМ І. Г. Ткаченко. – Тернопіль, ТДТУ, 2001. – 32 с. 25.

42. Правила технічної експлуатації електроустановок споживачів.– К.: УкрНДПВТІ «Укрсільенергопроект», 2007.– 150 с. 26.

43. ДЕРЖАВНІ БУДІВЕЛЬНІ НОРМИ УКРАЇНИ, Інженерне обладнання будинків і споруд, ПРИРОДНЕ І ШТУЧНЕ ОСВІТЛЕННЯ, ДБН В.2.5-28- 2006.

44. Датасет для виявлення паління : вебсайт. URL: https://www.kaggle.com/datasets/vitaminc/cigarette-smoker-detection (дата звернення: 03.02.2023).

45. Датасет для виявлення наркотиків : вебсайт. URL: https://www.kaggle.com/datasets/vencerlanz09/pharmaceutical-drugs-and-vitamins-synthetic-images (дата звернення: 03.02.2023).

46. Датасет для виявлення зброї : вебсайт. URL: https://www.kaggle.com/datasets/issaisasank/guns-object-detection (дата звернення: 03.02.2023).

ДОДАТОК А

Лістинг коду навчання моделі для фільтрації наркотиків

```
import tensorflow
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
model_store_dir = 'drugs/drugs_detector.model'
image_generator = ImageDataGenerator(rescale = 1 / 255, validation_split = 0.25)
train_dataset = image_generator.flow_from_directory(batch_size = 4, directory = 'drugs-dataset/Data Combined', shuffle = True, target_size = (200, 200), subset = "training", class_mode = 'categorical')
validation_dataset = image_generator.flow_from_directory(batch_size = 4, directory = 'drugs-dataset/Data Combined', shuffle = True, target_size = (200, 200), subset = "validation", class_mode = 'categorical')
model = keras.models.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = [200, 200, 3]),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(64, (2, 2), activation = 'relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(128, (2, 2), activation = 'relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(256, (2, 2), activation = 'relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Flatten(),
    keras.layers.Dense(100, activation = 'relu'),
    keras.layers.Dense(10, activation = 'softmax')])
model.compile(loss = tensorflow.losses.CategoricalCrossentropy(), optimizer = 'adam', metrics = ['accuracy'])
callback = keras.callbacks.EarlyStopping(monitor = 'val_loss', patience = 3, restore_best_weights = True)
h = model.fit(train_dataset, epochs = 100, validation_data = validation_dataset, callbacks = callback)
model.save(model_store_dir, save_format = "h5")
```

ДОДАТОК Б

Лістинг коду навчання моделі для фільтрації зброї

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.image import resize_with_pad
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, Dropout, LeakyReLU, MaxPooling2D, UpSampling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import load_img, img_to_array
BASE_IMAGE_PATH = 'guns-object-detection/Images'
BASE_LABELS_PATH = 'guns-object-detection/Labels'
model_store_dir = 'guns/guns_detector.model'
TARGET_IMAGE_SIZE = (256, 256)
LEARNING_RATE = 0.0001
BATCH_SIZE = 16
EPOCHS = 100
VAL_SPLIT = 0.25
image_data = []
image_labels = []
image_count = len(os.listdir(BASE_IMAGE_PATH))
for i in range(1, image_count + 1):
    img = load_img(os.path.join(BASE_IMAGE_PATH, str(i) + '.jpeg'))
    img = img_to_array(img, dtype = 'uint8')
    image_data.append(img)
    label_mask = np.zeros((*img.shape[:2], 1))
    with open(os.path.join(BASE_LABELS_PATH, str(i) + '.txt'), 'r') as label_file:
        l_count = int(label_file.readline())
        for i in range(l_count):
            box = list(map(int, label_file.readline().split()))
```

```
            label_mask[box[1]:box[3], box[0]:box[2]] = 1.0
    image_labels.append(label_mask)
for i in range(image_count):
    image_data[i] = resize_with_pad(image_data[i], *TARGET_IMAGE_SIZE).numpy() / 255.0
    image_labels[i] = resize_with_pad(image_labels[i], *TARGET_IMAGE_SIZE).numpy()
image_data = np.array(image_data)
image_labels = np.array(image_labels)
def weighted_bce(y_true, y_pred):
    LOSS_WEIGHT = 8
    loss_ones = LOSS_WEIGHT * y_true * tf.math.log(y_pred + 1e-7)
    loss_zeros = (1 - y_true) * tf.math.log(1 - y_pred + 1e-7)
    loss = - loss_ones - loss_zeros
    return tf.reduce_mean(loss)
model = Sequential([
    Conv2D(32, 5, padding = 'same', input_shape = (*TARGET_IMAGE_SIZE, 3), activation = LeakyReLU()),
    MaxPooling2D(),
    Conv2D(64, 4, padding = 'same', activation = LeakyReLU()),
    MaxPooling2D(),
    Conv2D(128, 3, padding = 'same', activation = LeakyReLU()),
    Dropout(0.1),
    UpSampling2D(),
    Conv2D(64, 4, padding = 'same', activation = LeakyReLU()),
    UpSampling2D(),
    Conv2D(32, 5, padding = 'same', activation = LeakyReLU()),
    Dropout(0.1),
    Conv2D(1, 3, padding = 'same', activation = 'sigmoid'),
])
model.compile(optimizer = Adam(LEARNING_RATE), loss = weighted_bce)
model.summary()
history = model.fit(image_data, image_labels, batch_size = BATCH_SIZE, epochs = EPOCHS, validation_split = VAL_SPLIT, verbose = 1)
model.save(model_store_dir, save_format = "h5")
```

ДОДАТОК В

Лістинг коду навчання моделі для фільтрації паління

```
import os
import numpy as np
from imutils.paths import list_images
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
INIT_LR = 1e-4
EPOCHS = 100
BS = 32
dataset_path = 'smoking-dataset'
model_store_dir = 'smoking/smoking_detector.model'
imagePaths = list(list_images(dataset_path))
data = []
labels = []
for imagePath in imagePaths:
    label = imagePath.split(os.path.sep)[-2]
```

```
    image = load_img(imagePath, target_size = (224, 224))
    image = img_to_array(image)
    image = preprocess_input(image)
    data.append(image)
    labels.append(label)
data = np.array(data, dtype="float32")
labels = np.array(labels)
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size = 0.25, stratify = labels, random_state = 42)
aug = ImageDataGenerator(rotation_range = 20, zoom_range = 0.15, width_shift_range = 0.2, height_shift_range = 0.2, shear_range = 0.15, horizontal_flip = True, fill_mode = "nearest")
baseModel = MobileNetV2(weights = "imagenet", include_top = False, input_tensor = Input(shape = (224, 224, 3)))
headModel = baseModel.output
headModel = AveragePooling2D(pool_size = (7, 7))(headModel)
headModel = Flatten(name = "flatten")(headModel)
headModel = Dense(128, activation = "relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation = "softmax")(headModel)
model = Model(inputs = baseModel.input, outputs = headModel)
for layer in baseModel.layers:
    layer.trainable = False
opt = Adam(lr = INIT_LR, decay = INIT_LR / EPOCHS)
model.compile(loss = "binary_crossentropy", optimizer = opt,metrics = ["accuracy"])
H = model.fit(aug.flow(trainX, trainY, batch_size = BS), steps_per_epoch = len(trainX) // BS, validation_data = (testX, testY), validation_steps = len(testX) // BS, epochs = EPOCHS)
model.save(model_store_dir, save_format = "h5")
```

ДОДАТОК Г

Лістинг коду графічного застосунку

```
import PySimpleGUI as sg
from PIL import Image, ImageTk
from network_helper import guns, drugs, smoking
sg.theme('DarkAmber')
file_list_column = [
    [
        sg.Text("Image folder: "),
        sg.In(size = (25, 1), enable_events = True, key = "-FILE-"),
        sg.FileBrowse(),
    ],
    [sg.Button("Start content filter", key = "-START FILTERING-")],
    [sg.Image(key = "-START IMAGE-")],
    [sg.Text("Drugs percent")],
    [sg.In(size = (25, 1), enable_events = True, key = "-DOUT-")],
    [sg.Text("Guns percent")],
    [sg.In(size = (25, 1), enable_events = True, key = "-GOUT-")],
    [sg.Text("Smoking percent")],
    [sg.In(size = (25, 1), enable_events = True, key = "-SOUT-")],
]
layout = [
    [
        sg.Column(file_list_column, element_justification = 'c')
    ]
]
window = sg.Window("Content Filter", layout, element_justification = 'c')
while True:
    event, values = window.read()
    if event == "Exit" or event == sg.WINDOW_CLOSED:
```

```
        break
    if event == "-FILE-":
        try:
            filename = values["-FILE-"]
            file_image = Image.open(filename)
            resized = file_image.resize((300, 300))
            window["-START IMAGE-"].update(data=ImageTk.PhotoImage(resized))
        except:
            pass
    elif event == "-START FILTERING-":
        g = guns(window['-START IMAGE-'].image)
        d = drugs(window['-START IMAGE-'].image)
        s = smoking(window['-START IMAGE-'].image)
        window['-DOUT-'].update(str(d))
        window['-GOUT-'].update(str(g))
        window['-SOUT-'].update(str(s))
from tensorflow import keras
def guns(image):
    model_g = keras.models.load_model('guns_detector.model')
    result = model_g.predict(image)
    return result
def drugs(image):
    model_d = keras.models.load_model('drugs_detector.model')
    result = model_d.predict(image)
    return result
def smoking(image):
    model_s = keras.models.load_model('smoking_detector.model')
    result = model_s.predict(image)
    return result
```