Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра Комп'ютерних інтелектуальних технологій та систем
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Генерація правдоподібних зображень для навчання класифікаційних моделей

Виконав:
студент II курсу, групи КІТм-19-1
Колесников О.В.

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми Освітньо-професійна
Освітня програма
Комп'ютерні інтелектуальні технології
(повна назва освітньої програми)

Керівник: проф. Руденко О.Г.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри КІТС _____ Руденко О.Г.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
Кафедра Комп’ютерних інтелектуальних технологій та систем
Рівень вищої освіти другий (магістерський)
Спеціальність 123 – Комп’ютерна інженерія
Тип програми Освітньо-професійна
Освітня програма Комп'ютерні інтелектуальні технології

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Колесникову Олександру Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Генерація правдоподібних зображень для навчання класифікаційних моделей

затверджена наказом по університету від “ 11 ” листопада 2020 р. № 1582 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 10 грудня 2020 р.
3. Вхідні дані до роботи ТЗ, документація, звітні документи, програмне забезпечення для генерації правдоподібних зображень.

4. Перелік питань, що потрібно опрацювати в роботі Розгляд теоретичних матеріалів та створення програмного забезпечення для генерації правдоподібних зображень для навчання класифікаційних зображень. Систематизація знань про класифікаційні моделі та їх відмінність. Розбір фундаментальної сегментації моделей машинного навчання. Розгляд програмної бібліотеки TensorFlow.js

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Презентація з 17 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд стану проблеми та постановка задачі	11.11 – 13.11	
2	Аналіз літератури за напрямком магістерської	13.11 – 20.11	
	роботи		
3	Аналіз логістичної ситуації в регіоні	21.11 – 23.11	
4	Вибір методів рішення для реалізації та їхнє	24.11 – 26.11	
	обґрунтування		
5	Розробка агентної моделі логістичних послуг	26.11 – 30.12	
6	Оформлення пояснювальної записки	01.12 – 07.12	
7	Оформлення графічної частини	08.12 – 10.12	

Дата видачі завдання 11 листопада 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Руденко О.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 88 с., 25 рис., 41 джерело, 2 дод.

ШТУЧНИЙ ІНТЕЛЕКТ, КЛАСИФІКАЦІЯ, ГЕНЕРАЦІЯ ЗОБРАЖЕНЬ, СИНТЕЗ, TENSOFFLOW.JS.

Метою атестаційної роботи є дослідження генерації правдоподібних зображень для навчання класифікаційних моделей. Мета проведення дослідження полягає у вивченні методів та варіантів навчання класифікаційих моделей, виявлення недоліків та переваг існуючого ПЗ, та визначення особливостей обранної програмної бібліотеки для виконання генераціх необхідних зображень. Були протестовані найпопулярніші аналоги які мають можливість генерувати зображення. Крім того, що був проведений аналіз, була сформована думка про використання данного направлення та його вдосконалення.

Методи дослідження : аналіз, сінтез, генерація, систематизація, порівняння, експеримент.

У ході виконання атестаційної роботи були розглянуті теоретичні й практичні матеріали за наданою темою. Проведенна робота над аналізом наданих матеріалів, а також був сформований висновок що штучний інтеллект має властивості для вдосконалення існуючих вирішень важливих задач.

ABSTRACT

88 pages, 25 figures, 41 sources, 2 appendices.

ARTIFICIAL INTELLIGENCE, CLASSIFICATION, IMAGE GENERATION, SYNTHESIS, TENSOFFLOW.JS.

The purpose of the certification work is to study the generation of plausible images for the training of classification models. The purpose of the study is to study the methods and options for teaching classification models, to identify the disadvantages and advantages of existing software, and to determine the features of the selected software library to generate the necessary images. The most popular analogues that have the ability to generate images were tested. In addition to the analysis, an opinion was formed about the use of this area and its improvement.

Research methods: analysis, synthesis, generation, systematization, comparison, experiment.

In the course of attestation work the theoretical and practical materials on the given topic were considered. Work was carried out on the analysis of the provided materials, and also the conclusion was formed that artificial intelligence has properties for improvement of the existing decisions of important problems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ШІ – штучний інтелект

НМ – нейронні мережі

AI – artificial intelligence

ЕОМ – електронно-обчислювальна машина

ЕО – еволюційні обчислення

API – application programming interface

PCA – principal component analysis

CPU – central processing unit

CPU – graphics processing unit

MNIST – modified national institute of standards and technology

ВСТУП

Постійно зростаючі досягнення складних моделей, що спираються на потужні алгоритми навчання, такі як ті, що використовуються в Deep Learning, останнім часом почали виходити за свої академічні межі до їх масового розгортання у безлічі сценаріїв застосування. Ця поява була особливо помітною у сільському господарстві, транспорті, промисловості та безлічі мобільних додатків, які допомагають користувачам у їх повсякденному житті. Однак дизайн цих методів моделювання часто зумовлюється його ефективністю (нехай модель працює якнайкраще для даного завдання), тим самим не враховуючи контекст, в якому модель, після навчання, буде використовуватися (відповідно, нехай модель може бути зручною та зрозумілою для користувача). Щоб подолати цю проблему, кількісна оцінка та передача можливостей, обмежень та застережень моделей глибокого навчання повинні бути обов'язковими для її практичності в реальних середовищах застосування. На жаль, користувач, який отримує інформацію про модель, може не бути експертом з питань штучного інтелекту, що, зрештою, породжує відсутність надійності та його / її кінцеве небажання приймати розроблену модель у складних нішах застосування.

У цьому контексті сучасні підходи до кількісної оцінки ефективності моделей глибокого навчання суперечать поняттям, які загальний користувач може оцінити. З цієї причини стратегії та методи, спрямовані на подолання розриву між цими двома сферами знань, були широко досліджені останнім часом. Серед них є дослідження, які перевіряють стійкість моделей глибокого навчання за допомогою змагальних атак. Однак результати цих досліджень все ще далекі від того, щоб бути зручними для користувачів, оскільки їх практичні наслідки, зроблені для певних додатків, не можна визначити. Подальші методи пояснення моделей глибокого навчання включають візуалізацію внутрішнього подання моделі, вимірювання обсягу

уваги, яку приділяє модель кожному з входів для певного тестового зразка, та багато іншого, що висвітлюється в нещодавно наданих оглядових оглядах.

1 ШТУЧНИЙ ІНТЕЛЕКТ БАЗОВІ ПОНЯТТЯ

1.1 Становлення штучного інтелекту невід’ємною частиною розвитку

Штучний інтелект - одна з найрозвиненіших областей світу. Більш того, роздивляються системи, у яких ШІ використовується частково наприклад, для роботи з текстами або для спеціалістів творчого напрямку.

Ця область створена на перетині багатьох дисциплін: інформатики, філософії, кібернетики, математики, психології, фізики, хімії та ін. На сьогодішній день використовується у багатьох галузях, замість людини котра виконувала це раніше. ШІ стає допомогую у роботі, і інколи краще замінити людину на ШІ для уникнення помилок, а також для виконання рутинних задач автоматизовано. Технологія все частіше використовується в системах, як програмних, так і апаратних, створених на основі штучного інтелекту. Це машини з електронікою, що використовують штучний інтелект, і останні роботи, пов’язані з виробництвом чогось, і комп’ютерне програмне забезпечення, що включає ігри зі штучним інтелектом. Мета полягає у створенні повного штучного інтелекту, тобто такого, який може виконувати операції з обробки інформації нарівні з людьми чи краще - це насамперед поліпшення людського життя та подальше підвищення ступеня автоматизації виробництва. Тоді людині доведеться виконати лише творчу роботу, яка принесе їй задоволення. Але на сучасному етапі розвитку цієї галузі створення таких систем повного штучного інтелекту досить далеко, і поки що нам довелося обмежитися лише частковим втручанням ШІ в інші інтелектуальні системи. В основному це програмні засоби. Наприклад, системи експертності, системи розпізнавання зображень тощо. Про них ми поговоримо пізніше. Вони класифікуються як системи штучного інтелекту, оскільки вони можуть виконувати власні, хоча і дуже обмежені завдання, які раніше не могли виконувати комп’ютери, а ефекти їх роботи схожі на

результати аналогічної інтелектуальної праці людей. Все це сьогодні може зробити штучний інтелект, але все розпочалось із простих завдань формалізації: логічних ігор (шахи, шашки, цифри тощо). Американський кібернетик А. Самуель створив комп’ютерну програму, яка допомагає йому грати в шашки, і під час гри машина запам’ятовує, або принаймні створює враження, що вона вчиться, і вдосконалює свою гру на основі накопиченого досвіду. У 1962 р. З цим додатком грав Р. Нілім, наймогутніший гравець у шашки США, і переміг. Звідси можна сказати і розпочати вивчення штучного інтелекту. Визначення штучного інтелекту Тьюрінга використовувалося тоді і сьогодні: "Комп'ютер можна вважати розумним, якщо він дозволяє нам вірити, що ми маємо справу не з машиною, а з людиною".

1.2 Базові поняття, визначення

Ідея штучного інтелекту (ще одне можливе скорочення ШІ), як і будь-якого іншого інтелекту, дуже розмита. Термін інтелект походить від латинського інтелекту - тобто розум; розумові здібності людини. Відповідно, штучний інтелект (ШІ) зазвичай трактується як характеристика автоматизованих систем для виконання певних функцій людського інтелекту, наприклад, вибір та прийняття оптимальних рішень на основі попереднього досвіду та раціонального аналізу зовнішніх впливів.

Інтелект - це здатність мозку розв’язувати інтелектуальні проблеми шляхом набуття, запам’ятовування та цілеспрямованої зміни знань у процесі вивчення досвіду та адаптації до різних обставин. У цьому визначенні термін "знання" стосується не лише інформації, яка надходить до мозку за допомогою органів чуття. Цей тип знань надзвичайно важливий, але недостатній для інтелектуальної діяльності. Справа в тому, що предмети нашого оточення мають таку властивість, що вони не лише впливають на органи чуття, але і залишаються в конкретних стосунках між собою. Очевидно, що для ведення інтелектуальної діяльності в навколишньому

середовищі (або принаймні просто існування) у системі знань потрібна модель знань. У цій інформаційній моделі навколишнього середовища реальні об'єкти, їх властивості та відносини між ними не тільки представлені та запам'ятовуються, але, як зазначено у цьому визначенні інтелекту, можуть бути "навмисно" психічно змінені. Важливо, щоб модель зовнішнього середовища формувалася «в процесі навчання на досвіді та адаптації до різних обставин».

Щоб пояснити, чим інтелектуальне завдання відрізняється від простого, необхідно додати термін "алгоритм" - один із наріжних термінів кібернетики. Під алгоритмом розуміють точний порядок виконання в певному порядку системи дій для вирішення будь-якої задачі із заданого класу (групи) завдань. Термін "алгоритм" походить від імені узбецького математика аль-Хурзамі, який виділив найпростіші арифметичні алгоритми в IX столітті. У математиці та кібернетиці тип задачі певного типу вважається розв’язаною, коли знайдений алгоритм її розв’язання. Пошук алгоритмів є природною метою людини при вирішенні різних типів проблем. Пошук алгоритму таких проблем передбачає делікатні та складні міркування, які вимагають винахідливості та великої майстерності. Передбачається, що цей вид діяльності вимагає участі людського інтелекту. Завдання, пов’язані з пошуком алгоритму вирішення типу задачі певного типу, називатимуться розумними.

Як тільки такий алгоритм знайдений, процес вирішення відповідних завдань стає таким, який може бути виконаний в точності людиною, комп'ютером (правильно запрограмованим) або роботом, які не мають уявлення про природу самої проблеми. Необхідно тільки, щоб особа, відповідальна за завдання, виконувала основні операції, що становлять процес, і, крім того, послідовно і ретельно керуватилася запропонованим алгоритмом. Така людина, діючи тільки механічно, може успішно вирішити будь-яке завдання подібного роду. Тому цілком природним можна вважати виключення зі статусу їх інтелектуальних завдань, для яких існують

стандартні методи рішення. Прикладами таких завдань є чисто обчислювальні завдання: рішення системи лінійних алгебраїчних рівнянь, чисельне інтегрування диференціальних рівнянь і т. д. Для того щоб вирішити цього типу завдання існують стандартні алгоритми, які представляють собою певну цепочку елементарних операцій, які можуть бути легко реалізовані у вигляді комп'ютерної програми. З іншого боку, для широкого кола інтелектуальних завдань, таких як розпізнавання образів, шахи і доказ пропозицій, це формальний поділ процесу пошуку рішення на окремі елементарні етапи досить складно, хоча їх рішення просто.

Таким чином, ми можемо переформулювати визначення інтелекту як універсального супер-алгоритму, здатного створювати алгоритми для вирішення конкретних проблем.

Ще однією цікавою приміткою є те, що професія програміста, виходячи з наших визначень, є однією з найрозумніших, оскільки продуктом діяльності програміста є програми - алгоритми в чистому вигляді. Тому створення елементів ШІ повинно збільшити продуктивність роботи.

Діяльність мозку (яка має інтелект), яка спрямована на вирішення інтелектуальних проблем, називається інтелектуальним мисленням або діяльністю. Інтелект і мислення органічно пов'язані з вирішенням проблем, таких як доказ пропозицій, логічний аналіз, ідентифікація ситуації, планування поведінки, ігри та управління в умовах невизначеності. Типовими характеристиками інтелекту, які відображаються у процесі вирішення проблеми, є здатність вчитися, узагальнювати, набувати досвід (знання та вміння) та пристосовуватися до мінливих умов у процесі вирішення проблеми. Завдяки цим особливостям інтелекту мозок може вирішувати різні завдання, а також легко переробляти одне завдання до іншого. Таким чином, мозок, оснащений інтелектом, є універсальним засобом вирішення широкого кола проблем (включаючи неформальні), яких не існує стандартних і звичних методів вирішення.

Маємо на увазі, що існують інші, суто поведінкові (функціональні)

визначення. Отже, на думку А. Н. Колмогорова, кожна матеріальна система, з якої з часом можна обговорювати проблеми науки, літератури та мистецтва, має інтелект. Іншим прикладом поведінкової інтерпретації інтелекту є відоме визначення А. Тьюрінга. Сенс такий, у різних кімнатах є люди та машина. Вони не можуть бачити одне одного, але у них є можливість обміну інформацією (наприклад, електронною поштою). Якщо в процесі діалогу між учасниками гри люди не визначать, що хтось із учасників є автомобілем, то такий автомобіль можна вважати розумним.

До речі, цікавий план мислительної фігури, запропонований А. Тьюрінгом. «Щоб імітувати інтелект дорослої людини, - пише Тьюрінг, - нам потрібно багато думати про процес, за допомогою якого мозок людини досяг теперішнього стану. Якщо інтелект дитини правильно підготовлений, він стає інтелектом дорослого. Наш розрахунок полягає в тому, що такий пристрій можна легко запрограмувати ... тому ми розбиваємо свою проблему на дві частини: завдання побудови “дитячої програми” та завдання “виховання” цієї програми».

Забігаючи наперед, ми можемо сказати, що саме так використовуються майже всі системи ШІ. Зрозуміло, що практично неможливо покласти всі знання у досить складну систему. Більше того, лише таким чином будуть виражені вищезазначені ознаки інтелектуальної діяльності (накопичення досвіду, адаптація тощо).

1.3 Штучний інтелект і системи ШІ

1.3.1 Проблема подання знань в штучном інтелекті

Головною характеристикою розумних систем є те, що вони базуються на знаннях, а точніше, на деяких своїх ідеях. Знання тут розуміються як інформація, що зберігається (через ЕОМ), формально відповідно до певних правил, які ЕОМ може використовувати для виведення певних алгоритмів.

Найважливішою проблемою є опис змісту проблем у найширшому розумінні, що означає, що повинна застосовуватися форма опису знань, яка забезпечує належну обробку змісту відповідно до деяких формальних правил. Ця проблема називається проблемою подання знань.

В даний час чотирма найвідомішими підходами до подання знань у розглянутих системах є: виробнича модель; логічна модель; семантичні мережі; рамки. Правила продукційності - це найпростіший спосіб представити знання. Він базується на поданні знань у формі правил, побудованих відповідно до комітету "якщо - то". Частина правила "Якщо" називається пакетом, а "те" - це висновок або дія.

1.3.2 Різні підходи до побудови систем штучного інтелекту

Історично склалося три основних напрямки моделювання ШІ.

У першому підході об’єктом дослідження є структура та механізми людського мозку, а кінцевою метою є розкриття розуму. Необхідними етапами досліджень у цьому напрямку перша - побудова моделей на основі психофізіологічних даних, друга - проведення на них експериментів, нові гіпотези про механізми інтелектуальної діяльності, третя - вдосконалення моделей тощо. Другий підхід як об'єкт дослідження розглядає ШІ. Тут ми говоримо про моделювання інтелектуальної діяльності за допомогою комп’ютерів. Метою роботи в цьому напрямку є створення алгоритмічних комп’ютерів та програмного забезпечення, які дозволяють вирішувати не менш інтелектуальні проблеми, ніж люди.

Нарешті, третій підхід фокусується на створенні змішаної людської машини, або, як вони кажуть, інтелектуальних інтелектуальних систем, симбіозі можливостей природного та штучного інтелекту. Основними проблемами цих досліджень є оптимальний розподіл функцій між природним та штучним інтелектом та організація діалогу між людиною та машиною.

Існує кілька підходів до побудови систем ШІ. Ця класифікація не є історичною, коли одна думка поступово замінює іншу, а різні підходи все ще існують. Більше того, оскільки в даний час не існує справді повноцінних систем ШІ, не можна сказати, що один підхід є хорошим, а інший - ні.

Спочатку ми коротко розглянемо логічний підхід. Чому воно ввиникло? Основою цього логічного підходу є булева алгебра. Кожен програміст знає це і з логічними операторами, оскільки він керує оператором IF. Булева алгебра отримала свій подальший розвиток у формі предикатної арифметики - в якій вона була розширена шляхом подання тематичних символів, взаємозв'язків між ними, кількісного існування та загальності. Майже кожна система ШІ, побудована за логічним принципом, є машиною для доведення теорем.

Вихідні дані зберігаються в базі даних у вигляді аксіом, правила будуються як зв'язок між ними. Крім того, кожна з цих машин має цільовий блок виробництва, і система опалення намагається довести цю мішень як теорему. Як тільки мета доведена, дотримання застосованих правил дасть вам ряд дій, необхідних для досягнення мети. Потужність такої системи визначається можливостями генератора цілі та машини доводити теореми.

Звичайно, можна сказати, що ступеня вираженості алгебри виразу недостатньо для повноцінної реалізації ШІ, але варто згадати, що основи для всіх існуючих комп’ютерів - розділ пам'яті, яка може приймати лише значення 0 і 1. Таким чином, мало б сенс припустити, що все можливе може бути реалізовано у формі парадоксальної логіки. Хоча вже деякий час тут нічого нового не говорится.

Досягнення більшої виразності логічного підходу дозволяє здійснити такий порівняно новий напрямок, як незрозуміла логіка. Основна відмінність полягає в тому, що істина в твердженні може приймати в ньому крім так / ні (1/0) ще й про проміжні значення - не знаю. Цей підхід більше нагадує людське мислення, оскільки рідко відповідає на запитання лише так чи ні.

Більшість логічних методів займають багато часу, оскільки доказів

можна шукати багато. Отже, цей підхід вимагає ефективної реалізації процесу обчислення і, як правило, добре працює при відносно невеликому розмірі бази даних.

Під структурним підходом мається на увазі спроби побудови ШІ шляхом моделювання структури людського мозку. Однією з перших таких спроб стала концепція Френка Розенблатта. Найважливішою уявною структурною одиницею у людини (як і в більшості інших версій моделювання мозку) є нейрон.

Пізніше з'явилися інші моделі, що отримали назву "нейронні мережі" (НМ). Ці моделі відрізняються будовою окремих нейронів, топологією взаємозв’язків та алгоритмами навчання.

Серед найбільш відомих версій НМ на даний момент ми можемо виділити НМ із зворотним поширенням помилок, мережу Хопфілда, стохастичні нейронні мережі.

НМ здебільшого успішно використовується в завданнях розпізнавання зображень, включаючи завдання з перебільшеним шумом, але є приклади успішної реалізації для побудови систем ШІ.

Моделі, побудовані на мотивах людського мозку, характеризуються не надто експресивністю, простою аналогією алгоритмів та відповідною високою продуктивністю паралельно застосовуваних НМ. Такі мережі також характеризуються властивістю, яка наближає їх до людського мозку - нейронні мережі також працюють з неповною інформацією про навколишнє середовище, тобто як люди вони можуть відповідати на питання не лише `` так "і " "ні", але і "не впевнений, але скорше так".

Еволюційний підхід є досить поширеним явищем. Створюючи системи штучного інтелекту за цим підходом, основна увага приділяється побудові оригінальної моделі та правил, які вона може змінювати відповідно до (еволюціонувати). І модель може складатися з ряду методів, це може бути НМ і набір логічних правил та будь-яка інша модель. Потім ми вмикаємо комп’ютер, і він вибирає на основі перевірки моделей найкращу, на основі

якої створюються нові моделі за різними правилами, з яких знову обираються найкращі тощо.

В принципі, можемо сказати, що еволюційні моделі самі по собі не існують, це лише еволюційні алгоритми навчання, але моделі, отримані за допомогою еволюційного підходу, мають деякі характерні особливості, що дозволяють класифікувати їх в окремий клас.

Такі особливості є найважливішою передачею роботи програміста для побудови моделі на основі його алгоритму трансформації, а також тим фактом, що отримані моделі не супроводжуються виробництвом нових знань про навколишнє середовище навколо системи ШІ, тобто вона стає річ сама по собі.

Іншим широко застосовуваним підходом до побудови систем ШІ є моделювання. Цей підхід є класичним для кібернетики з одним з основних понять - "чорною скринькою" (чс).

Чс - пристрій, програмний модуль або набір даних, інформація про його внутрішню структуру та вміст, яка повністю відсутня, але специфікація для введення та виведення відома. Об'єкт, поведінка якого імітується, є таким "чорною скринькою". Що б не було в моделі і як вона працює, важливо те, що наша модель буде поводитися однаково в подібних ситуаціях.

Так формується ще одна риса людини: здатність копіювати те, що роблять інші, не вдаючись у подробиці, чому це необхідно. Ця здатність зазвичай економить йому багато часу, особливо на ранньому етапі життя.

Найбільшим недоліком методу моделювання є також низька інформаційна здатність більшості моделей, побудованих з ним.

Нарешті, використовуючи різні методи та підходи для побудови систем штучного інтелекту, зазначу, що на практиці між ними немає чіткої межі. Часто бувають змішані системи, де частина робіт виконується над одним типом та інша - інакшими.

1.4 Аналіз питання штучного інтелекту

1.4.1 Перспективні технології і застосування ШІ систем

Спочатку ми коротко обговоримо найбільш активно розвиваються підходи до ШІ - порядок зниження популярності серед професіоналів. Слід зазначити, що менша популярність часто пов’язана з технологічним потенціалом, але з віддаленістю від шансів на реалізацію (наприклад, найвищий потенціал для маскових рослин все ще не дуже важливий через невирішені проблеми управління.)

1.4.2 Нейронні мережі

Цей напрям знаходится на першому місці. Вдосконалення алгоритмів навчання та класифікації в режимі реального часу, обробки природної мови, розпізнавання зображень, мови, сигналів та створення зручних моделей інтерфейсу. Одне з основних додатків, розв'язуваних нейронними мережами - фінансове прогнозування, видобуток даних, діагностика системи, моніторинг мережі, шифрування даних. В останні роки шукаються ефективні методи синхронізації роботи нейронних мереж на паралельних пристроях.

1.4.3 Еволюційні обчислення

На розвиток галузі еволюційної інформатики суттєво вплинули, головним чином, інвестиції в нанотехнології. ЕО впливає на практичні проблеми самозбірки, самоконфігурації та самовідновлення систем, що складаються з багатьох вузлів, що працюють паралельно. Можуть бути використані наукові досягнення в сфері цифрових торгових автоматів.

Іншим аспектом ЕО є використання автономних агентів для вирішення повсякденних завдань, таких як особисті секретарі, особисті бухгалтери,

асистенти, що вибирають необхідну інформацію в мережах за допомогою алгоритмів пошуку третього покоління, планувальники роботи, особисті репетитори, віртуальні продавці тощо. робототехніка та всі суміжні галузі. Основними напрямками розвитку є розробка стандартів, відкритих архітектур, розумних оболонок, мов сценаріїв, запитання про методи ефективної взаємодії між програмами та людьми. Очікується, що моделі автономної поведінки будуть активно впроваджуватися в різні побутові прилади, які можуть прибирати місце, замовляти та готувати, керувати автомобілями тощо.

У майбутньому групи автономних агентів будуть використовуватися для вирішення складних проблем (швидке вивчення веб-контенту, великі масиви даних). Для цього необхідно вивчити можливі напрями розвитку подібних колективів, спільне планування роботи, методи спілкування, групові дослідження, поведінку спільних дій у незрозумілому середовищі з неповною інформацією, поведінку коаліції для агентів, які об’єднуються "на основі інтересів", навчитися спілкуватися з конфліктами тощо.

Існують різні соціальні аспекти - як суспільство насправді буде ставитись до таких спільнот інтелектуальних програм.

1.4.4 Інші популярні группи технологій

Нечітка логіка - нечіткі логічні системи будуть найактивніше використовуватися, переважно в гібридних системах управління.

Напрямок обробки зображень дуже популярний. На далі продовжиться розробка методів представлення та аналізу зображень (стиснення, кодування під час надсилання за різними протоколами, обробка біометричних зображень, супутникових зображень), незалежно від обладнання для відтворення, оптимізації відтворення кольорів на екрані та при друці, методів отримання розподіленого зображення .

Подальший розвиток включатиме засоби пошуку, індексації та аналізу

значення зображень, узгодження змісту довідкового каталогу в автоматизованому каталозі, організація захисту від копіювання, а також алгоритми машинного сигналу, розпізнавання та класифікація зображень.

Попит на експертні системи все ще досить високий. Сьогодні найбільша увага приділяється системам прийняття рішень, орієнтованим на час, близьким до реальності, зберіганню, пошуку, аналізу та моделюванню систем знань та динамічного проектування.

Збільшення кількості інтелектуальних додатків, які можуть швидко знайти оптимальні рішення комбінаторних проблем (таких як проблеми, що виникають у транспортних задачах), пов'язане з промисловим зростанням у розвинених країнах.

Безліч мереж передачі даних та створення потужних кластерів викликали інтерес до розподілених обчислень - баланс ресурсів, оптимальне навантаження процесора, самоконфігурація пристрою для максимальної ефективності, елементи відстеження, які слід оновити, виявлення невідповідності мережевих об'єктів, правильна діагностика роботи програми, подібні системи.

Поява незалежних робототехнічних блоків збільшує вимоги до систем управління в режимі реального часу - організації процесів самоадаптації, планування технічного обслуговування, використання засобів штучного інтелекту для прийняття рішень в умовах дефіциту.

Останніми роками компанії, які займаються організацією великих програмних систем (програмна інженерія), особливо цікавляться ШІ. Методи ШІ все частіше використовуються для аналізу оригінальних текстів та розуміння їх змісту, управління вимогами, розробки специфікацій, дизайну, створення коду ШІ, перевірки, тестування, оцінки якості, розпізнавання багаторазового використання, вирішення проблем у паралельних системах. Розробка програмного забезпечення поступово переростає у так звану інтелектуальну інженерію, яка займається більш загальними проблемами представлення та обробки знань (тоді як основні зусилля в інтелектуальній

інженерії зосереджені на способах перетворення інформації у знання).

Традиційно великий інтерес до ШІ спостерігається серед розробників ігор та розважальних програм (це окрема тема). Серед нових сфер їх досліджень - моделювання соціальної поведінки, спілкування, емоцій людини, творчості.

1.5 Підсумки і проблеми

Звіти про унікальні досягнення експертів зі штучного інтелекту, що обіцяють унікальні можливості, зникли зі сторінок науково-популярних видань багато років тому. Ейфорія, яка супроводжувала перші практичні успіхи в ШІ, була досить швидкою, оскільки перехід від експериментальних досліджень комп'ютерного моделювання до вирішення реальних додатків виявився набагато складнішим, ніж очікувалося. Експерти з усього світу вказали на труднощі такого переходу, і після детального аналізу стало зрозуміло, що майже всі проблеми пов'язані з нестачею ресурсів двох типів: комп'ютерних (обчислювальна потужність, оперативна пам'ять і ємність зовнішньої пам'яті) та людських. Розумна розробка програмного забезпечення вимагає залучення провідних експертів у різних галузях знань та організації довгострокових дослідницьких проектів).

Наразі ресурси першого типу досягли (або будуть випущені в найближчі п’ять-десять років) того рівня, який дозволяє системам ШІ вирішувати дуже складні практичні людські проблеми. Але з ресурсами другого типу ситуація у світі ще гірша, і тому результати діяльності зі штучного інтелекту здебільшого пов’язані з обмеженою кількістю провідних центрів штучного інтелекту у найбільших університетах.

У всіх цих сферах найбільші труднощі випливають з того, що принципи інтелектуальної діяльності людини, прийняття рішень та вирішення проблем недостатньо вивчені та зрозумілі. В основному, «розуміння», «мислення» вимагає розуміння.

Якщо питання «Чи може комп’ютер думати» часто обговорюється в 1960-х роках, питання переноситься: «Чи досить добре людина розуміє, як вона має намір передати цю функцію від комп’ютера»? З цієї причини робота у галузі штучного інтелекту тісно пов’язана з дослідженнями у відповідних розділах психології, фізіології та лінгвістики.

Найважливішим фактором, що в даний час сприяє розвитку технології ШІ, є зростання обчислювальної потужності, оскільки принципи психіки людини досі незрозумілі. Тому тема конференцій ШІ здається досить стандартною, і склад вже давно не змінювався. Але збільшення випуску сучасних комп’ютерів у поєднанні з поліпшенням якості алгоритмів дає можливість застосовувати різні наукові методи на практиці. Так сталося з інтелектуальними іграшками, так сталося з домашніми помічниками.

Тимчасово забуті методи простого пошуку варіантів (як у шахових програмах) будуть інтенсивно розроблятися знову, використовуючи дуже простий опис об'єктів. Але при такому підході (головне джерело успішної реалізації - продуктивність), як і очікувалося, можна вирішити безліч різних завдань (наприклад, у галузі шифрування). Безпечна робота автономних пристроїв у складному світі допоможе простим, але ресурсомістким адаптивним алгоритмам поведінки. Мета - розробити системи, які не схожі на людей, а поводяться як люди.

Вчені намагаються дивитись у більш далеке майбутнє. Чи можна створити автономні пристрої, які можуть самостійно збирати (дублювати) подібні копії, якщо це необхідно? Чи може наука створити відповідні алгоритми? Чи можемо ми керувати такими машинами? На ці запитання досі немає відповідей.

Продовження впровадження формальної логіки в прикладні системи подання та обробки знань буде знов актуальним. У той же час така логіка не здатна відображати реальне життя, і в одній оболонці буде поєднання декількох логічних систем опалення. Може бути можливим перейти від концепції подання детальної інформації про об'єкти та методи до

маневрування цією інформацією до більш абстрактних формальних описів та використання універсальних механізмів зсуву, а самі об'єкти характеризуватимуться невеликим розміром вибірки даних на основі розподілу ймовірностей.

Сфера ШІ, яка стала зрілою наукою, поступово розвивається - повільно, але впевнено вперед. Тому результати були досить добре передбачені, хоча не можна виключати раптових проривів, пов’язаних зі стратегічними ініціативами. Наприклад, у 1980-х рр. Національна комп'ютерна ініціатива США видалила з лабораторій багато областей ШІ та мала значний вплив на розвиток теорії високопродуктивних обчислень та її застосування в широко застосовуваних проектах. Такі ресурси можуть виникати на стику різних математичних дисциплін - теорії ймовірностей, нейронних мереж, неясної логіки.

2 РОЗВИТОК ТЕХНОЛОГІЙ, ЗАВДЯКИ ЯКИМ З'ЯВИЛИСЯ АЛГОРИТМИ ГЕНЕРАЦІЇ ПІДРОБНИХ ЗОБРАЖЕНЬ

2.1 Спосіб дії нейронних мереж

Найрозвиненішою сферою машинного навчання є глибокі нейронні мережі. Внаслідок розвитку цієї галузі почала з'являтися велика кількість алгоритмів, заснованих на глибокому навчанні. Усі основні алгоритми для створення підроблених зображень використовують для цього нейронні мережі.

Штучні нейронні мережі - це комп'ютерні системи, натхненні біологічними нейронними мережами, що складають людський мозок. Такі системи вчаться виконувати різні завдання (крок за кроком для підвищення їх продуктивності), маючи на увазі приклади, як правило, без спеціального програмування цих завдань. Наприклад, нейронні мережі при розпізнаванні зображень можуть навчитися ідентифікувати зображення, що містять собак, аналізуючи приклади зображень, що розподіляються як "собака" та "не собака" та використовуючи результати для ідентифікації собак на інших зображеннях. Вони роблять це без подальшого знання собак, таких як шерсть, хвіст та вуха. Натомість вони розробляють власний набір відповідних характеристик із навчального матеріалу, який вони обробляють.

Основною метою підходу до штучної нейронної мережі було вирішення проблем таким же чином, як і мозок людини. З часом увага була зосереджена на дотриманні певних розумових здібностей, що призвело до відхилень від біології. Штучні мережі використовувались для різноманітних завдань, включаючи розпізнавання мови, комп'ютерний зір, машинний переклад, настільні та відеоігри, фільтрацію соціальних мереж та медичну діагностику.

Нервові мережі містять нейрони, складені з подразників, які залежать

від окремого часового параметра; Можливий поріг, який залишається незмінним, якщо не змінюється функцією навчання; Функція збудження, яка обчислює нове збудження в певний момент; І функція виводу, яка обчислює операційний результат.

Вхідний нейрон діє як вхідний інтерфейс для всієї мережі. Вихідний нейрон також не має спадкоємців, і тому він діє як вихідний інтерфейс для всієї мережі (рисунок 2.1).

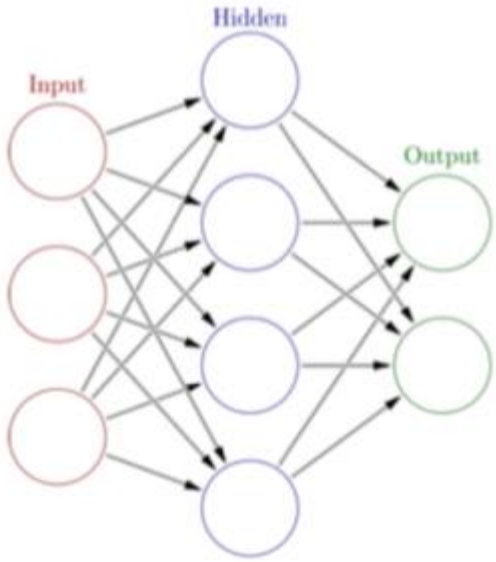

Рисунок 2.1 – Структура нейронної мережи

Мережа складається зі з’єднань, кожне з яких передає вихід нейрону *i* до входу нейрону *j*. В цьому сенсі *i* є попередником *j*, а *j* є наступником *i*. Кожному з’єднанню призначено вагу w_{ij}.

Також нейронна мережа містить в собі функцію поширення, яка обчислює вхід $p_j(t)$ до нейрону j з виходів $o_i(t)$ нейронів-попередників, і зазвичай має вигляд:

$$p_j(t) = \sum_i o_i(t) w_{ij} \, . \qquad (2.1)$$

Правило навчання - це правило або алгоритм, який змінює параметри нейронної мережі відповідно до якогось алгоритму, так що певний вхід в мережу забезпечує відповідний результат. Цей навчальний процес, як правило, передбачає зміну ваги та порогів для мережевих змінних.

Моделі нейронних мереж можна розглядати як прості математичні моделі, що визначають функцію $f: X \rightarrow Y$, або розподіл над X, або над X та Y. Іноді моделі тісно пов’язані з певним правилом інструкції. Загальновживаним терміном "модель ШНМ" є фактично визначення типу таких функцій (де компоненти цього класу отримуються за параметрами, наголошуючи на зв'язках або архітектурних властивостях, таких як кількість нейронів або їх зв'язок).

З математичної точки зору, нейромережеву функцію $f(x)$ визначають як композицію інших функцій $g_i(x)$, які можуть бути розкладені далі на інші функції. Це може зручно представляти як мережеву структуру, де стрілки зображують залежність між функціями. Широко застосованим способом компонування є нелінійна зважена сума, де $f(x) = \text{K}(\sum_i wg(x))$ (що часто називають функцією збудження) є визначеною наперед функцією, такою як, гіперболічний тангенс, або нормована експоненційна функція, або сигмоїдна функція, або пряма функція. Важливою характеристикоюфункції збудження є те, що вона забезпечує плавний перехід при зміні значень входу, тобто, невелика зміна входу призводить до невеликої зміни виходу.

Функціональний підхід – це вхід x перетворено на тривимірний вектор h, який відтак перетворено на двовимірний вектор g, який нарешті перетворювано на f. Цей підхід найчастіше зустрічається в контексті оптимізації.

Ймовірний підхід – це випадкова змінна $F = f(G)$ залежить від випадкової змінної $G = g(H)$, яка залежить від $H = h(X)$, яка залежить від випадкової змінної X. Цей підхід найчастіше зустрічається в контексті моделей графів.

Ці два підходи в основному однакові. У будь-якому випадку, для цієї

конкретної архітектури компоненти окремих шарів не взаємозалежні (наприклад, компоненти g не взаємозалежні від заданого вводу h). Це, звичайно, забезпечує певне прийняття в реалізації. Такі мережі зазвичай називають мережами прямого розповсюдження, оскільки їх графік є спрямованим нециклічним графіком. Сітки з петлями зазвичай називають повторюваними. Такі мережі зазвичай представлені таким чином, що f виявляється самостійним. Однак передбачувана залежність від часу не показана.

Найбільше важливою ознакою нейронних мереж є їх здатність до навчання. Для даної конкретної задачі для розв'язання та класу функцій F навчання означає використання набору спостережень для знаходження $f^* \in F$, яка розв'язує цю задачу в певному оптимальному сенсі.

Це тягне за собою визначення такої функції витрат $C : F \to R$, що, для оптимального роз'язку $f^* = C(f^*) \leq C(f) \forall f \in F$ - тобто, жоден розв'язок не має витрат, менших за витрати оптимального розв'язку.

Функція витрат C є важливим поняттям у навчанні, оскільки вона вимірює, наскільки дане рішення далеке від оптимального рішення проблеми, яку потрібно вирішити. Алгоритми навчання шукають простір для рішень, щоб знайти функцію за мінімально можливою ціною.

Для застосувань, де розв'язок залежить від даних, витрати обов'язково мають бути функцією спостережень, бо інакше модель не матиме зв'язку з даними. Їх часто визначають як статистику, для якої може бути зроблено лише наближення. Як приклад, розглянемо задачу знаходження моделі f, яка зводить, до мінімуму $C = E[f(x) - y)^2]$ для пар даних (x,y), що витягають з певного розподілу D. В практичних ситуаціях ми матимемо лише N зразків з D, відтак для наведеного вище прикладу ми будемо зводити до мінімуму лише. Таким $\hat{C} = \frac{1}{N}\sum_{i=1}^{N}(f(x_1) - y_1)^2$ чином витрати зводяться до мінімуму над вибоіркою даних, а не над усім розподілом.

Коли $N \to \infty$, застосовує різновид інтерактивного машинного навчання, в якому витрати знижуються з кожним побаченим зразком. І, хоча

інтерактивне машинне навчання часто застосовують за незмінного D найкориснішим воно є у випадку, коли цей розподіл повільно змінюється з часом. В нейромережевих методах різновиди інтерактивного машинного навчання часто застосовують для великих наборів даних.

Визначаючи функцію витрат, яка орієнтована на конкретний випадок, ви повинні використовувати конкретну витрату (функцію витрат), оскільки вони мають бажані властивості або тому, що вони природно зумовлені певною формулою проблеми (наприклад, у вирогідному формуліванні, оскільки зворотні можуть буті використані в моделі зворотної ймовірності). Функція витрат залежить від завдання.

Глибинне навчання можна чітко навчити, використовуючи стандартний алгоритм зворотних подій. Зворотне поширення - це метод розрахунку нахилу функції втрат, пов’язаного з даною ситуацією, відносно ваг у штучній нейронній мережі.

Уточнення ваг зворотного поширення можливо здійснювати за допомогою стохастичного градієнтного спуску із застосуванням наступного рівняння:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}} + \xi(t)\,, \qquad (2.2)$$

де $\boldsymbol{\eta}$ – темп навчання;

C – функція витрат;

ε(t) – стохастичний член.

Вибір функції витрат залежить від таких чинників як: тип навчання (кероване, спонтанне, з підкріпленням) та функції збудження. Наприклад, при здійсненні керованого навчання на задачі багатокласової класифікації поширеними варіантами вибору функції збудження та функції витрат є нормована експоненційна функція та функція перехресної ентропії відповідно. Нормалізовану експоненційну функцію визначають як:

$$p_j = \frac{\exp(x_j)}{\sum_k \exp(x_k)}, \tag{2.3}$$

де p – ймовірність класу (вихід вузла j);

x_j та x_k – загальний вхідний сигнал вузлів j та k одного й того ж рівня відповідно.

Перехресну ентропію визначають як:

$$C = -\sum_j d_j \log(p_j), \tag{2.4}$$

де d_j – цільова ймовірність для вузла виходу j;

p_j – вихід ймовірності для j після застосування функції збудження.

Відображає межі об'єкта у вигляді двійкової маски. Вони також використовуються для багатовимірної регресії для підвищення точності позиціонування. Регресія на основі ШНМ може вивчати функції, які включають геометричну інформацію на додаток до функціонування, а також класифікуються. Вони виключають необхідність чіткого моделювання деталей та їх умов. Це допомагає розширити коло предметів, які можна вивчити. Модель складається з декількох шарів, кожен з яких вирівняний як функція збудження при нелінійних змінах. Одні шари згорнуті, інші повністю з’єднані. Кожен торсійний шар має додатковий максимум накопичення. Мережа навчена мінімізувати похибку L2 для прогнозування маски та виконує повний навчальний набір, що включає поля обмежень, які відображаються як маски.

Варіанти множення включають екстремальні навчальні машини, «безпоширні» мережі, тренування без пошуку з звертанням, «без вагові» мережі та не-колективістські нейронні мережі.

Існує три основні парадигми навчання, кожна відповідає певній цілі навчання. Це кероване навчання, спонтанне навчання та посилене навчання (з

підкріпленням).

Кероване навчання використовує набір прикладів пар $(x,y), x \in X, y \in Y$, і має на меті пошук функції $f:X \rightarrow Y$ в дозволеному класі функцій, яка відповідає цим прикладом. Іншими словами, потрібно вивести відображення, на яке натякають ці дані; функцію витрат пов’язано з невідповідністю між нашим відображенням та даними, і вона неявно містить знання про предметну область.

Загальновживаними витратами є похибка - середньоквадратична, яка намагається мінімізувати квадратну помилку значення квадратного кореня між сіткою, $f(x)$, та цільовим значенням y для всіх пар. Поступове зниження цих витрат для типу нейронних мереж, званих багатошаровими персептронами (БШП), забезпечує зворотний алгоритм навчання нейронних мереж.

Завданнями, пов'язаними з парадигмою контрольованого навчання, є розпізнавання зразків (відоме як класифікація) та регресія (відоме як функціональна галузь). Парадигма керованого навчання може також використовуватися в наборах даних (наприклад, розпізнавання рукописного вводу, розпізнавання мови та розпізнавання руху).

2.2 Згорткові нейронні мережі

Аналіз зображень - одне з найдорожчих завдань. Якщо ви подаєте триканальне зображення невеликого розміру 800 x 600, вам знадобиться нейронна мережа з вхідним рівнем 144 000 нейронів. Тому для роботи із зображеннями була створена спеціальна архітектура нейронної мережі, без якої неможливе машинне спуфінг зображень.

Складні нейронні мережі - це архітектура штучних нейронних мереж, запропонована Джеєм Лейконом у 1988 р., Мета якої ефективна з точки зору потужності та точності розпізнавання зображень, є частиною технології глибокого навчання. Використовує різні функції зорової кори для виявлення

так званих простих клітин, які реагують на прямі лінії під різними кутами, та складних клітин, реакція яких включає активацію певної групи простих клітин. Тож ідея нейронних мереж, що падають, полягає в тому, щоб замінити два основні шари шарами згортки та шарами зразка. Структура цієї мережі односпрямована, насправді багатошарова. Для навчання використовуються стандартні методи, наприклад, неправильний метод.

Свою назву архітектура отримала через наявність дії згортки, що означає, що кожен сегмент зображення множиться на матрицю ємності (ядро) елемент за елементом, а результат узагальнюється та документується в місці, подібному до вихідного зображення.

Зазвичай робота із згортанням нейронних мереж трактується як перехід від конкретних особливостей зображення до більш абстрактної, а потім переходить до вибору концепцій на високому рівні. Мережа адаптує та виробляє необхідну ієрархію абстрактних функцій, фільтрує дрібні деталі та підкреслює найважливіше.

У стандартному перцепторі, повністю зв’язаній нейронній мережі, кожен нейрон пов’язаний з усіма нейронами попереднього шару, і кожне з’єднання має свій особистий ваговий коефіцієнт. Реакція судом на нейронній мережі Конволюція використовує обмежену серію невеликих масштабів, які «рухаються» через оброблений шар (спочатку - безпосередньо на вхідному зображенні), поступово створюючи сигнал активації для наступного нейронного шару з подібним розташуванням.

Тобто для різних нейронів у вихідному шарі використовується однакова вагова матриця, яку також називають торсійним ядром. Ця матриця інтерпретується як графічне кодування кожної функції, наприклад, наявність скісної риски під певним кутом. Тоді наступний шар, який є результатом крутильного ефекту такої вагової матриці, показує наявність цієї функції в оброблюваному шарі та її координатах і створює те, що називається картою функції. У спазматичній нейронній мережі масштаб не один, а ціла серія, яка кодує елементи на зображенні (наприклад, лінії та дуги під різними кутами).

При цьому такі крутильні ядра не визначаються дослідником, а формуються самостійно шляхом вивчення мережі класичним методом поширення помилок. Перехід до кожної групи шкал створює власну копію карти функцій, роблячи нейронну мережу багатоканальною (багато незалежних функціональних карт в одному шарі). Слід також зазначити, що коли ми шукаємо команду з кількістю ваг, вона зазвичай проходить не весь крок (розмір масиву), а невелику відстань. Наприклад, при ваговій матриці 5 × 5 вона передається одним або двома нейронами (пікселями) замість п’яти, щоб не «перемикати» бажану функцію.

Розглянемо типову структуру згортки нейронної мережі більш докладно. Мережа складається з великої кількості команд. Після першого шару (вхідного зображення) сигнал проходить через ряд згорткових шарів, де фактичне поділ і субдискретизація чергуються. Заміна шарів дає можливість створювати "карти функцій" карт функцій, у кожному наступному шарі карта стає меншою, але кількість каналів збільшується. На практиці це означає можливість виявлення складних ієрархій властивостей. Зазвичай, пройшовши кілька шарів, карта функцій вироджується у вектор або навіть скаляр, але такі карти функцій стають значно більшими. Крім того, декілька шарів повністю підключеної нейронної мережі встановлені в кінці рівня згортки мережі до входу, де використовуються найновіші карти функцій.

Згортковий шар є основним блоком спазматичної нейронної мережі. Він містить власний фільтр для кожного каналу, його ядро розкладання обробляє попередній шар фракціями (узагальнює результати матричного продукту для кожної фракції). Ваги ядра згортки (маленька матриця) невідомі і визначаються під час тренування.

Характерним для згортокового шару є відносно невелика кількість параметрів, які встановлюються під час тренування. Наприклад, якщо вихідне зображення має розмір 100 х 100 пікселів у трьох каналах (тобто 30000 вхідних нейронів), а торсійний шар використовує фільтри з ядром 3 х 3 пікселі з 6 вихідними каналами, лише в процесі навчання. 9 вагових

коефіцієнтів для всіх комбінацій каналів, тобто 9 × 3 × 6 = 162, але цей шар у цьому випадку вимагає пошуку лише 162 параметрів, які є менш значущими, ніж кількість параметрів, необхідних у повністю підключеній нейронній мережі (рисунок 2.2).

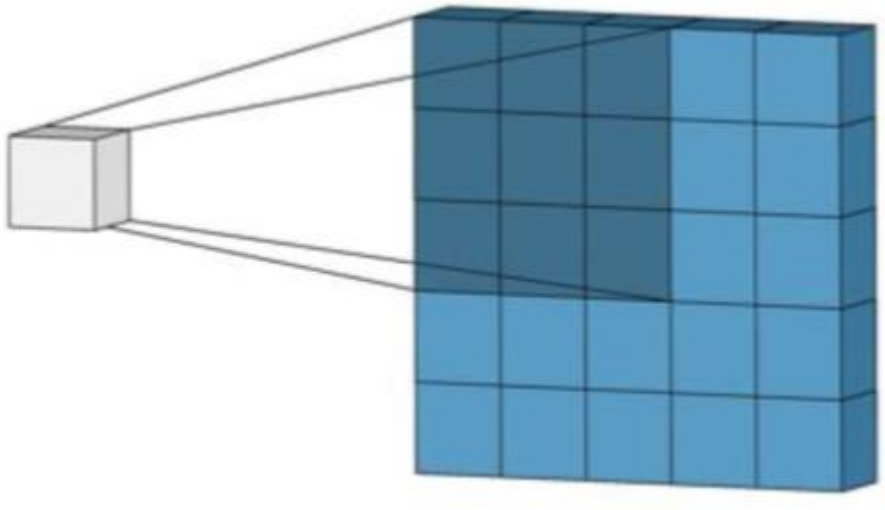

Рисунок 2.2 – Двовимірна згорткова нейронна мережа

Двовимірна згортка є досить простою операцією: вона починається в серцевині, матриці ваги. Це ядро "ковзає" по двовимірному зображенню, виконує поетапну операцію множення з тією частиною вхідних даних, в якій він знаходиться в даний момент, а потім підсумовує всі отримані значення до вихідного пікселя.

Ядро повторює цю процедуру в будь-якому положенні, на якому вона «ковзає», перетворюючи двовимірний масив в інший двовимірний масив властивостей. Вихідні символи - це зважені суми (де ваги - це значення самого ядра) вхідних символів, приблизно в тому самому місці, що і вихідний піксель у вхідному шарі (рисунок 2.3).

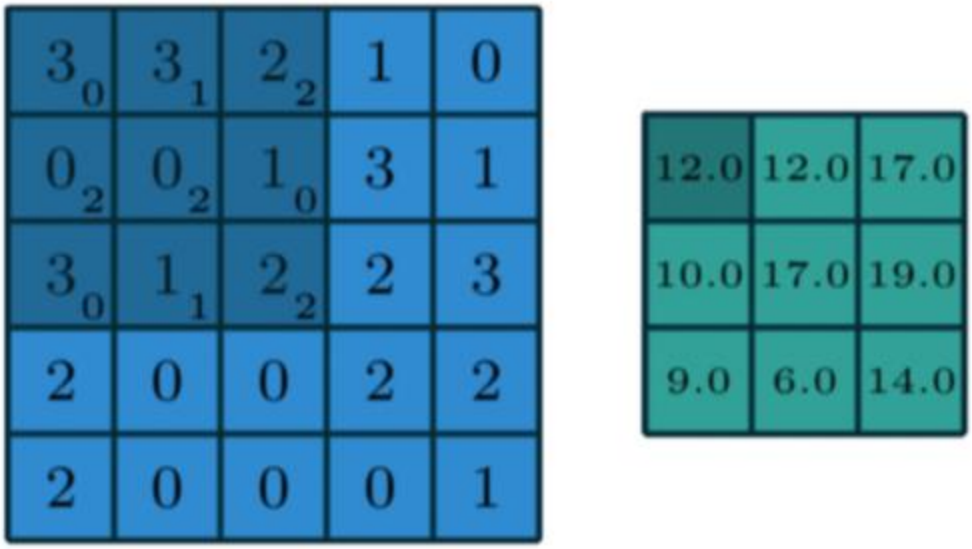

Рисунок 2.3 – Операція згортки

Не має значення, чи вхідна функція потрапляє "приблизно в одне і те ж місце", відповідно визначається, знаходиться вона в області ядра, що генерує вихід, чи ні. Це означає, що розмір ядра нейромережі ємності визначає кількість функцій, які необхідно об'єднати для досягнення нової функції на виході.

У наведеному вище прикладі ми маємо 5 × 5 = 25 символів на вході та 3 × 3 = 9 символів на виході. Для стандартного шару, який повністю зв’язаний, ми маємо матрицю ваги з 25 × 9 = 225 параметрів, і кожна вихідна функція є зваженою сумою всіх функцій на вході. За допомогою Convolution ви можете виконати таку операцію лише з 9 параметрами, оскільки кожна функція на виході досягається не аналізом усіх функцій на вході, а лише одним входом, який знаходиться "приблизно в тому самому місці".

Після декількох переходів згортання та відшаровування зображень система відновлюється із певної сітки пікселів високої роздільної здатності до більш абстрактних карт функцій, які зазвичай збільшують кількість каналів у кожному наступному шарі та зменшують розмір зображення в кожному каналі. Нарешті, існує велика кількість каналів, які зберігають невелику кількість даних (навіть параметр), які трактуються як абстрактні

поняття, виявлені з вихідного зображення.

Ці дані інтегруються та передаються у звичайну і повністю підключену нейронну мережу, яка може складатися з декількох шарів. У цьому випадку повністю сусідні шари втрачають всю просторову структуру пікселів і мають відносно невеликий розмір (відносно кількості пікселів на вихідному зображенні).

На практиці більшість вхідних зображень мають 3 канали, і чим глибше ви знаходитесь у мережі, тим більша кількість. Як правило, канали слід розглядати як "погляди" у всій картині, надаючи більше значення одному аспекту, а менше іншому.

У більшості випадків ми маємо справу з зображеннями RGB з трьома каналами (рисунок 2.4).

Істотні відмінності між термінами стають необхідними у випадку одного каналу, коли терміни "фільтр" і "ядро" взаємозамінні, вони, як правило, різні. Кожен фільтр насправді є сукупністю ядер, і кожен окремий вхідний канал для цього шару має ядро, кожне ядро є унікальним.

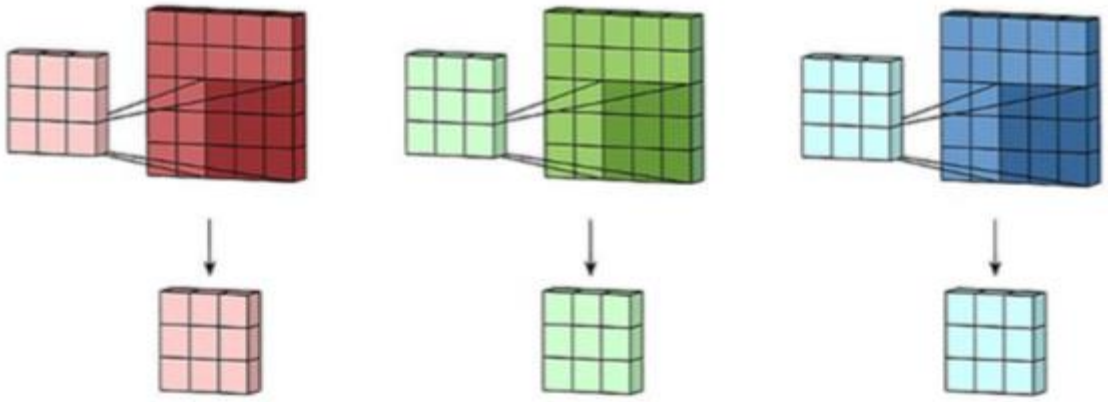

Рисунок 2.4 – Багатоканальне зображення

Кожен фільтр у згортковому шарі створює лише один вихідний канал, і вони роблять це наступним чином: кожен із фільмів "переглядає" відповідні вхідні канали, створюючи сфабриковану версію кожного (рисунок 2.5).

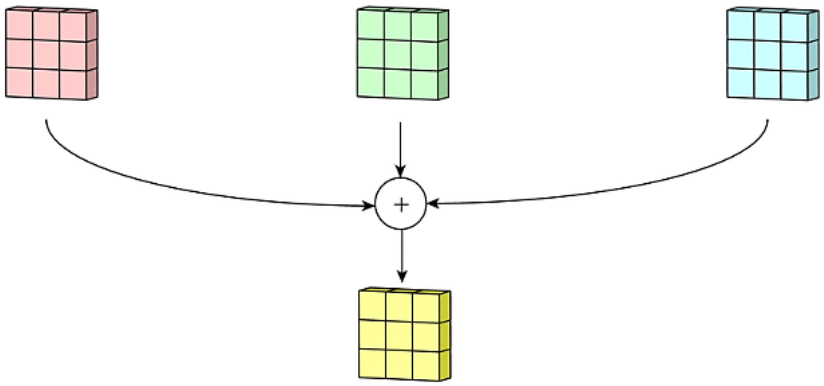

Рисунок 2.5 – Операція згортки на триканальному зображенні

Потім кожна з оброблених версій каналу додається для створення єдиного каналу. Ядра кожного фільтра створюють версію кожного каналу, а фільтр в цілому створює загальний вихідний канал (рисунок 2.6). Принцип заміни полягає в тому, що кожен вихідний файл має власний період зміщення. Зміщення додається до вихідного каналу для створення кінцевого вихідного каналу.

Результат для декількох фільтрів однаковий: кожен фільтр обробляє вхід з різним набором ядер і скалярних зсувів у процесі, описаному вище, створюючи вихідний канал. Потім вони об'єднуються для отримання загального виходу, де кількість вихідних каналів дорівнює кількості фільтрів. Зазвичай це стосується нелінійності перед тим, як надсилати вхідні дані з іншого шару обмотки, а потім повторювати цей процес.

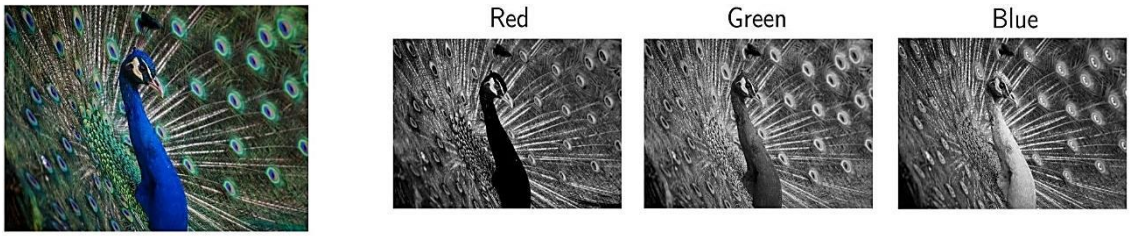

Рисунок 2.6 – Об’єднання операцій по всім каналам

Навіть з описаною механікою згорткового шару все ще важко пов’язати це з нейронною мережею прямого розповсюдження, що все ще не пояснює, чому локони розширюються і працюють набагато краще із зображеннями. Скажімо, у нас є вхід 4 × 4, і ми хочемо зробити це мережею 2 × 2. Якби ми використовували пряму мережу, ми перетворили б вхід 4 × 4 у вектор довжиною 16 та пропустили б його через добре зв’язаний шар з 16 входами та 4 виходами. Ви можете уявити матрицю ваги W на шар (рисунок 2.7).

І хоча спочатку операції ядерної згортки можуть здаватися дещо дивними, вони все ще є лінійними перетвореннями з подібною матрицею переходів.

$$\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} & w_{1,6} & w_{1,7} & w_{1,8} & w_{1,9} & w_{1,10} & w_{1,11} & w_{1,12} & w_{1,13} & w_{1,14} & w_{1,15} & w_{1,16} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} & w_{2,6} & w_{2,7} & w_{2,8} & w_{2,9} & w_{2,10} & w_{2,11} & w_{2,12} & w_{2,13} & w_{2,14} & w_{2,15} & w_{2,16} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} & w_{3,6} & w_{3,7} & w_{3,8} & w_{3,9} & w_{3,10} & w_{3,11} & w_{3,12} & w_{3,13} & w_{3,14} & w_{3,15} & w_{3,16} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} & w_{4,6} & w_{4,7} & w_{4,8} & w_{4,9} & w_{4,10} & w_{4,11} & w_{4,12} & w_{4,13} & w_{4,14} & w_{4,15} & w_{4,16} \end{bmatrix}$$

Рисунок 2.7 – Вагова матриця

Якщо ми маємо використовувати ядро K рзміром 3 на іншому вході 4 × 4, щоб отримати вихід 2 × 2, відповідна матриця (рисунок 2.8). Хоча вищевказана матриця подібна до матриці переходу, фактична операція зазвичай використовується шляхом множення іншої матриці.

$$\begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{1,1} & k_{1,2} & k_{1,3} & 0 & k_{2,1} & k_{2,2} & k_{2,3} & 0 & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix}$$

Рисунок 2.8 – Матриця переходу

Згортка - це, як правило, лінійне перетворення, але в той же час і зовсім

інша форма перетворення. Для матриці з 64 елементами 9 параметрів повторно використовуються кілька разів. Кожен вузол виводу отримує лише певну кількість входів (які знаходяться в ядрі). Немає взаємодії з іншими входами, оскільки вага для них становить 0.

Операцію згортки можна представити як вищий пріоритет для матриць ваги. У цьому контексті пріоритет означає заздалегідь визначені параметри мережі. Наприклад, при використанні попередньо обробленої моделі класифікації зображень також використовуються попередні мережеві параметри, такі як перевага зображення для жорсткого покриття. Обидва ефективні порівняно з їхніми альтернативами.

Навчання передачі є більш ефективним, ніж випадкова ініціалізація, оскільки вам потрібно лише оптимізувати параметри повністю зв’язаного кінцевого шару, а це означає, що ви зможете отримати кращі результати за допомогою декількох десятків зображень на клас.

3 МОДЕЛІ МАШИННОГО НАВЧАННЯ

3.1 Фундаментальна сегментація моделей машинного навчання

Всі моделі машинного навчання поділяються (рисунок 3.1) на навчання з учителем (supervised) і без вчителя (unsupervised). В першу категорію входять регрессионная і класифікаційна моделі. Розглянемо значення цих термінів і входять в ці категорії моделі.

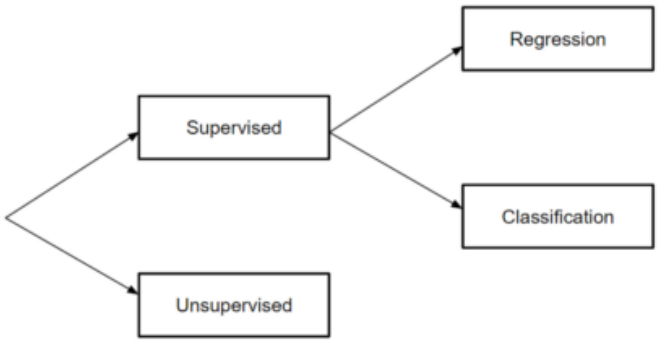

Рисунок 3.1 – Моделі машинного навчання

3.2 Навчання з вчителем

Навчання з вчиталем являє собою вивчення функції, яка перетворює вхідні дані у вихідні на основі прикладів пар введення-виведення.

Наприклад, з набору даних з двома змінними: вік (вхідні дані) і зростання (вихідні дані), можна реалізувати модель навчання для прогнозування зростання людини на основі його віку (рисунок 3.2).

Навчання з учителем підрозділяється на дві підкатегорії: регресія і класифікація.

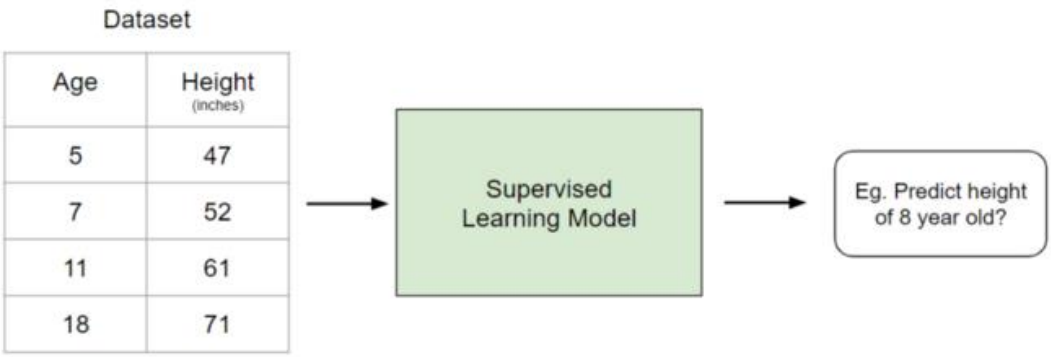

Рисунок 3.2 – Приклад навчання з учителем

3.2.1 Регресія

У регресійних моделях висновок є безперервним. Нижче наведені деякі з найбільш поширених типів регресійних моделей.

Завдання лінійної регресії (рисунок 3.3) полягає в знаходженні лінії, яка найкращим чином відповідає даним. Розширення лінійної регресії включають множинну лінійну регресію (наприклад, пошук найбільш підходящої площині) і поліноміальних регресію (наприклад, пошук найбільш підходящої кривої).

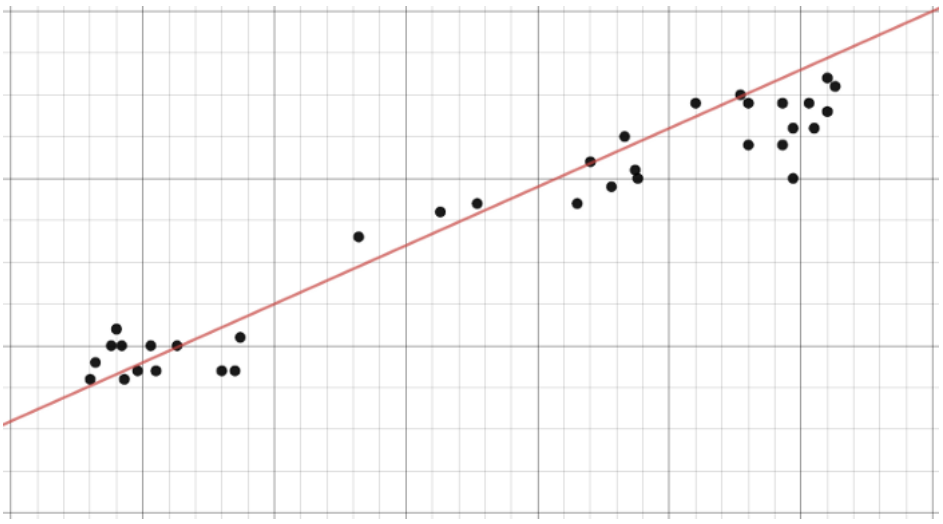

Рисунок 3.3 – Приклад лінійної регресії

Дерево рішень - популярна модель, яка використовується в дослідженні операцій, стратегічному плануванні та машинному навчанні. Кожен прямокутник вище називається вузлом. Чим більше вузлів, тим точнішим буде дерево рішень. Останні вузли, в яких приймається рішення, називаються листям дерева. Дерева рішень (рисунок 3.4) інтуїтивні і прості в створенні, проте не надають точні результати.

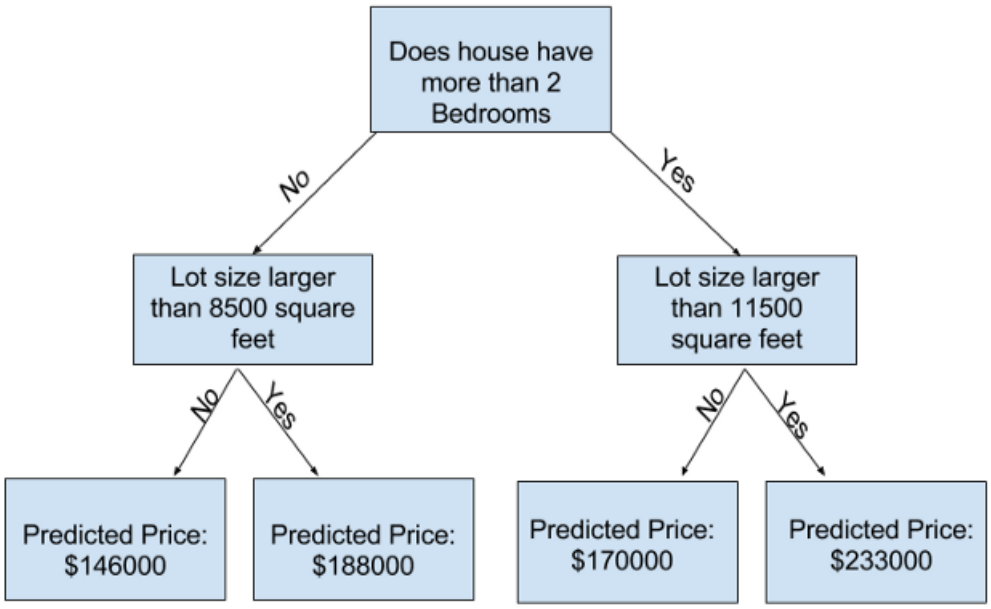

Рисунок 3.4 – Приклад дерев рішень

Випадковий ліс - це техніка ансамблю методів, заснована на деревах рішень. Випадкові лісу включають створення декількох дерев рішень з використанням первинних наборів даних і випадковий вибір поднабора змінних на кожному етапі. Потім модель вибирає моду (значення, яке зустрічається частіше за інших) з усіх прогнозів кожного дерева рішень. Який у цьому сенс? Модель "перемоги більшості" знижує ризик помилки окремого дерева.

Наприклад (рисунок 3.5), у нас є одне дерево рішень (третє), яке передбачає 0. Однак якщо покладатися на моду всіх 4 дерев, прогнозоване значення дорівнюватиме 1. У цьому полягає перевага випадкових лісів.

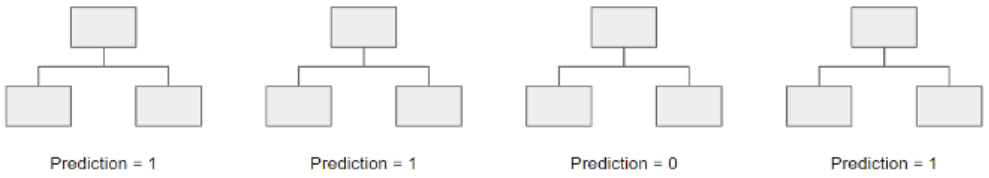

Рисунок 3.5 – Приклад випадковий ліс

Нейронна мережа (рисунок 3.6) - це багатошарова модель, влаштована по системі людського мозку. Як і нейрони в нашому мозку, кола вище представляють вузли. Синім позначено шар вхідних даних, чорним - приховані шари, а зеленим - шар вихідних даних. Кожен вузол в прихованих шарах являє функцію, через яку проходять вхідні дані, що призводять до виходу в зелених колах.

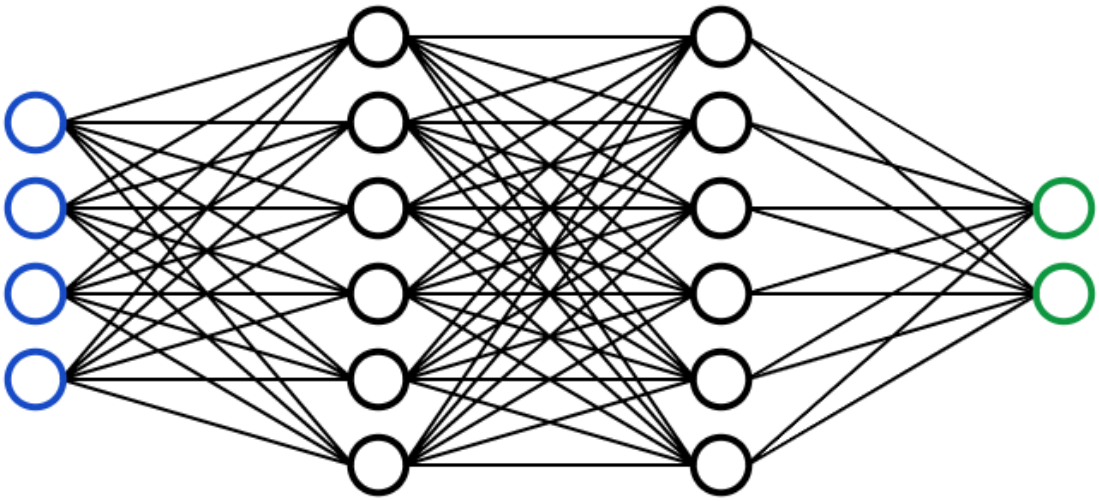

Рисунок 3.6 – Приклад нейронної мережі

3.2.2 Класифікація

У класифікаційних моделях висновок є дискретним. Нижче наведені деякі з найбільш поширених типів класифікаційних моделей.

Логістична регресія аналогічна лінійної регресії, але використовується

для моделювання ймовірності обмеженого числа результатів, зазвичай двох. Логістичне рівняння створюється таким чином, що вихідні значення можуть знаходитися тільки між 0 і 1 (рисунок 3.7):

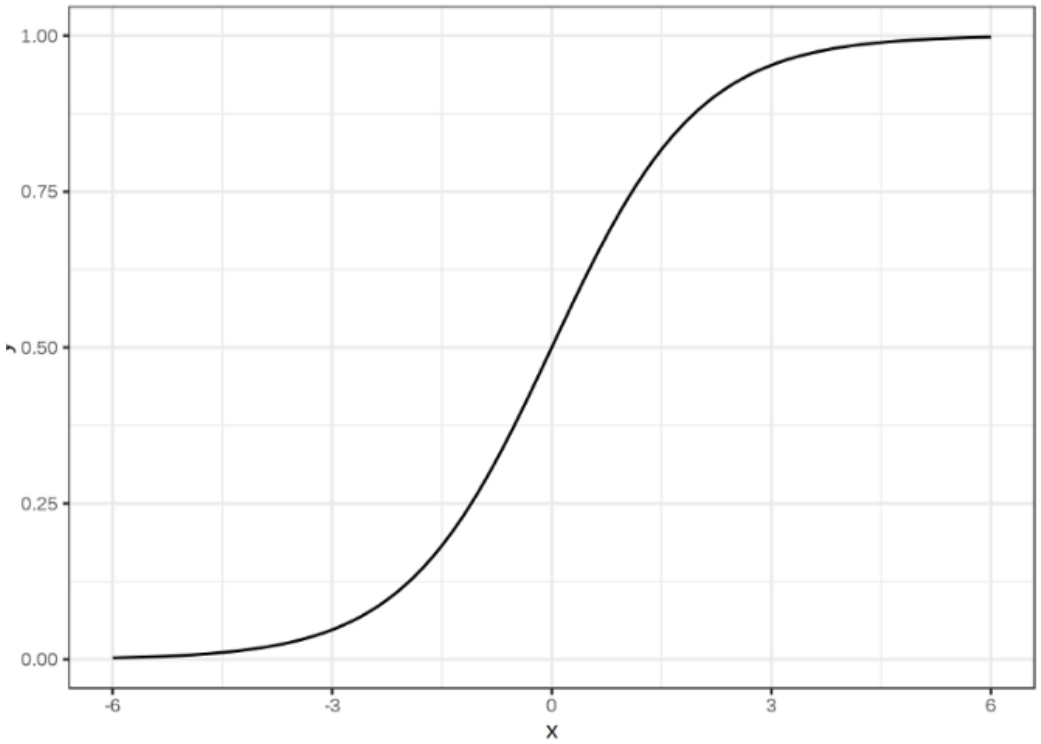

Рисунок 3.7 – Логістична регресія

Метод опорних векторів (рисунок 3.8) - це класифікаційний метод навчання з учителем, досить складний, але досить інтуїтивний на базовому рівні.

Припустимо, що існує два класи даних. Метод опорних векторів знаходить гіперплоскость або кордон між двома класами даних, яка максимізує різницю між двома класами. Є безліч площин, які можуть розділити два класи, але тільки одна з них максимізує різницю або відстань між класами.

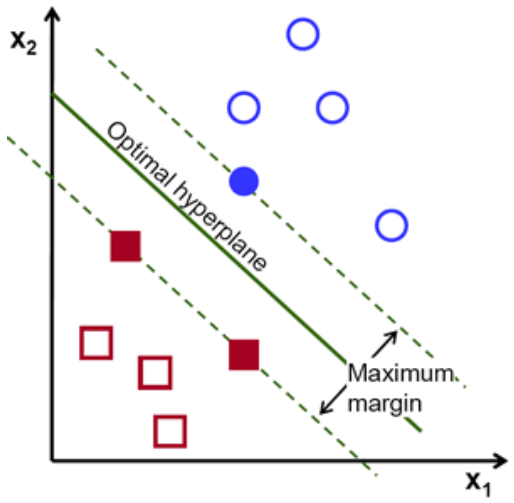

Рисунок 3.8 – Метод опорних векторів

Наївний Байес - ще один популярний класифікатор, який використовується в науці про дані. Його ідея лежить в основі теореми Байєса.

Незважаючи на ряд нереалістичних припущень, зроблених щодо наявного Байеса (звідси і назва "наївний"), він не тільки довів свою ефективність в більшості випадків, але і відносно простий в побудові.

3.3 Навчання без вчителя

На відміну від навчання з учителем, навчання без вчителя використовується для того, щоб зробити висновки і знайти шаблони з вхідних даних без відсилань на помічені результати (рисунок 3.9). Два основні методи, які використовуються в навчанні без учителя, включають кластеризацію і зниження розмірності.

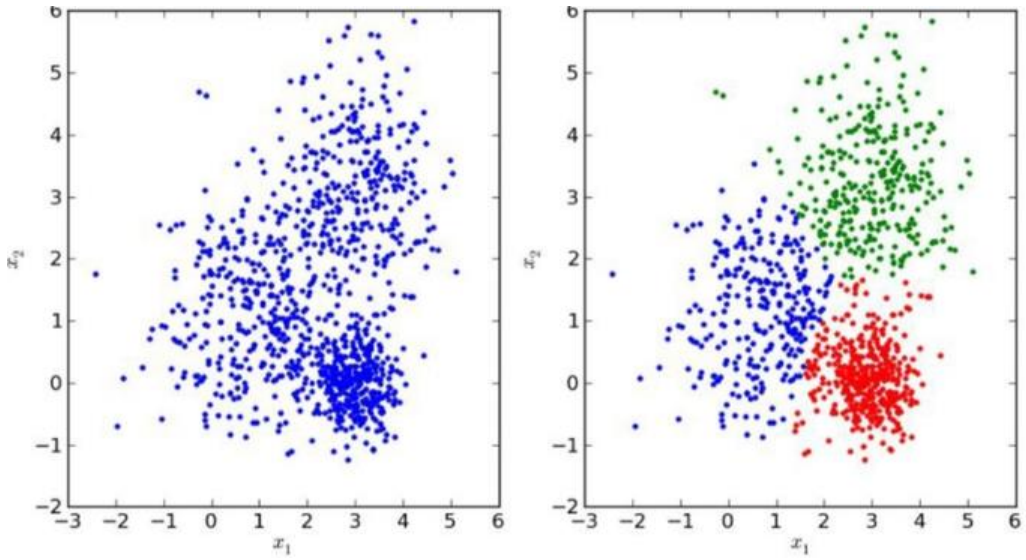

Рисунок 3.9 – Навчання без вчителя

3.3.1 Кластеризація

Кластеризація (рисунок 3.10) - це техніка навчання без учителя, яка включає в себе групування або кластеризацію точок даних. Найчастіше вона використовується для сегментації споживачів, виявлення шахрайства та класифікації документів.

Поширені методи кластеризації включають кластеризацію за допомогою k-середніх, ієрархічну кластеризацію, зрушення середнього значення і кластеризації на основі щільності. У кожного з них є свій спосіб пошуку кластерів, проте всі вони призначені для досягнення одного результату.

3.3.2 Зниження розмірності

Зниження розмірності - це процес зменшення числа розглянутих випадкових змінних шляхом отримання набору головних змінних. Простіше кажучи, це процес зменшення розміру набору ознак (зменшення кількості ознак). Більшість методів зниження розмірності можуть бути класифіковані

як відбір або витяг ознак.

Популярний метод зниження розмірності називається методом головних компонент (PCA). Він являє собою проектування багатовимірних даних (наприклад, 3 вимірювання) в менший простір (наприклад, 2 вимірювання). Це призводить до зменшення розмірності даних (2 вимірювання замість 3) при збереженні всіх вихідних змінних в моделі.

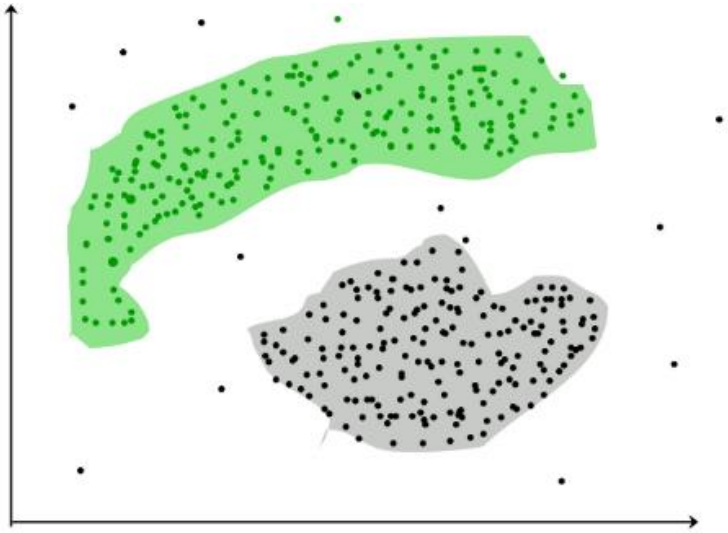

Рисунок 3.10 – Кластеризація

4 ГЕНЕРАТОР ПРАВДОПОДІБНИХ ЗОБРАЖЕНЬ ДЛЯ КЛАСИФІКАЦІЙНИХ МОДЕЛЕЙ

4.1 TensorFlow.js

TensorFlow - відкрита програмна бібліотека для машинного навчання цілій низці задач, розроблена компанією Google для задоволення її потреб у системах, здатних будувати та тренувати нейронні мережі для виявлення та розшифровування образів та кореляцій, аналогічно до навчання й розуміння, які застосовують люди. Її наразі застосовують як для досліджень, так і для розробки продуктів Google, часто замінюючи на його ролі її закритого попередника, DistBelief. TensorFlow було початково розроблено командою Google Brain для внутрішнього використання в Google, поки її не було випущено під відкритою ліцензією Apache 2.0 9 листопада 2015 року.

Платформа спочатку розроблена командою Google Brain і використовуються в сервісах Google для розпізнавання мови, виділення облич на фотографіях, визначення схожості зображень, відсіювання спаму в Gmail, підбору новин у Google News і організації перекладу з урахуванням смислу. Розподілені системи машинного навчання можна створювати на типовому обладнанні, завдяки вбудованій підтримці в TensorFlow рознесення обчислень на кілька CPU або GPU.

Серед застосувань, для яких TensorFlow є основою, є програмне забезпечення автоматизованого опису зображень, таке як DeepDream. 26 жовтня 2015 року Google офіційно реалізувала RankBrain, який підтримує TensorFlow. RankBrain тепер обробляє суттєве число пошукових записів, замінюючи та доповнюючи традиційні статичні алгоритми на основі результатів пошуку.

Іншими застосуванням є використання у складі програм FakeApp з метою безшовного поєднання фото- та відеозображень для створення

підробних, але правдоподібних відео, відомих під назвою Deepfake.

TensorFlow надає бібліотеку готових алгоритмів чисельних обчислень, реалізованих через графи потоків даних (data flow graphs). Вузли в таких графах реалізують математичні операції або точки входу/виводу, в той час як ребра графа представляють багатовимірні масиви даних (тензори), які перетікають між вузлами. Вузли можуть бути закріплені за обчислювальними пристроями і виконуватися асинхронно, паралельно обробляючи разом все підходящі до них тензори, що дозволяє організувати одночасну роботу вузлів в нейронної мережі за аналогією з одночасною активацією нейронів в мозку.

Google TensorFlow бібліотека відкритого джерела машинного навчання була розширена на JavaScript з Tensorflow.js, бібліотекою JavaScript для розгортання моделей машинного навчання в браузері.

Прискорена бібліотека WebGL, Tensorflow.js також працює з сервером JavaScript на сервері Node.js і є частиною екосистеми TensorFlow (рисунок 4.1). З машинним навчанням безпосередньо в браузері немає потреби в драйверах; розробники можуть просто запустити код.

Проект, який містить екосистему інструментів JavaScript, розвинувся з бібліотеки Deeplearn.js для машинного навчання на основі браузера; Deeplearn.js тепер відомий як Core Tensorflow.js.

API TensorFlow.js можна використовувати для побудови моделей за допомогою бібліотеки лінійної алгебри JavaScript JavaScript або шарів API більш високого рівня. Перетворювачі моделі TensorFlow.js можуть запускати існуючі моделі в браузері або під Node.js. Існуючі моделі можна перекваліфікувати за допомогою даних датчиків, підключених до браузера.

Тензор служить центральною одиницею даних. Крім того, для побудови нейронних мереж включено високорівневий API, натхненний Keras.

Але TensorFlow.js не єдина бібліотека JavaScript, побудована для нейронних мереж; TensorFire, побудований студентами MIT, виконує нейронні мережі на веб-сторінці.

Tensorflow.js має API, схожий на API Python Tensorflow. Але JavaScript API ще не підтримує всі функціональні можливості API Python. Будівельники Tensorflow.js зобов'язуються досягти паритету, де це має сенс, але хочете надати ідіоматичний JavaScript API. TensorFlow з WebGL також працює на швидкості від 50 до 60% швидкості API TensorFlow Python з бібліотекою AVX.

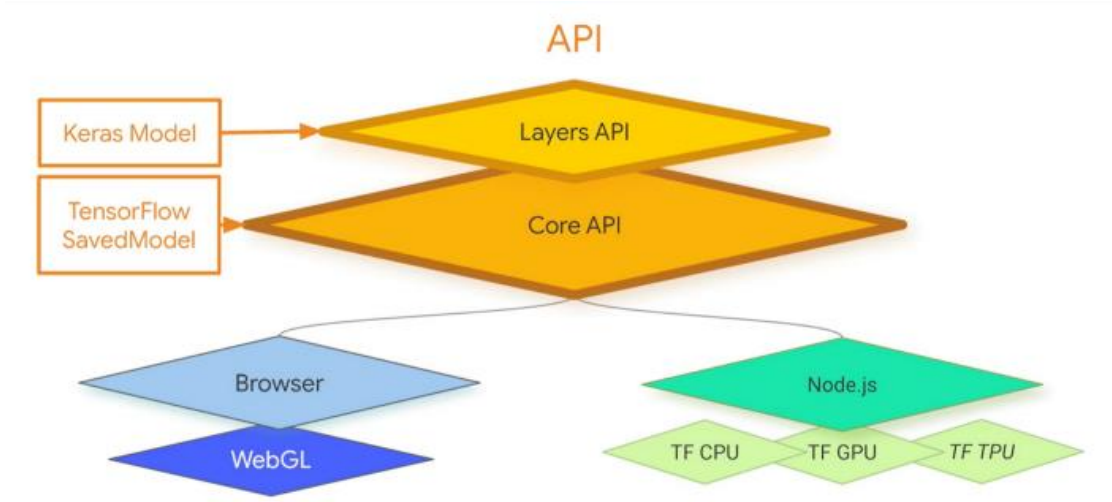

Рисунок 4.1 – Дизайн та архітектура TensorFlow.js

4.2 Огляд програми

4.2.1 Вхідні данні

У ході роботи було створено нейронну мережу, яка здатна виявляти та класифікувати рукописні цифри. З цією метою ми використовували набір даних MNIST (рисунок 4.2). Це добре відомий набір даних у світі нейронних мереж. Він розширює свій попередник NIST і має навчальний набір із 60 000 зразків та набір для тестування з 10 000 зображень від руки написаних цифр. Усі цифри були нормалізовані за розмірами та відцентровані. Розмір зображень також встановлений на 28 × 28 пікселів. Ось чому цей набір даних настільки популярний.

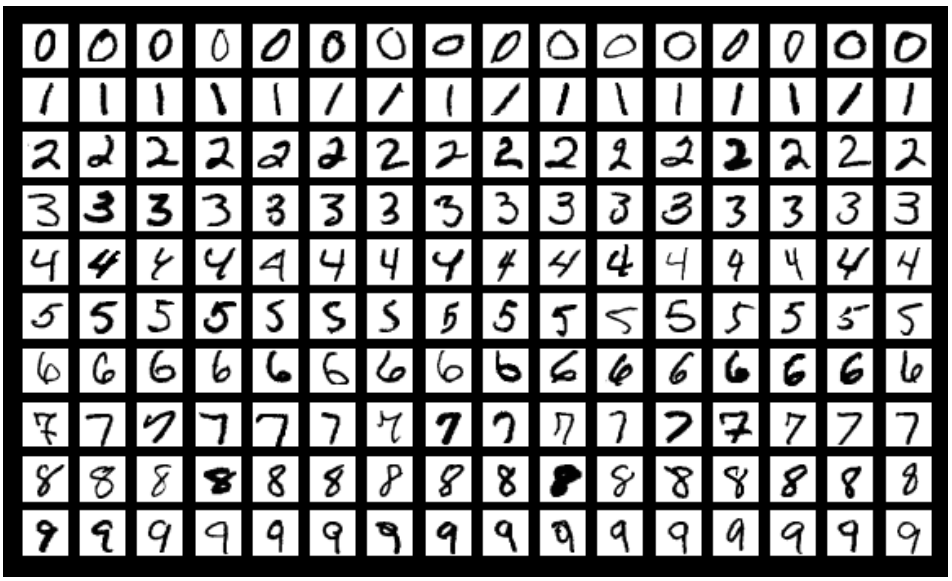

Рисунок 4.2 – Зразки набору даних MNIST

Використовуючи згорткові нейронні мережі, можна отримати майже людські результати. Що стосується точності прогнозування на цьому наборі даних, то в основному вони дають правильні результати в 99,79% випадків.

4.2.2 Завантаження даних

Для швидшого завантаження вищезазначених даних було використано готовий файл правдоподібних зображень. Завантаження данних для навчання нейронної мережі наведно у прикаді 4.1.

```
      const IMAGE_SIZE = 784;
      const NUM_CLASSES = 10;
      const NUM_DATASET_ELEMENTS = 65000;

      const TRAIN_TEST_RATIO = 5 / 6;

      const     NUM_TRAIN_ELEMENTS     =     Math.floor(TRAIN_TEST_RATIO     *
NUM_DATASET_ELEMENTS);
      const NUM_TEST_ELEMENTS = NUM_DATASET_ELEMENTS - NUM_TRAIN_ELEMENTS;

      const MNIST_IMAGES_SPRITE_PATH =
          'https://storage.googleapis.com/learnjs-data/model-
builder/mnist_images.png';
      const MNIST_LABELS_PATH =
          'https://storage.googleapis.com/learnjs-data/model-
builder/mnist_labels_uint8';

      /**
```

```
       * A class that fetches the sprited MNIST dataset and returns shuffled
batches.
       *
       * NOTE: This will get much easier. For now, we do data fetching and
       * manipulation manually.
       */
      export class MnistData {
        constructor() {
          this.shuffledTrainIndex = 0;
          this.shuffledTestIndex = 0;
        }

        async load() {
          // Make a request for the MNIST sprited image.
          const img = new Image();
          const canvas = document.createElement('canvas');
          const ctx = canvas.getContext('2d');
          const imgRequest = new Promise((resolve, reject) => {
            img.crossOrigin = '';
            img.onload = () => {
              img.width = img.naturalWidth;
              img.height = img.naturalHeight;

              const datasetBytesBuffer =
                  new ArrayBuffer(NUM_DATASET_ELEMENTS * IMAGE_SIZE * 4);

              const chunkSize = 5000;
              canvas.width = img.width;
              canvas.height = chunkSize;

              for (let i = 0; i < NUM_DATASET_ELEMENTS / chunkSize; i++) {
                const datasetBytesView = new Float32Array(
                    datasetBytesBuffer, i * IMAGE_SIZE * chunkSize * 4,
                    IMAGE_SIZE * chunkSize);
                ctx.drawImage(
                    img,  0,  i  *  chunkSize,  img.width,  chunkSize,  0,  0,
img.width,
                    chunkSize);

                const  imageData  =  ctx.getImageData(0,  0,  canvas.width,
canvas.height);

                for (let j = 0; j < imageData.data.length / 4; j++) {
                  // All channels hold an equal value since the image is
grayscale, so
                  // just read the red channel.
                  datasetBytesView[j] = imageData.data[j * 4] / 255;
                }
              }
              this.datasetImages = new Float32Array(datasetBytesBuffer);

              resolve();
            };
            img.src = MNIST_IMAGES_SPRITE_PATH;
          });

          const labelsRequest = fetch(MNIST_LABELS_PATH);
          const [imgResponse, labelsResponse] =
              await Promise.all([imgRequest, labelsRequest]);

          this.datasetLabels          =          new          Uint8Array(await
labelsResponse.arrayBuffer());

          // Create shuffled indices into the train/test set for when we
```

```
select a
          // random dataset element for training / validation.
          this.trainIndices                                                    =
tf.util.createShuffledIndices(NUM_TRAIN_ELEMENTS);
          this.testIndices = tf.util.createShuffledIndices(NUM_TEST_ELEMENTS);

          // Slice the the images and labels into train and test sets.
          this.trainImages =
              this.datasetImages.slice(0, IMAGE_SIZE * NUM_TRAIN_ELEMENTS);
          this.testImages     =     this.datasetImages.slice(IMAGE_SIZE     *
NUM_TRAIN_ELEMENTS);
          this.trainLabels =
              this.datasetLabels.slice(0, NUM_CLASSES * NUM_TRAIN_ELEMENTS);
          this.testLabels =
              this.datasetLabels.slice(NUM_CLASSES * NUM_TRAIN_ELEMENTS);
        }

        nextDataBatch(batchSize, test = false) {
          if(test)
              return this.nextBatch(
                  batchSize, [this.trainImages, this.trainLabels], () => {
                    this.shuffledTrainIndex =
                        (this.shuffledTrainIndex        +        1)        %
this.trainIndices.length;
                    return this.trainIndices[this.shuffledTrainIndex];
                  });
          else
              return        this.nextBatch(batchSize,        [this.testImages,
this.testLabels], () => {
                  this.shuffledTestIndex =
                      (this.shuffledTestIndex + 1) % this.testIndices.length;
                  return this.testIndices[this.shuffledTestIndex];
                });
        }

        nextBatch(batchSize, data, index) {
          const batchImagesArray = new Float32Array(batchSize * IMAGE_SIZE);
          const batchLabelsArray = new Uint8Array(batchSize * NUM_CLASSES);

          for (let i = 0; i < batchSize; i++) {
            const idx = index();

            const image =
                data[0].slice(idx   *   IMAGE_SIZE,   idx   *   IMAGE_SIZE   +
IMAGE_SIZE);
            batchImagesArray.set(image, i * IMAGE_SIZE);

            const label =
                data[1].slice(idx   *   NUM_CLASSES,   idx   *   NUM_CLASSES   +
NUM_CLASSES);
            batchLabelsArray.set(label, i * NUM_CLASSES);
          }

          const xs = tf.tensor2d(batchImagesArray, [batchSize, IMAGE_SIZE]);
          const    labels    =    tf.tensor2d(batchLabelsArray,    [batchSize,
NUM_CLASSES]);

          return {xs, labels};
        }
      }
```

Приклад 4.1 – Формування вхідних данних (файл data.js)

Ось кілька основних моментів цього класу MnistData. Зображення та ярлики цих зображень завантажуються у поля trainImages, testImages, trainLabels та testLabels. Цю функцію потрібно викликати спочатку. Після цього ми можемо використовувати методи nextDataBatch для отримання пакетів даних із цих наборів даних. Знизу ці методи викликають функцію nextBatch з різними зображеннями та мітками. Ця функція також перетворює дані в тензори, які необхідні для обробки.

4.2.3 Реалізація

Почнемо з файлу index.html. У ньому ініціалузіється TensorFlow.js, завантажуються дані з файлу data.js та підключається головний скрипт програми.

Ось як виглядає основна функція запуску (приклад 4.2):

```
async function run() {
    const data = await getData();

    await displayDataFunction(data, 30);

    const model = createModel();
    tfvis.show.modelSummary({name: 'Model Architecture'}, model);

    await trainModel(model, data, 20);

    await evaluateModel(model, data);
}
```

Приклад 4.2 – Функція запуску (файл script.js)

На початку ми завантажуємо дані за допомогою функції getData (приклад 4.3):

```
async function getDataFunction() {
    var data = new MnistData();
    await data.load();
    return data;
}
```

Приклад 4.3 – Завантаження данних (файл script.js)

Спочатку створюється об’єкт класу MnistData. Цей клас знаходиться у файлі data.js. Потім викликається згадану функцію навантаження. Він ініціалізує властивості створеного об’єкта. Після цього повертаємо цей об’єкт і використовувати його для відображення вхідних даних у браузері за допомогою функції displayData (приклад 4.4):

```
async function singleImagePlot(image)
{
  const canvas = document.createElement('canvas');
  canvas.width = 28;
  canvas.height = 28;
  canvas.style = 'margin: 4px;';
  await tf.browser.toPixels(image, canvas);
  return canvas;
}

async function displayDataFunction(data, numOfImages = 10) {

  const inputDataSurface =
    tfvis.visor().surface({ name: 'Input Data Examples', tab: 'Input
Data'});

  const examples = data.nextDataBatch(numOfImages, true);

  for (let i = 0; i < numOfImages; i++) {
    const image = tf.tidy(() => {
      return examples.xs
        .slice([i, 0], [1, examples.xs.shape[1]])
        .reshape([28, 28, 1]);
    });

    const canvas = await singleImagePlot(image)
    inputDataSurface.drawArea.appendChild(canvas);

    image.dispose();
  }
}
```

Приклад 4.4 – Завантаження зображень (файл script.js)

У цій функції спочатку створюється нова вкладку під назвою InputData. Потім отримується пакет тестових даних за допомогою функції nextDataBatch із класу MnistData. Потім перебирається ці зображення і перетворюємо їх з тензора в дані, які можна відобразити. Нарешті, вони відображаються у створеній вкладці (рисунок 4.1):

Рисунок 4.1 – Вхідні данні

Після візуалізації даних переходимо до більш цікавої частини реалізації та створення моделі. Це робиться у функції createModel (приклад 4.5):

```
function createModelFunction() {
  const cnn = tf.sequential();

  cnn.add(tf.layers.conv2d({
      inputShape: [28, 28, 1],
      kernelSize: 5,
      filters: 8,
      strides: 1,
      activation: 'relu',
      kernelInitializer: 'varianceScaling'
    }));
  cnn.add(tf.layers.maxPooling2d({poolSize: [2, 2], strides: [2, 2]}));

  cnn.add(tf.layers.conv2d({
      kernelSize: 5,
      filters: 16,
      strides: 1,
      activation: 'relu',
      kernelInitializer: 'varianceScaling'
    }));
  cnn.add(tf.layers.maxPooling2d({poolSize: [2, 2], strides: [2, 2]}));

  cnn.add(tf.layers.flatten());

  cnn.add(tf.layers.dense({
    units: 10,
    kernelInitializer: 'varianceScaling',
    activation: 'softmax'
  }));

  cnn.compile({
    optimizer: tf.train.adam(),
    loss: 'categoricalCrossentropy',
    metrics: ['accuracy'],
  });

  return cnn;
}
```

Приклад 4.5 – Створення та реалізація моделі (файл script.js)

В основному, створюється два згорткові шари, за якими слідують шари з максимальним опитуванням. Нарешті, згладжується дані в масив і передаються їх через повністю зв'язаний шар, який у цьому випадку є лише одним щільним шаром з 10 нейронами. Цей останній шар фактично є вихідним шаром, який передбачає клас зображення. Потім модель компілюється за допомогою категоричної перехресної ентропії та оптимізатора Адама. Після того, як надрукується резюме моделі за допомогою tfjs-vis, ось що ми отримаємо (рисунок 4.2):

Model Architecture

Layer Name	Output Shape	# Of Params	Trainable
conv2d_Conv2D1	[batch,24,24,8]	208	true
max_pooling2d_MaxPooling2D1	[batch,12,12,8]	0	true
conv2d_Conv2D2	[batch,8,8,16]	3,216	true
max_pooling2d_MaxPooling2D2	[batch,4,4,16]	0	true
flatten_Flatten1	[batch,256]	0	true
dense_Dense1	[batch,10]	2,570	true

Рисунок 4.2 – Архітектура моделі

Після цих дій, було підготувано вхідні дані та модель, щоб була змога їх навчити. Це робиться всередині функції trainModel (приклад 4.6):

```
async function trainModelFunction(model, data, epochs) {
    const metrics = ['loss', 'val_loss', 'acc', 'val_acc'];
    const container = {
      name: 'Model Training', styles: { height: '1000px' }
    };
    const fitCallbacks = tfvis.show.fitCallbacks(container, metrics);

    const batchSize = 512;

    const [trainX, trainY] = getBatch(data, 5500);
    const [testX, testY] = getBatch(data, 1000, true);

    return model.fit(trainX, trainY, {
      batchSize: batchSize,
      validationData: [testX, testY],
      epochs: epochs,
      shuffle: true,
      callbacks: fitCallbacks
```

```
    });
}
```

Приклад 4.6 – Навчання моделі (файл script.js)

По суті, було отримано пакет даних для навчання та пакет даних про тести. Потім визивається метод придатності на моделі і передаються дані для навчання та дані тестів для оцінки. Такі показники, як втрата та точність, відображаються після кожної епохи за допомогою tf-vis (рисунок 4.3).

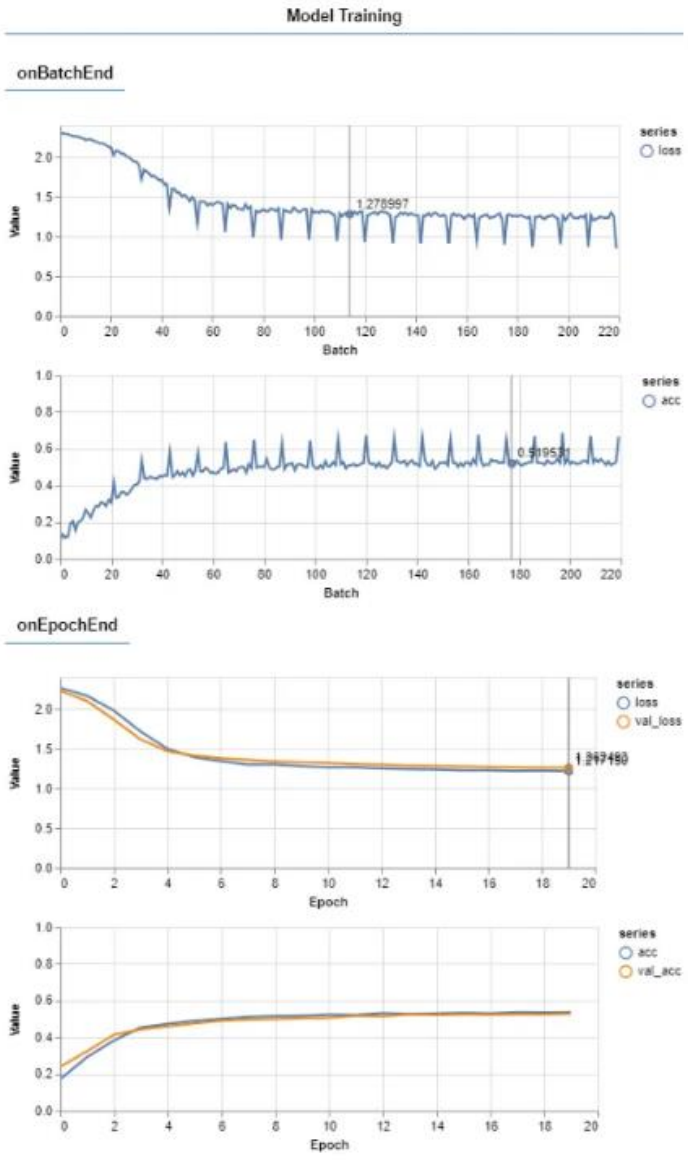

Рисунок 4.3 – Втрата та точність після кожної епохи

Заключний крок у цьому процесі - це оцінка моделі (приклад 4.7). Для

цього відіброжається точність на цифру і використовується концепція матриці плутанини. Ця матриця є лише таблицею, яка часто використовується для опису ефективності класифікаційної моделі. Це все робиться у функції evaluModel.

```
      function predict(model, data, testDataSize = 500) {
        const testData = data.nextDataBatch(testDataSize, true);
        const testxs = testData.xs.reshape([testDataSize, 28, 28, 1]);
        const labels = testData.labels.argMax([-1]);
        const preds = model.predict(testxs).argMax([-1]);

        testxs.dispose();
        return [preds, labels];
      }

      async function displayAccuracyPerClass(model, data) {
        const [preds, labels] = predict(model, data);
        const  classAccuracy  =  await  tfvis.metrics.perClassAccuracy(labels,
preds);
        const container = {name: 'Accuracy', tab: 'Evaluation'};
        tfvis.show.perClassAccuracy(container, classAccuracy, classNames);

        labels.dispose();
      }

      async function displayConfusionMatrix(model, data) {
        const [preds, labels] = predict(model, data);
        const  confusionMatrix  =  await  tfvis.metrics.confusionMatrix(labels,
preds);
        const container = {name: 'Confusion Matrix', tab: 'Evaluation'};
        tfvis.render.confusionMatrix(
            container, {values: confusionMatrix}, classNames);

        labels.dispose();
      }

      async function evaluateModelFunction(model, data)
      {
        await displayAccuracyPerClass(model, data);
        await displayConfusionMatrix(model, data);
      }
```

Приклад 4.6 – Оцінка моделі (файл script.js)

У результаті отримуються досить добрі результати (рисунок 4.4) за допомогою цієї простої моделі і всього за 20 епох (рисунок 4.5). Є можливість покращити ці результати, додавши додаткові згорткові шари або збільшивши кількість епох. Це, звичайно, вплине на тривалість навчального процесу.

Accuracy

Class	Accuracy	# Samples
Zero	0.8158	38
One	0.9831	59
Two	0.8421	57
Three	0.7895	38
Four	0.8246	57
Five	0.8043	46
Six	0.7826	46
Seven	0.8667	60
Eight	0.7708	48
Nine	0.7255	51

Рисунок 4.4 – Результати програми

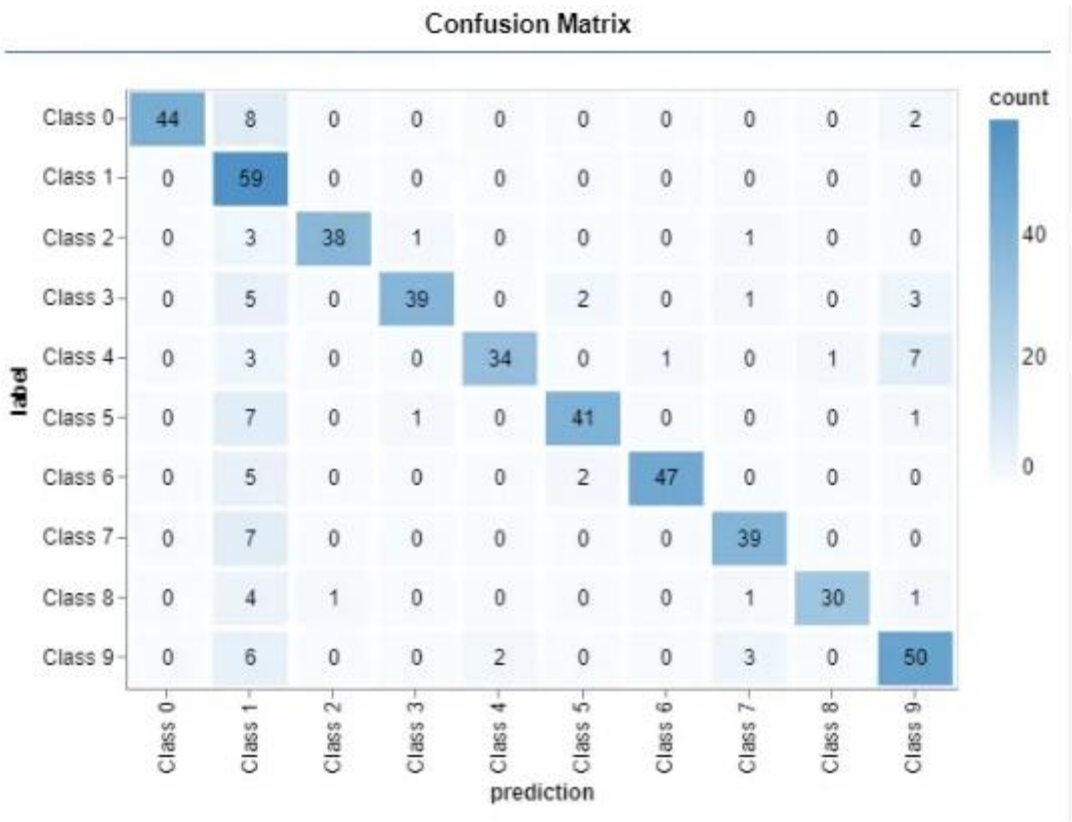

Рисунок 4.5 – Матриця помилок

ВИСНОВКИ

Створення штучного інтелекту, повного та неповного, представляє багато проблем. І на шляху до його творіння, і далі. На шляху створення ШІ виникають такі проблеми, як обмежені ресурси, так і недостатньо знань у цій галузі, проблему загальної життєздатності та багато інших технічних проблем. Після появи людиноподібного ШІ виникне низка проблем. Спочатку втрата інтересу людини до творчості у випадку її заміни, а потім повне приниження людини. Але з іншого боку, творчість повинна приносити людині радість, і вона не повинна здаватися. Можлива і друга проблема: при повній адаптації до ресурсів суспільство втратить свою структуру, а людина стане знеособленою та нерозвиненою протягом життя. По-друге, існує можливість помилок штучного інтелекту роботі в тих областях, де помилки можуть бути фатальними для всього людства. Це, наприклад, національний захист чи енергетика. У будь-якому випадку вирішальне слово має бути в руках особи, яка приймає рішення, наприклад, після початку війни або скасування відключення електроенергії. Врешті-решт, кожна людина може вийти з-під контролю, тому штучний інтелект також може вийти на її образ.

Я можу зробити основні висновки:

а) штучний інтелект - це наукова область, пов’язана з машинним моделюванням інтелектуальних функцій людини;

б) поняття штучного інтелекту часто використовується для позначення здатності комп’ютерної системи виконувати завдання, властиві людському інтелекту, такі як логічні задачі та навчання;

в) будь-яка проблема, якщо алгоритм не відомий заздалегідь або дані неповні, може бути віднесена до завдань зі штучним інтелектом;

г) системи, програми, що виконують дії для вирішення проблеми, можна віднести до штучного інтелекту, якщо результат їх діяльності

подібний до результату людини при вирішенні тієї самої проблеми. Тому штучний інтелект може включати різноманітні програмні засоби: системи розпізнавання тексту, автоматизоване проектування тощо. Але не лише через це, а й тому, що вони діють за подібними принципами у людей;

ґ) є два найбільш перспективні напрямки досліджень ШІ. Перший - наблизити системи ШІ до принципів людського мислення. Другий - це створення ШІ, який поєднує вже створені систем ШІ в єдину систему, яка могла вирішувати людські проблеми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бобровський С. Перспективи і тенденції розвитку штучного інтелекту. PC Week : RE №32, 2001. 32-34 с.

2. Ноткин Л.І. Штучний інтелект і проблеми навчання. Москва: КомКніга, 1999. 12-18 с.

3. Вінер Н. Кібернетика. Москва: Наука, 1983. 31-32 с.

4. Ендрю А. Штучний інтелект - Москва: Мир, 1985. 13-16 с.

5. Шалютін С. М. Штучний інтелект, - Москва: Думка, 1985. 24-30 с.

6. Перспективи розвитку обчислювальної техніки. Кн.2. Інтелектуалізація ЭВМ. Москва, 2002. 51-53 с.

7. Венда В.Ф. Системи гібридного інтелекту - Москва: Машинобудування, 1990. 7-10 с.

8. Уоссерман Ф. Нейрокомп'ютерна техніка: Теорія і практика. Пер. З англ. Москва: Мір, 1992. 34-40 с.

9. Осипов Г.С. Штучний інтелект: стан досліджень і погляд в майбутнє. URL: http://www.raai.org/about/persons/osipov/pages/ai/ai.html .

10. Уїнстон П. Штучний інтелект. Москва, 2001. 56-65 с.

11. A.Kamilarisand, F.X.Prenafeta-Boldu Deep Learning Agriculture: A survey, Computers and Electronics in Agriculture. URL: https://arxiv.org/abs/1807.11809 .

12. J. Del Ser, E. Osaba, J. J. Sanchez-Medina, and I. Fister, Bioinspired computational intelligence and transportation systems: a long road ahead, IEEE Transactions on Intelligent Transportation Systems, 2019. 264-268 с.

13. A. Diez-Olivan, J. Del Ser, D. Galar, and B. Sierra, Data fusion and machine learning for industrial prognosis: Trends and perspectives towards industry 4.0, Information Fusion, № 50, 2019. 92-111 с.

14. N. D. Lane and P. Georgiev, Can deep learning revolutionize mobile sensing? In 16th ACM International Workshop on Mobile Computing Systems and

Applications, 2015. 117-122 с.

15. M. A. Alsheikh, D. Niyato, S. Lin, and Z. Han, Mobile big data analytics using deep learning and Apache Spark, IEEE Network, № 30, 2016. 22-29 с.

16. Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, Droid-sec: deep learning in android malware detection, in ACM SIGCOMM Computer Communication Review, № 44, 2014. 371-372 с.

17. I. Goodfellow, J. Pouget-Abadie, and others, Generative adversarial nets, in Adv. in Neural Information Processing Systems, 2014, 2672-2680 с.

18. M. Arjovsky, S. Chintala, Wasserstein generative ad- versarial networks, in International Conference on Machine Learning, 2017, 214-223 с.

19. N.Papernot and P.McDaniel, Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning, URL: https://arxiv.org/abs/1803.04765 .

20. Y. Gal and Z. Ghahramani, Dropout as a bayesian approximation: Representing model uncertainty in deep learning, in International Conference on Machine Learning, № 48, 2016. 1050-1059 с.

21. A. Subramanya, S. Srinivas, and others, Confidence estimation in deep neural networks via density modeling, URL: https://arxiv.org/abs/1707.07013.

22. M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, Deconvolutional networks, in IEEE Conference on Computer Vision and Pattern Recognition, 2010. 2528-2535 с.

23. K. Simonyan, A. Vedaldi, and A. Zisserman, Deep inside convolutional networks: Visualizing image classification models and saliency maps, URL: https://arxiv.org/abs/1312.6034 .

24. S. Bach, A. Binder, G. Montavon, F.Klauschen, K.-R. Mu ̈ller, and W. Samek, On pixel-wise explanations for non-linear classifier determinations by layer-wise relevance propagation, PloS one, №. 10, 2015. 3-4 с.

25. A.BarredoArrieta, ND ́ıaz-Rodr ́ıguez, J.DelSer, A.Bennetot, S.Tabik, A. Barbado, S. Garc ́ıa, S. Gil-Lo ́pez, D. Molina, R. Benjamins, and F. Herrera, Explainable artificial intelligence (XAI) Concepts, taxonomies, opportunities and

challenges towards responsible AI, Information Fusion, № 58, 2020. 82-115 с.

26. A. Hindupur, The GAN zoo: A list of all named GANs, 2017. 96-115 с.

27. H. Zhang, T. Xu, H. Li, and others Text to photo-realistic image synthesis with stacked generative adversarial networks, in IEEE International Conference on Computer Vision, 2017. 5907-5915 с.

28. H. Wu, S. Zheng, J. Zhang, and K. Huang, Gp-gan Towards realistic high-resolution image blending, in ACM International Conference on Multimedia, 2019. 2487-2495 с.

29. Z. He, W. Zuo, M. Kan, S. Shan, and X. Chen, Attgan: Facial attribute editing by only changing what you want, IEEE Transactions on Image Processing, URL: https://arxiv.org/abs/1711.10678 .

30. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein et al., Imagenet large scale visual recognition challenge, International Journal of Computer Vision, № 115, no.3, 2015. 211-252 с.

31. A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in Adv. in Neural Information Processing Systems, 2012. 1097-1105 с.

32. Z. C. Lipton, The mythos of model interpretability, Queue, № 16, no. 3, 2018. 31-57 с.

33. M. T. Ribeiro, S. Singh, and C. Guestrin, Why should i trust you? Explaining the predictions of any classifier, in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016. 1135-1144 с.

34. S.M.Lundberg and S.-I. Lee,A unified approach to interpreting model predictions, in Adv. Neural Information Processing Systems, 2017. 4765-4774 с.

35. S. Liu, B. Kailkhura, D. Loveland, and Y. Han, Generative counterfactual introspection for explainable deep learning, URL: https://arxiv.org/abs/1907.03077 .

36. Z. Liu, P. Luo, X. Wang, and X. Tang, Deep learning face attributes in the wild, in International Conference on Computer Vision, 2015. 3730-3738 с.

37. S. Arora, A. Risteski, and Y. Zhang, Theoretical limitations of encoder-decoder GAN architects, URL: https://arxiv.org/abs/1711.02651 .

38. P. Lyu, X. Bai, C. Yao, Z. Zhu, T. Huang, and W. Liu, Auto-encoder guided GAN for chinese calligraphy synthesis, in IEEE International Conference on Document Analysis and Recognition, URL: https://arxiv.org/abs/1706.08789 .

39. A. Benitez-Hidalgo, AJ Nebro, J. Garcia-Nieto, I. Oregi, and J. Del Ser, jMetalPy: a Python framework for multi-objective optimization with metaheuristics, Swarm and Evolutionary Computation, URL: https://arxiv.org/abs/1903.02915 .

40. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A fast and elitist multiobjective genetic algorithm NSGA-II, IEEE Transactions on Evolutionary Computation, URL: https://www.iitk.ac.in/kangal/Deb_NSGA-II.pdf .

41. S. Kukkonen and J. Lampinen, GDE3 The third evolution step of generalized differential evolution, in IEEE Congress on Evolutionary Computation, URL: https://www.iitk.ac.in/kangal/papers/k2005013.pdf .

42. AJ Nebro, JJ Durillo, J. Garcia-Nieto, CC Coello, F. Luna, and E. Alba, SMPSO A new PSO-based metaheuristic for multi-objective optimization, in the IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making, 2009. 66-73 c.