МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

дискретного аналізу та інтелектуальних систем
(повна назва кафедри)

Магістерська робота

"Згорткові мережі і їх застосування"

Виконав: студент групи ПМіМ-23с
спеціальності
122 «Комп'ютерні науки»
(шифр і назва спеціальності)

(підпис) Тимець Д.І.
(прізвище та ініціали)

Керівник (підпис) Щербина Ю.М.
(прізвище та ініціали)

Рецензент (підпис) Г. Шинкаренко
(прізвище та ініціали)

ДЕКАН
факультету прикладної
математики та інформатики
ЛНУ ім. Івана Франка

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики

Кафедра Дискретного аналізу та інтелектуальних систем

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

«ЗАТВЕРДЖУЮ»
Завідувач кафедри Притула М.М.

"31" серпня **2022 року**

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Тимцю Данилу Ігоровичу
(прізвище, ім'я, по батькові)

1 .Тема роботи "Згорткові мережі і їх застосування"

керівник роботи Щербина Юрій Миколайович, канд. фіз.-мат. наук, доц. ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від **" 13 " вересня 2022 року № 15**

2. Строк подання студентом роботи **12.12.2022р.**

3. Вихідні дані до роботи
Документація по Python, Pandas, NumPy, Keras, TensorFlow, SciPy. Науково-технічні публікації, Goodfellow I. Deep learning, Інтернет-ресурси за темою, цифрові зображення у сховищах даних із відкритим доступом. Середовище розробки - Jupyter Notebook.

4. Зміст магістерської роботи (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі.
2. Опис використаних технологій.
3. Програма реалізація.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
1. Архітектура та структура згорткової нейронної мережі.
2. Блок-схеми методу розпізнавання емоцій.
3. Демонстрація роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 31 серпня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми магістерської роботи	29.06.2022 - 03.07.2022	
2	Аналіз предметної галузі та опрацювання літератури	04.07.2022 - 08.08.2022	
3	Дослідження необхідних технологій	09.08.2022 - 28.08.2022	
4	Написання першого розділу роботи	29.08.2022 - 13.09.2022	
5	Презентація першого розділу та проведених досліджень	13.09.2022	
6	Написання другого розділу роботи	14.09.2022 -05.10.2022	
7	Розробка системи та логіки алгоритму розпізнавання	20.09.2022 - 13.10.2022	
8	Тестування моделі	14.10.2022 - 17.10.2022	
9	Демонстрація роботи розробленої системи	18.10.2022	
10	Доопрацювання системи та виправлення помилок	19.10.2022 - 23.10.2022	
11	Написання третього розділу роботи	24.10.2022 - 11.10.2022	
12	Написання четвертого розділу роботи	11.10.2022 - 14.10.2022	
13	Написання вступу, висновків та оформлення магістерської роботи	15.10.2022 - 20.10.2022	
14	Подання магістерської роботи керівнику на рецензування	21.10.2022	

Студент _____ Тимець Д.І.
(підпис) (прізвище та ініціали)

Керівник _____ Щербина Ю.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У магістерській роботі реалізована глибока згорткова нейронна мережа (DCNN) за допомогою фреймворку мови високого рівня Python TensorFlow на ядрі Keras API. У якості середовища розробки було використано Jupyter Notebook, оскільки він дає можливість візуалізувати проміжні результати виконання програми без додаткових затримок та накладань.

У якості функції активації виступає функція активації ELU. З метою узагальнення я використав відсіювання через рівні проміжки часу та функцію оптимізації Надам. Усі використані фреймворки, бібліотеки, пакети та матеріали, результати виконання та оцінка ефективності глибокої згорткової нейронної мережі наведені у роботі.

SUMMARY

In the master's thesis, a deep neural network (DCNN) is implemented using the high-level framework of the Python TensorFlow language on the Keras API core. Jupyter Notebook was used as the development environment, which allows you to visualize intermediate results of program execution without additional delays and overlays.

The activation function is the ELU activation function. The shared usage fee uses Nadam's periodic filtering and optimization feature. The paper lists all used packages of frameworks, libraries, materials and materials, execution results, and performance evaluation of deep convolutional neural networks.

ВСТУП

Нині застосування штучних нейронних мереж стало популярним у різних сферах людських потреб. Багато організацій інвестують у нейронні мережі для вирішення проблем у різних сферах та економічному секторі, які традиційно підпадають під відповідальність за дослідження операцій.

Що робить штучний інтелект унікальним, так це те, що він здебільшого пропонується для аналізу даних вченими в галузі соціальних наук і мистецтва, окрім його корисності в науці та техніці через його широке застосування. Наприклад, останнім часом штучний інтелект (ШІ) широко використовується для оптимізації в різних сферах, таких як промислове виробництво та розвідка покладів нафти і бізнес налаштування.

Від фінансів і соціальних медіа до закону та порядку, нейронні мережі сьогодні всюди. Нижче наведено кілька найкращих та найпопулярніших застосувань нейронних мереж у 2022 році:

Правопорядок: незважаючи на те, що їх використання обмежене в певних юрисдикціях, системи розпізнавання обличчя набувають популярності як надійна форма спостереження. Ці рішення порівнюють людські обличчя з базою даних цифрових зображень. Окрім сповіщення органів влади про присутність втікачів і забезпечення виконання мандатів на маски, ця пропозиція нейронної мережі також корисна для забезпечення вибіркового проникнення до конфіденційних фізичних місць, таких як офіс.

Фінанси: у минулому фінансові ринки були схильні до ризиків, які було майже неможливо передбачити. Сьогодні це вже не так: нейронні мережі допомогли значною мірою пом’якшити високу варіативність на фондових ринках.

Соціальні медіа: У світі після пандемії соціальні мережі охопили майже кожну нішу людського життя. Користувачі часто дивуються тому, як платформи соціальних мереж можуть «читати їхні думки», хоча насправді вони мають за це дякувати нейронним мережам.

Аерокосмічна: нейронна мережа відіграє вирішальну роль у всій аерокосмічній промисловості, від техніки до польотів. Під час виробничого процесу нейронні мережі розгортаються для бездоганної діагностики несправностей, оскільки навіть найменший дефект у літаку може призвести до втрати сотень життів.

Захист: з огляду на посилення геополітичної нестабільності в Азії та Європі у 2022 році перевірені оборонні рішення стають надзвичайно важливими для кожної країни. Надійна оборонна позиція дозволяє країні отримати прихильне визнання на світовій арені.

Охорона здоров'я: Тести на основі зображень є основою індустрії охорони здоров’я, яка використовує потужність обробки зображень згорткових нейронних мереж для виявлення захворювань.

Розбір підпису та почерку: рішення для перевірки підписів на основі ШІ поступово стають нормою у фінансовій, адміністративній та пов’язаних сферах. Фінансові установи та бюрократія покладаються на перевірку підпису, щоб підтвердити особу кінцевих користувачів і запобігти шахрайським транзакціям.

Метеорологія є важливою частиною повсякденного життя, вона допомагає людям заздалегідь підготуватися до майбутніх погодних умов і навіть передбачити можливість стихійних лих. Із впровадженням нейронних мереж у сферу метеорології прогнози погоди стають точнішими.

Нейронні мережі — це руйнівне застосування штучного інтелекту, яке дозволяє використовувати можливості глибинного навчання для вирішення проблем, щоб покращити якість нашого життя. Технології нейронної мережі все частіше використовуються для вирішення абстрактних проблем, таких як дизайн ліків, обробка природної мови та перевірка підпису. Оскільки нейронні мережі продовжуватимуть ставати швидшими та точнішими, технологічний прогрес людства значно прискориться.

У магістерській роботі я розробив згорткову нейронна мережу, яка здатна розпізнавати емоції людей. Даний проект може застосовуватися в області

правопорядку для розпізнавання емоцій, як певний детектор брехні, який базується на теорії міміки.

Виявилося, що інструмент машинного навчання, навчений виявляти ознаки брехні, справляється краще, ніж звичайна людина, використовуючи трохи більше, ніж дані від датчиків, які можна носити, які вловлюють крихітне мерехтіння м'язів обличчя.

Зазвичай використовувані технології детектора брехні, такі як поліграф, зазвичай покладаються на такі фізіологічні реакції, як частота серцевих скорочень, артеріальний тиск і частота дихання – усі функції, які люди можуть навчитися контролювати під тиском. Незважаючи на те, що поліграфи постійно використовуються в різних сферах правоохоронних органів, поліграфи в кращому випадку вважаються неточними.

Також автоматичний аналіз виразу обличчя (AFEA) можна використовувати в інших сферах, включаючи реляційну та семантичну комунікацію, клінічну психологію, психіатрію, неврологію, оцінку болю, виявлення брехні, інтелектуальні налаштування та мультимодальний інтерфейс людина–комп'ютер (HCI).

Звісно, даний проект ще потребує великої кількості удосконалень, але складніші системи можуть бути реалізовані на основі даної роботи.

РОЗДІЛ 1. ЗАГАЛЬНІ ПОНЯТТЯ ПРО НЕЙРОННІ МЕРЕЖІ. ЗГОРТКОВІ МЕРЕЖІ ТА ЇХ ЗАСТОСУВАННЯ.

Нейронна мережа — це програмне рішення, яке використовує алгоритми машинного навчання (ML) для «імітації» операцій людського мозку. Нейронні мережі ефективніше обробляють дані та мають покращені можливості розпізнавання образів і вирішення проблем порівняно з традиційними комп’ютерами. Нейронні мережі також відомі як штучні нейронні мережі (ANN) або імітовані нейронні мережі (SNN).

Нейронні мережі є підтипом машинного навчання та важливим елементом алгоритмів глибокого навчання. Як і її функціональність, архітектура нейронної мережі також базується на людському мозку. Його міцно взаємопов’язана структура дозволяє імітувати процеси передачі сигналів біологічних нейронів.

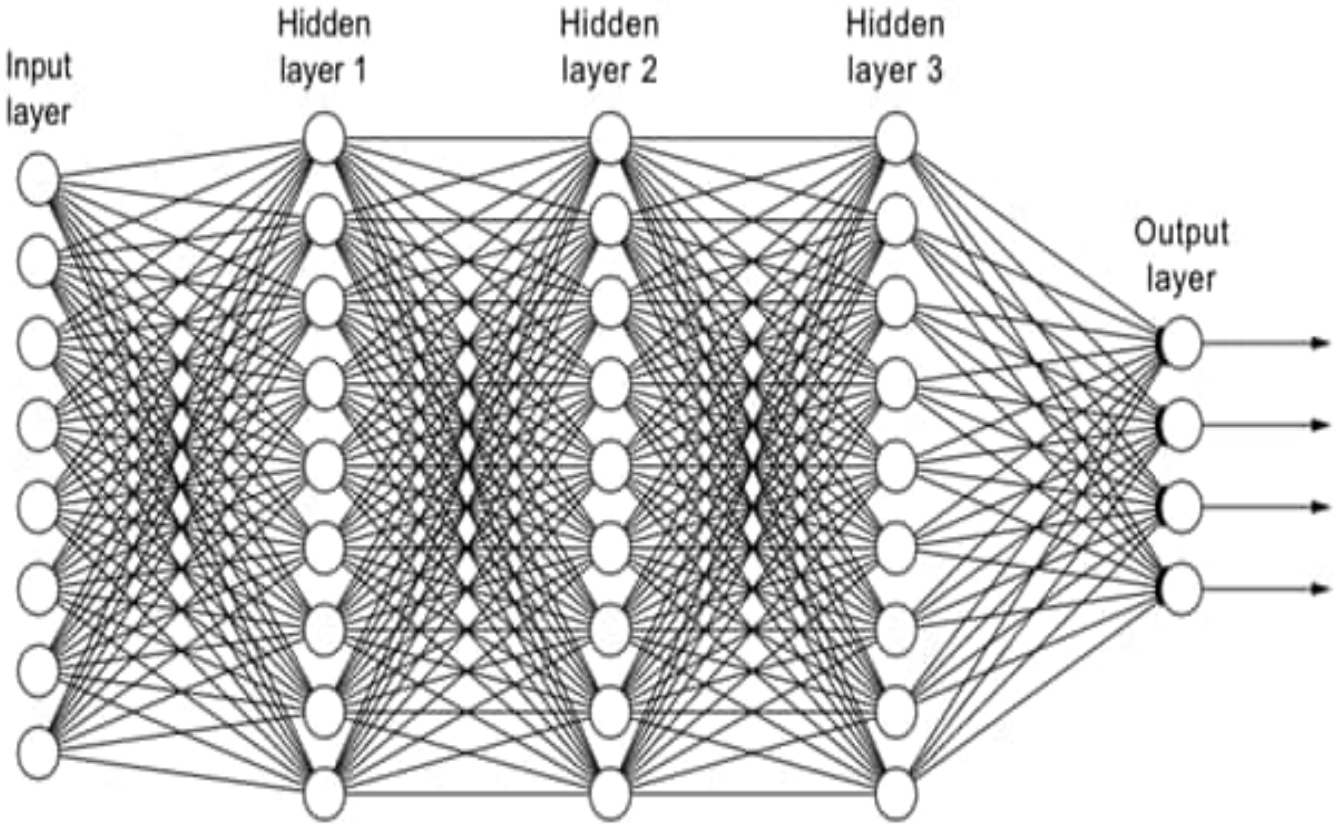

Рис 1.1. Схематичний вигляд нейронної мережі

1. Історія розвитку нейронних мереж

У 1943 році нейрофізіолог Уоррен МакКаллох і математик Уолтер Піттс написали статтю про те, як можуть працювати нейрони. Щоб описати, як можуть працювати нейрони в мозку, вони змоделювали просту нейронну мережу за допомогою електричних схем [1].

У 1949 році Дональд Гебб написав «Організацію поведінки », працю, яка вказувала на той факт, що нейронні шляхи зміцнюються кожного разу, коли вони використовуються, концепція, фундаментально важлива для способів, якими люди навчаються. Якщо два нерви спрацьовують одночасно, стверджував він, зв'язок між ними посилюється.

Коли комп'ютери стали більш досконалими в 1950-х роках, нарешті стало можливим змоделювати гіпотетичну нейронну мережу. Перший крок до цього зробив Натаніал Рочестер з дослідницьких лабораторій IBM. На жаль для нього, перша спроба зробити це виявилася невдалою.

У 1959 році Бернард Уідроу та Марсіан Гофф зі Стенфорда розробили моделі під назвою «ADALINE» і «MADALINE». У типовому прояві любові Стенфордського університету до абревіатур, назви походять від використання ними кількох ADAptive LINEar Elements [1].

ADALINE було розроблено для розпізнавання двійкових шаблонів, щоб, якщо він зчитував потокові біти з телефонної лінії, міг передбачити наступний біт.

MADALINE була першою нейронною мережею, застосованою до реальної проблеми з використанням адаптивного фільтра, який усуває відлуння на телефонних лініях. Хоча ця система така ж давня, як і системи управління повітряним рухом, вона все ще використовується в комерційних цілях.

У 1962 році Widrow & Hoff розробили процедуру навчання, яка перевіряє значення до того, як вага регулює його (тобто 0 або 1) відповідно до правила: Зміна ваги = (значення рядка попереднього зважування) * (Помилка / (Кількість вхідних даних)). Він заснований на ідеї, що хоча один активний персептрон може мати велику помилку, можна налаштувати значення ваги, щоб розподілити

її по мережі або принаймні на сусідні персептрони. Застосування цього правила все одно призводить до помилки, якщо рядок перед вагомістю дорівнює 0, хоча з часом це виправиться. Якщо помилка зберігається так, що вся вона розподіляється на всі ваги, тоді помилка усувається.

Незважаючи на пізніший успіх нейронної мережі, традиційна архітектура фон Неймана зайняла обчислювальну сцену, а нейронні дослідження залишилися позаду. За іронією долі, сам Джон фон Нейман запропонував імітувати нейронні функції за допомогою телеграфних реле або вакуумних трубок.

У той же період часу була написана стаття, яка припускала, що не може бути розширення від одношарової нейронної мережі до багатошарової нейронної мережі. Крім того, багато людей у цій галузі використовували функцію навчання, яка була фундаментально помилковою, оскільки її не можна було диференціювати по всій лінії. У результаті, дослідження та фінансування різко скоротилися.

Це поєднувалося з тим фактом, що перші успіхи деяких нейронних мереж призвели до перебільшення потенціалу нейронних мереж, особливо з огляду на практичні технології того часу. Обіцянки залишилися невиконаними, а іноді серйозніші філософські запитання викликали страх. Письменники міркували про вплив, який так звані «мислячі машини» матимуть на людей, ідеї, які існують і сьогодні.

Ідея комп'ютера, який сам себе програмує, дуже приваблива. Якби Windows 2000 від Microsoft могла перепрограмувати себе, вона могла б виправити тисячі помилок, допущених програмістами.

Такі ідеї були привабливі, але дуже складні для реалізації. Крім того, архітектура фон Неймана набирала популярності. У цій галузі було досягнуто певних успіхів, але здебільшого досліджень було небагато.

У 1972 році Кохонен і Андерсон розробили подібну мережу незалежно один від одного, яка буде детальніше описана в подальшому викладі. Вони обидва використовували матричну математику для опису своїх ідей, але не

усвідомлювали, що вони створюють низку аналогових схем ADALINE. Передбачається, що нейрони активують набір виходів, а не один.

Перша багаторівнева мережа була розроблена в 1975 році, неконтрольована мережа [2].

У 1982 році інтерес до цієї галузі відновився. Джон Хопфілд з Каліфорнійського технологічного інституту представив доповідь Національній академії наук. Його підхід полягав у створенні більш корисних машин за допомогою двох-направлених ліній. Раніше зв'язки між нейронами були тільки одним шляхом.

Того ж року Рейлі та Купер використовували «гібридну мережу» з кількома рівнями, кожен з яких використовував різну стратегію вирішення проблем.

Також у 1982 році відбулася спільна американсько-японська конференція з кооперативних/конкурентних нейронних мереж. Японія оголосила про нову спробу п’ятого покоління нейронних мереж, і американські газети викликали занепокоєння, що США можуть залишитися позаду в цій галузі. (Обчислення п’ятого покоління включають штучний інтелект. Перше покоління використовувало перемикачі та дроти, друге покоління використовувало транзистор, третє покоління використовувало твердо тільну технологію, як-от інтегральні схеми та мови програмування вищого рівня, а четверте покоління – це генератори коду.) У результаті було більше фінансування і, отже, більше досліджень у цій галузі.

У 1986 році, коли багатошарові нейронні мережі були в новинах, проблема полягала в тому, як поширити правило Уідроу-Гоффа на кілька рівнів. Три незалежні групи дослідників, до однієї з яких входив Девід Румелгарт, колишній співробітник факультету психології Стенфордського університету, висунули подібні ідеї, які тепер називаються мережами зворотного поширення, оскільки вони розподіляють помилки розпізнавання образів по всій мережі. Гібридні мережі використовують лише два шари, а мережі зворотного поширення використовують багато. Результатом цього є те,

що мережі зворотного розповсюдження «повільно навчаються», і для навчання їм потрібні, можливо, тисячі ітерацій.

На даний час, нейронні мережі використовуються в кількох програмах, деякі з яких я опишу у цій роботі. Фундаментальна ідея природи нейронних мереж полягає в тому, що якщо вони працюють у природі, вони повинні мати можливість працювати в комп'ютерах. Однак майбутнє нейронних мереж — за розробкою апаратного забезпечення. Подібно до вдосконалених машин для гри в шахи, таких як Deep Blue, швидкі та ефективні нейронні мережі залежать від апаратного забезпечення, визначеного для їх можливого використання.

Дослідження, зосереджені на розробці нейронних мереж, є відносно повільними. Через обмеження процесорів нейронні мережі навчаються тижнями. Деякі компанії намагаються створити так званий «кремнієвий компілятор» для створення певного типу інтегральної схеми, оптимізованої для застосування нейронних мереж. Цифрові, аналогові та оптичні чіпи — це різні типи чіпів, які розробляються. Можна було б відразу відмовитися від аналогових сигналів як від минулого. Однак нейрони в мозку насправді працюють більше як аналогові сигнали, ніж цифрові сигнали. У той час як цифрові сигнали мають два різних стани (1 або 0, ввімкнено або вимкнено), аналогові сигнали змінюються між мінімальними та максимальними значеннями. Проте може пройти деякий час, перш ніж оптичні чіпи можна буде використовувати в комерційних цілях.

2. Архітектура нейронної мережі

Архітектура нейронної мережі включає шари вузлів, які розподілені між вхідним шаром, один або кілька прихованих шарів і вихідний рівень. Вузли — це «штучні нейрони», пов'язані один з одним і пов'язані з певною вагою та порогом. Як тільки вихідні дані одного вузла перевищують заданий поріг, цей конкретний вузол активується, а його дані передаються на наступний рівень у мережі. Якщо порогове значення вузла не перевищено, дані не передаються на наступний рівень мережі [3].

На відміну від традиційних комп’ютерів, які обробляють дані послідовно, нейронні мережі можуть навчатися та виконувати багато завдань. Іншими словами, у той час як звичайні комп’ютери лише дотримуються вказівок свого програмування, нейронні мережі постійно розвиваються за допомогою вдосконалених алгоритмів. Можна сказати, що нейронні комп’ютери «програмують себе» для пошуку рішень раніше небачених проблем.

Крім того, традиційні комп’ютери працюють за допомогою логічних функцій на основі певного набору обчислень і правил. Навпаки, нейронні комп’ютери можуть обробляти логічні функції та необроблені вхідні дані, такі як зображення, відео та голос.

У той час як традиційні комп’ютери готові вийти з коробки, нейронні мережі необхідно «навчати» з часом, щоб підвищити їх точність і ефективність. Точне налаштування цих навчальних машин для забезпечення точності приносить значні дивіденди, надаючи користувачам потужний обчислювальний інструмент у галузі штучного інтелекту (ШІ) та інформатики.

Нейронні мережі здатні класифікувати та кластеризувати дані на високій швидкості. Це, серед іншого, означає, що вони можуть завершити розпізнавання мови та зображень за кілька хвилин замість годин, які б виконували люди-експерти. Найпоширенішою нейронною мережею сьогодні є пошукові алгоритми Google.

3. Принцип роботи нейронної мережі

Здатність нейронної мережі «мислити» зробила революцію в обчисленнях, як ми їх знаємо. Ці розумні рішення здатні інтерпретувати дані та враховувати контекст.

Чотири важливі кроки, які виконують нейронні мережі для ефективної роботи [4]:

- Асоціювання або навчання дозволяє нейронним мережам "запам'ятовувати" шаблони. Якщо комп’ютеру буде показано незнайомий шаблон, він об’єднає шаблон із найбільш близьким збігом, наявним у його пам’яті.

- Класифікація або організація даних або шаблонів у попередньо визначені класи.
- Кластеризація або ідентифікація унікального аспекту кожного екземпляра даних для його класифікації навіть без будь-якого іншого контексту.
- Прогнозування або отримання очікуваних результатів за допомогою відповідних вхідних даних, навіть якщо весь контекст не надано заздалегідь.

Нейронні мережі потребують високої пропускної здатності для точного виконання цих функцій майже в реальному часі. Це досягається шляхом розгортання багатьох процесорів, які працюють паралельно один одному, які розташовані на рівнях.

Процес нейронної мережі починається з того, що перший рівень отримує необроблені вхідні дані. Ви можете порівняти це із зоровими нервами людини, які отримують візуальні дані. Після цього кожен наступний рівень отримує результати попереднього. Це триває до тих пір, поки останній рівень не обробить інформацію та виведе результат.

Кожен окремий вузол обробки містить свою базу даних, включно з усіма попередніми знаннями та правилами, за якими він був або запрограмований спочатку, або розроблений з часом. Усі ці вузли та рівні сильно взаємопов’язані.

Процес навчання (також відомий як навчання) починається, коли нейронну мережу структуровано для конкретного застосування. Навчання може здійснюватися як під наглядом, так і без нагляду. У першому випадку мережа отримує правильні результати або через доставку бажаної комбінації входу та виходу, або через ручну оцінку продуктивності мережі. З іншого боку, неконтрольоване навчання відбувається, коли мережа інтерпретує вхідні дані та генерує результати без зовнішніх інструкцій чи підтримки.

Адаптивність є однією з основних якостей нейронної мережі. Ця характеристика дозволяє модифікувати алгоритми машинного навчання в міру того, як вони навчаються на основі свого навчання та подальших операцій. Моделі навчання в основному зосереджені навколо ваги вхідних

потоків, при цьому кожен вузол призначає вагу вхідним даним, які він отримує від своїх попередніх вузлів. Вхідні дані, які є важливими для отримання правильних відповідей, отримують більшу вагу в наступних процесах.

Окрім адаптивності, нейронні мережі використовують численні принципи для визначення правил роботи та прийняття рішень. Нечітка логіка, навчання на основі градієнта, Байєсівські методи та генетичні алгоритми відіграють важливу роль у процесі прийняття рішень на рівні вузла. Це допомагає окремим вузлам вирішити, що слід надіслати на наступний рівень на основі вхідних даних, отриманих від попереднього рівня.

Основні правила зав'язків об'єктів також можуть допомогти забезпечити якісніше моделювання даних. Наприклад, нейронній мережі розпізнавання обличчя можна дати вказівку, що «зуби завжди знаходяться під носом» або «вуха знаходяться з обох боків обличчя». Додавання таких правил вручну може допомогти скоротити час навчання та допомогти у створенні більш ефективної моделі нейронної мережі.

Однак додавання правил не завжди добре. Це також може призвести до неправильних припущень, коли алгоритм намагається вирішити проблеми, не пов'язані з правилами. Попереднє завантаження неправильного набору правил може призвести до створення нейронних мереж, які надають нерелевантні, неправильні, некорисні або контр-продуктивні результати. Тому дуже важливо ретельно вибирати правила, які додаються до системи.

Хоча нейронним мережам і особливо неконтрольованому навчанню ще потрібно пройти довгий шлях до досягнення досконалості, ми можемо бути ближчими до досягнення визначального прориву, ніж ми думаємо. Фактом є те, що зв'язки всередині нейронної мережі не можуть бути настільки численними чи ефективними, як зв'язки в людському мозку. Проте закон Мура [6], який стверджує, що середня обчислювальна потужність комп'ютерів подвоюється кожні два роки, все ще процвітає. Ця тенденція визначає наші очікування від штучного інтелекту та нейронних мереж.

4. Типи нейронних мереж

Нейронні мережі класифікуються на основі кількох факторів, включаючи їхню глибину, кількість прихованих шарів і можливості введення/виведення кожного вузла.

Рис 1.2. Типи нейронних мереж

Згорткові нейронні мережі

Будучи дуже популярною моделлю нейронних мереж, згорткові нейронні мережі використовують тип багатошарового персептрону та включають один або більше згорткових шарів. Ці шари можуть бути як об’єднаними, так і повністю з’єднаними [7].

Ця модель нейронної мережі використовує принципи лінійної алгебри, особливо множення матриць, для виявлення та обробки шаблонів у зображеннях. Згорткові шари в цій моделі можуть створювати карти функцій, які захоплюють певну область у візуальному введенні. Потім, ця область далі розбивається та аналізується для отримання цінних результатів.

Згорткові нейронні мережі корисні для програм розпізнавання зображень на базі ШІ. Цей тип нейронної мережі зазвичай використовується в розширених випадках використання, таких як розпізнавання обличчя, обробка природної мови (NLP), оптичне розпізнавання символів (OCR) і класифікація зображень. Він також розгортається для ідентифікації перефразування та обробки сигналу.

Деконволюційні нейронні мережі

Деконволюційні нейронні мережі працюють за тими ж принципами, що й згорткові мережі, за винятком навпаки. Це спеціальне застосування штучного інтелекту спрямоване на виявлення втрачених сигналів або функцій, які раніше могли бути відкинуті як неважливі, оскільки згорткова нейронна мережа виконувала поставлене завдання. Нейронні мережі деконволюції корисні для різних програм, включаючи аналіз і синтез зображень [8].

Рекурентні нейронні мережі

Ця складна модель нейронної мережі працює, зберігаючи вихідні дані, згенеровані вузлами процесора, і повертаючи їх назад в алгоритм. Цей процес дозволяє рекурентним нейронним мережам покращити свої можливості прогнозування.

У цій моделі нейронної мережі кожен вузол поводиться як осередок пам’яті. Ці осередки працюють, щоб забезпечити інтелектуальне обчислення та реалізацію шляхом обробки отриманих даних. Однак те, що відрізняє цю модель, так це її здатність згадувати та повторно використовувати всі оброблені дані.

Сильна петля зворотного зв’язку є однією з найважливіших особливостей рекурентної нейронної мережі. Ці рішення для нейронних мереж можуть «самонавчатися» на своїх помилках. Якщо зроблено неправильний прогноз, система навчається на основі зворотного зв’язку та намагається зробити правильний прогноз, пропускаючи дані через алгоритм вдруге.

Повторювані нейронні мережі зазвичай використовуються в програмах синтезу мовлення, а також для прогнозування продажів і фондового ринку.

Прямі нейронні мережі

Цей простий варіант нейронної мережі передає дані в одному напрямку через різні вузли обробки, поки дані не досягнуть вихідного вузла. Нейронні мережі прямого зв’язку призначені для обробки великих обсягів «зашумлених»

даних і створення «чистих» виходів. Цей тип нейронної мережі також відомий як модель багатошарових персептронів (MLP).

Архітектура нейронної мережі прямого зв’язку включає вхідний рівень, один або більше прихованих рівнів і вихідний рівень. Незважаючи на свою альтернативну назву, ці моделі використовують сигмоподібні нейрони, а не персептрони, що дозволяє їм вирішувати нелінійні проблеми реального світу.

Нейронні мережі прямого зв’язку є основою для розпізнавання обличчя, обробки природної мови, комп’ютерного зору та інших моделей нейронних мереж.

Модульні нейронні мережі

Модульні нейронні мережі містять серію незалежних нейронних мереж, робота яких контролюється посередником. Кожна незалежна мережа є «модулем», який використовує окремі вхідні дані для виконання певної частини загальної мети великої мережі.

Модулі не спілкуються один з одним і не втручаються в процеси один одного під час обчислення. Це робить виконання обширних і складних обчислювальних процесів більш ефективним і швидким.

Генеративні змагальні мережі

Генеративні змагальні мережі – це рішення для генеративного моделювання, яке використовує згорткові нейронні мережі та інші пропозиції глибокого навчання для автоматизації виявлення шаблонів у даних. Генеративне моделювання використовує неконтрольоване навчання для отримання правдоподібних висновків із вихідного набору даних.

Генеративні суперницькі мережі тренують генеративні моделі, створюючи «проблему навчання під керівництвом», що містить модель генератора та модель дискримінатора. Перший готовий зробити нові висновки з вхідних даних. У той же час останній прагне позначити згенеровані висновки як «справжні» (зсередини набору даних) або «фальшиві» (генеровані алгоритмом). Після того, як модель дискримінатора приблизно в половині випадків

неправильно позначає згенеровані висновки, модель генератора створює правдоподібні висновки.

РОЗДІЛ 2. РОЗПІЗНАВАННЯ ЕМОЦІЙ ТА ОБЛИЧЧЯ ЗА ДОПОМОГОЮ ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ

За останні кілька років сфера машинного навчання зазнала серйозного розвитку. Одним із важливих досягнень є техніка, відома як «глибоке навчання», яка спрямована на моделювання високого рівня абстракції даних, шляхом використання глибоких мережевих архітектур, що складаються з кількох лінійних/нелінійних перетворення.

Системи глибокого навчання — це інтелектуальні системи, які імітують роботу людського мозку у представленні складних даних зі сценаріїв реального світу та допомагають в прийнятті розумних рішень.

Глибоке навчання, також відоме як глибоке структуроване навчання або ієрархічне навчання, належить до сімейства методів машинного навчання, які базуються на розумінні представлення даних [10]. Це зробило значний вплив на продуктивність комп’ютерного зору, який раніше був недосяжний для багатьох завдань, таких як класифікація зображень і виявлення об'єктів.

Глибоке навчання використовується в дослідженнях щодо графіки моделювання, розпізнавання образів, обробка сигналів, комп’ютерного зору, розпізнавання мови, розпізнавання звуку та розпізнавання обличчя (FR).

Технологія розпізнавання обличчя (FR) визначена як область активних досліджень в останні роки, оскільки підвищення вимог до безпеки та потенціалу технології в правоохоронній та комерційній сферах набули неабиякого значення. FR містить два режими роботи. По-перше, режим перевірки відомий як відповідність один до одного в біометрії. Операційний режим перевірки використовується для виділення обличчя з багатьох облич в базі даних, щоб дізнатися, чи належать деталі обличчя конкретній людині.

По-друге, ідентифікація, режим відомий як відповідність один до багатьох. Ідентифікація передбачає взяття особи та порівняння її біометричних даних до бази даних можливих ідентичностей. Технологія FR складається з чотирьох етапів, які включають виявлення обличчя, вирівнювання, представлення (виділення рис обличчя) та класифікація.

У розпізнаванні обличчя основною проблемою є схема представлення ознак, яка використовується для вилучення ознак за допомогою кращого методу представлення для даної біометричної ознаки [11]. Вилучення ознак є одним із найбільш важливих кроків для класифікації зображень. Вилучення особливостей означає збереження найважливішої інформації, необхідної для класифікації. Існує багато процедур вилучення функцій, які були запропоновані для використання в біометричній системі, включаючи аналіз головних компонентів (PCA), незалежний компонентний аналіз (ICA), локальні бінарні шаблони (LBP) та гістограмний метод. Останнім часом типовим підходом до вилучення ознак, який використовується для розпізнаванні обличчя, є глибоке навчання, особливо серед них виділяється згорткова нейронна мережа (CNN), яка має значні переваги.

Існують різні підходи до використання CNN. По-перше, це вивчення моделі з нуля. У цьому випадку використовується та навчається архітектура попередньо навченої моделі відповідно до набору даних.

По-друге, у випадках, коли набір даних є великим, використовується передача навчання з функціями попередньо навченого CNN. Нарешті, CNN можна використовувати за допомогою навчання перенесення, зберігаючи згорткову базу в її початковій формі, а потім використовуючи її результати для подачі класифікатора. Попередньо навчена модель використовується як механізм вилучення фіксованих ознак у випадках, коли набір даних невеликий або коли проблема схожа на ту, що підлягає класифікації.

1. Згорткові нейронні мережі

Знайти гарне внутрішнє представлення зображень об’єктів і функцій було основною метою з початку комп’ютерного зору. Тому було винайдено багато інструментів для роботи з зображеннями. Багато з них засновані на математичній операції, яка називається згорткою . Навіть коли нейронні мережі використовуються для обробки зображень, згортка залишається основною операцією.

Згорткові нейронні мережі використовують переваги нейронних мереж загалом і йдуть ще далі, щоб працювати з двовимірними даними. Таким чином, навчальні параметри є елементами двовимірних фільтрів. У результаті застосування фільтра до зображення створюється карта функцій, яка містить інформацію про те, наскільки добре фільтр відповідає відповідній позиції на зображенні.

Крім того, згортка з'єднує перцептрони локально. Оскільки об'єкти завжди належать до свого просторового положення на зображенні, немає потреби повністю пов'язувати кожен етап один з одним. Конвертація зберігає інформацію про навколишні перцептрони та обробляє їх відповідно до приналежних ваг. На кожному етапі дані додатково обробляються шляхом нелінійності та виправлення. Зрештою, відбувається об'єднання підвибірок кожного шару.

Загалом, етапи глибокого навчання, призводять до ієрархічної структурованості внутрішнього представлення. Виявилося, особливо для зображень, що таке уявлення надзвичайно потужне, адже, у такому представленні, ступені низького рівня використовуються для виявлення первинних країв зображення. Щодо етапів високого рівня, то вони пов'язують інформацію про те, де та як розташовані об'єкти щодо первинного зображення.

Згорткова нейронна мережа (CNN) — це різновид нейронної мережі із згортковими шарами. Загалом, CNN містить два типи прихованих шарів, тобто згорткові шари та шари об'єднання, які зазвичай розташовані по черзі в нейронній мережі [14].

Подібно до біологічної нейронної мережі, вагові коефіцієнти з'єднань CNN можуть бути загальними для всієї нейронної мережі, що може не тільки зменшити кількість вагових коефіцієнтів з'єднань, але й спростити її модель. Таким чином, у більшості випадків час навчання CNN можна значно скоротити. Зокрема, коли зображення є вхідним сигналом CNN, зображення можна ввести безпосередньо в нейронну мережу, щоб уникнути кількох складних етапів, таких як вилучення ознак і реконструкція даних. Завдяки перевагам розподілу

ваги, об’єднання та локального сприйнятливого поля, CNN має надійну продуктивність у кількох операціях трансформації зображення, наприклад перетворення, обертання та масштабування. Для кращого розуміння, в продовжені цього розділу коротко представлено попередні дані CNN.

1.1. Рівні згорткової нейронної мережі

Згорткові шари нейронної мережі поділяються на три типи: згортковий, об’єднуючий і повністю зв’язаний. Кожен шар відіграє різну роль. Архітектура CNN показана на рисунку 2.1.

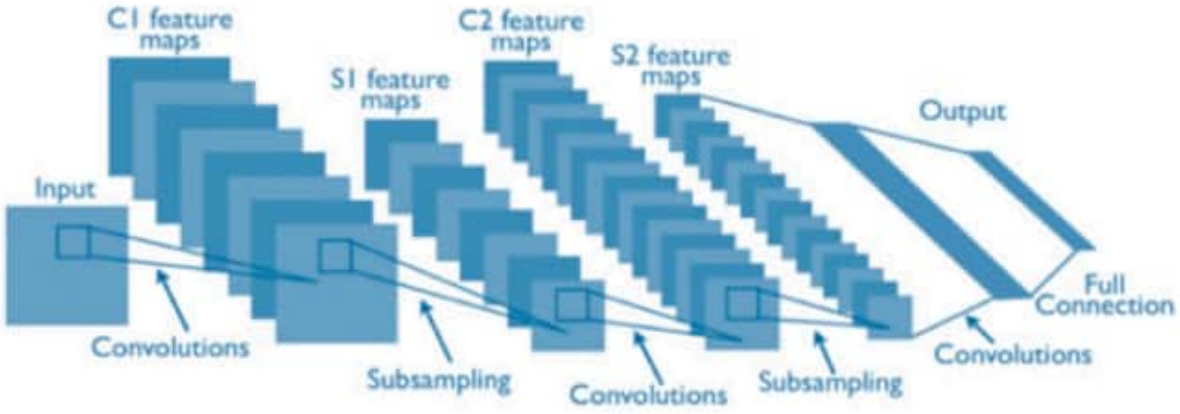

Рис 2.1. Типова архітектура згорткової мережі.

Згортковий рівень відомий як елементарний блок розробки для CNN [15]. У технології CNN дуже важливо розуміти, що параметри шарів складаються з набору фільтрів або нейронів, які можна навчати. Ці фільтри мають невелике сприйнятливе поле, але вони проходять через весь вхідний обсяг. У процесі прямого проходу кожен окремий фільтр проходить по ширині та висоті вхідного об’єму, обчислюючи скалярний добуток із записів фільтра та вхідних даних. Результатом цього обчислення є двовимірна карта активації цього фільтра. Завдяки цьому мережа вивчає фільтри, створені у момент, коли вона знаходить певний тип об’єкта в просторовому місці в межах вхідних даних карти об’єктів X, генеруючи карту об’єктів зважених підсумків Y. Кожен із нейронів обчислює згортки з невеликими областями в X, як показано у рівнянні 1.

$$y_i = b_i + \sum_{x_i \in X} \omega_{ij} * x_i \ (1)$$

, де $y_j \in Y$, j = 1, 2, ..., D. D — це глибина згорткового шару, а кожен фільтр ω_{ij} — тривимірна матриця розміром [F × F × C_x]. Її розмір визначається вибраним рецептивним полем (F) і глибиною введення карти функцій (C_x); наприклад, якщо сприйнятливе поле становить п’ять пікселів, а вхід X карти функцій є зображенням RGB розміром [32 × 32 × 3], тоді розмір фільтра буде [5,5,3].

Розмір фільтра представляє кількість вагових коефіцієнтів, які має нейрон, що з’єднується з регіоном у вхідних даних. Згортковий рівень має перевагу використання тих самих нейронів для кожного пікселя в шарі для покращення продуктивності системи. До того ж, це призводить до зменшення пам’яті сліду, що робить його ефективним.

Шари об’єднання відповідають за регулювання розмірів ширини за висотою матриці шляхом зменшення просторових розмірів вхідного об’єму для наступного згорткового шару без впливу на розмірну глибину об’єму. Процес, який виконується рівнем об’єднання, також відомий як зменшення дискретизації або субдискретизація, оскільки зменшення розміру призводить до одночасної втрати інформації, що приносить користь мережі. Зменшення стає менш обчислювальним, оскільки інформація просувається до наступних шарів об’єднання, і воно також працює проти надмірної підгонки. Найпоширенішими стратегіями, що використовуються в мережах рівнів об’єднання, є максимальне об’єднання та об’єднання середніх [16].

Повністю зв’язані шари (FC) – це шари, де виконуються рівні високого міркування, ці шари пов'язують вихідні нейрони із вхідними. Фільтри та нейрони в цьому шарі пов’язані з усією активацією попередніх шарів, що призводить до повних зв’язків, як випливає з їх назви. Обчислення на цьому рівні виконуються шляхом множення матриці з подальшим зміщенням. Шар FC проходить процес, який перетворює двовимірну карту об’єктів у вектор об’єктів. Крім того, вектор, сформований у цьому процесі, або класифікується як класи для класифікації або вектор ознак піддається подальшій обробці [17].

1.2. Функція активації

Продуктивність нейронної мережі тісно пов’язана не лише з її структурою, але й із прийнятою функцією активації, яка зазвичай вибирається як нелінійна функція для вирішення деяких складних питань. Три функції активації, які часто використовуються в CNN, це сигмоїдна, гіперболічний тангенс (tanh) і випрямлена лінійна одиниця (ReLU), які можна сформулювати таким поданням:

$$sigmoid: f(x) = \frac{1}{1+e^{-x}} \quad (1);$$

$$tanh: f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad (2);$$

$$ReLU: f(x) = (0, x) \quad (3).$$

Дані функції проілюструвати на рисунку 2.2.

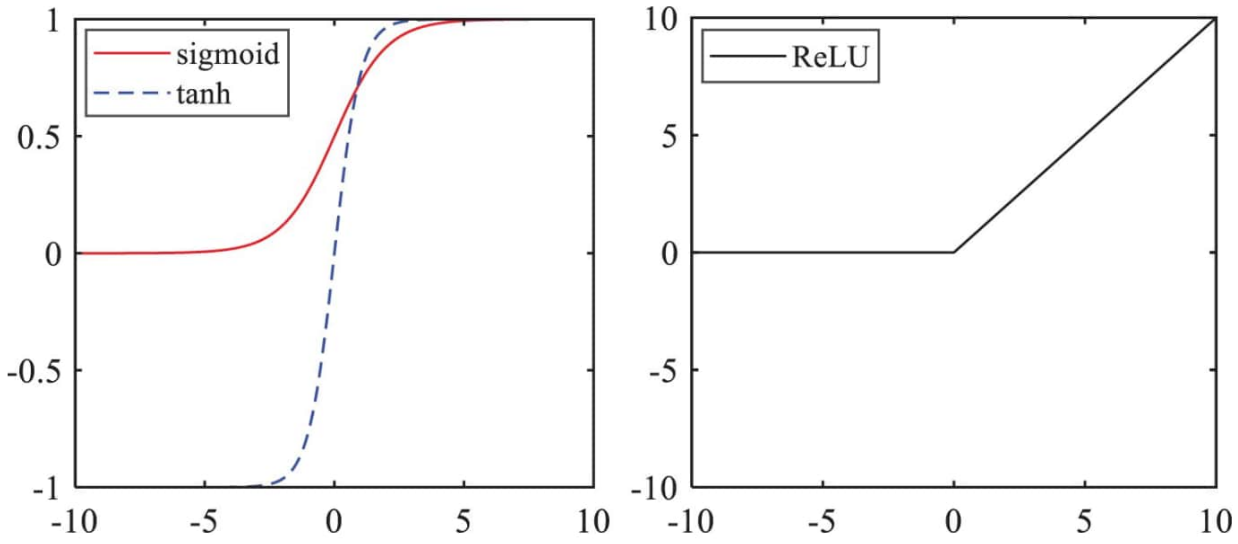

Рис 2.2. Функції активації.

1.3. Алгоритм зворотного поширення

Алгоритм зворотного розповсюдження (BP) є одним із найбільш часто використовуваних алгоритмів для навчання нейронної мережі, а відображення вхідних і вихідних даних фактично є проблемою нелінійної оптимізації ваг з’єднання [18]. На основі градієнтного спуску (GD) алгоритму BP вагові коефіцієнти з’єднання нейронної мережі можна оновлювати ітераційним шляхом мінімізації середньої квадратичної помилки (MSE) між реальним і

очікуваним значеннями результату. Тут MSE, яка зазвичай визначається як функція витрат у навчанні нейронної мережі, може бути виражена як:

$$E(W,B) = \frac{1}{N_L}\sum_{i=1}^{N_L}\left(a_{iL} - t_{iL}\right)^2 \quad (4)$$

, де W і B позначають, відповідно, матриці ваги та зсуву, які необхідно оптимізувати в нейронній мережі; a_{iL} і t_{iL} вказують, відповідно, реальні та очікувані вихідні значення i-го нейрона у вихідному шарі з нейронами N_L.

У нейронній мережі, показаній на рисунку 2.3, вихід i-го нейрона в l-му шарі можна обчислити наступним чином:

$$a_{il} = f_{il}\left(\sum_{j=1}^{n_{l-1}} w_{ijl}a_{j,l-1} + b_{il}\right) \quad (5)$$

, де $f_{il}(\cdot)$ та b_{il} – функція активації та зміщення a_{il} відповідно; w_{ijl} позначає вагу зв'язку між i-м нейроном l-го шару та j-м нейроном попереднього шару; l вказує на номер шару L-рівня нейронної мережі; N_l – кількість нейронів l-го шару. На основі прямого поширення вхідних даних і функції вартості, визначеної вище, ваги та зміщення нейронної мережі можна оновлювати ітераційно наступним чином:

$$w_{ijl}(k+1) = w_{ijl}(k)+\Delta w_{ijl}(k) \quad (6),$$

$$b_{il}(k+1) = b_{il}(k)+\Delta b_{il}(k) \quad (7),$$

, де Δw_{ijl} і Δb_{il} можна обчислити згідно з ланцюговим правилом обчислення наступним чином:

$$\Delta w_{ijl} = -\eta\frac{\delta E}{\delta w_{ijl}} = -\eta\frac{\delta E}{\delta a_{il}}\frac{\delta a_{il}}{\delta w_{ijl}} \quad (8),$$

$$\Delta b_{il} = -\eta\frac{\delta E}{\delta b_{il}} = -\eta\frac{\delta E}{\delta a_{il}}\frac{\delta a_{il}}{\delta b_{il}} \quad (9)$$

, де η – швидкість навчання, яка може контролювати швидкість градієнтного спуску. Необхідно зауважити, що згадані вище ланцюжки не розгорнуті в деталях задля простоти.

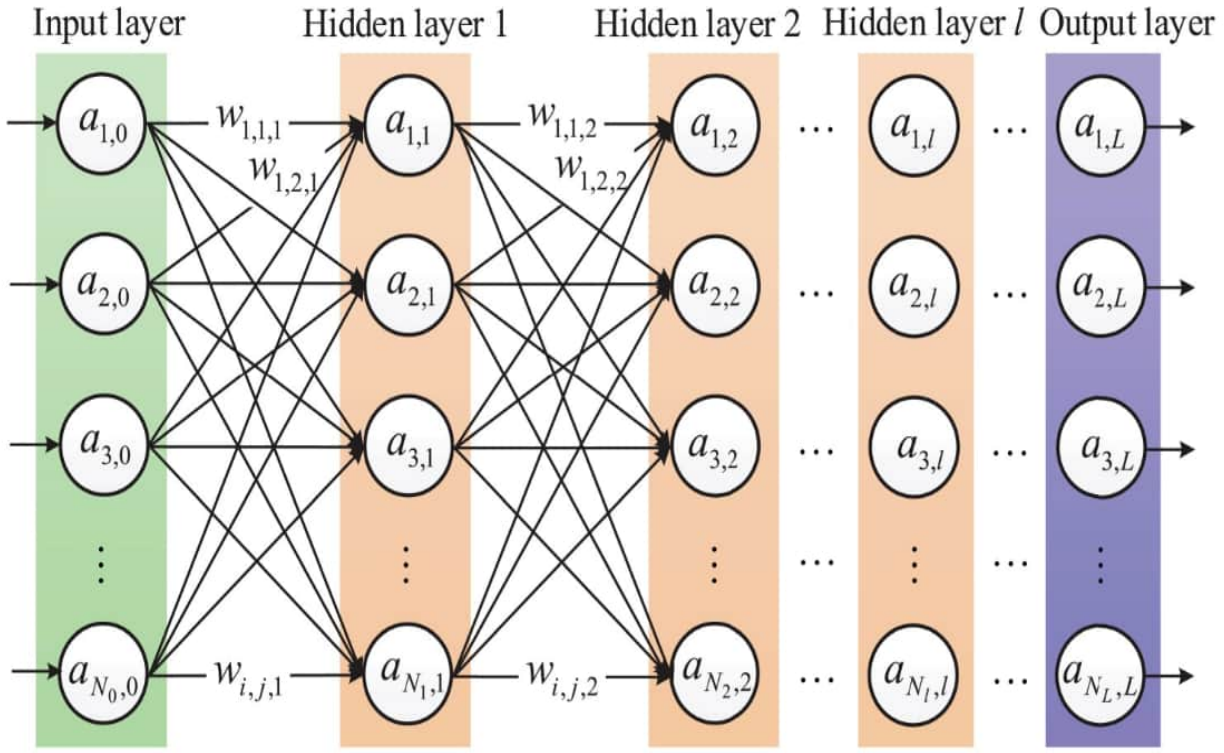

Рисунок 2.3. Структура нейронної мережі.

1.4. Операція згортки

Згортка — це різновид математичної операції, яка широко використовується в обробці зображень. Результат згортки можна відсортувати за трьома режимами, тобто режимами «Повний», «Одноманітний» і «Дійсний», які можна використовувати в різних випадках. Наприклад, дійсний режим зазвичай використовується для прямого розповсюдження, щоб полегшити виділення ознак зображення, а повний режим часто використовується у зворотному поширенні для отримання оптимальних ваг.

В операції згортки операція обнулення краю реалізована для вхідного зображення, де розмірність шару краю може бути визначена відповідно до розміру ядра згортки. Метою обнулення країв є забезпечення раціональності результатів, тобто елементи вхідного зображення та ядра згортки можуть бути зважені та підсумовані послідовно. Крім того, ядро згортки слід повертати вгору та вниз, як показано на малюнку 2.4, де ядро фактично повернуто на 180 градусів навколо центру.

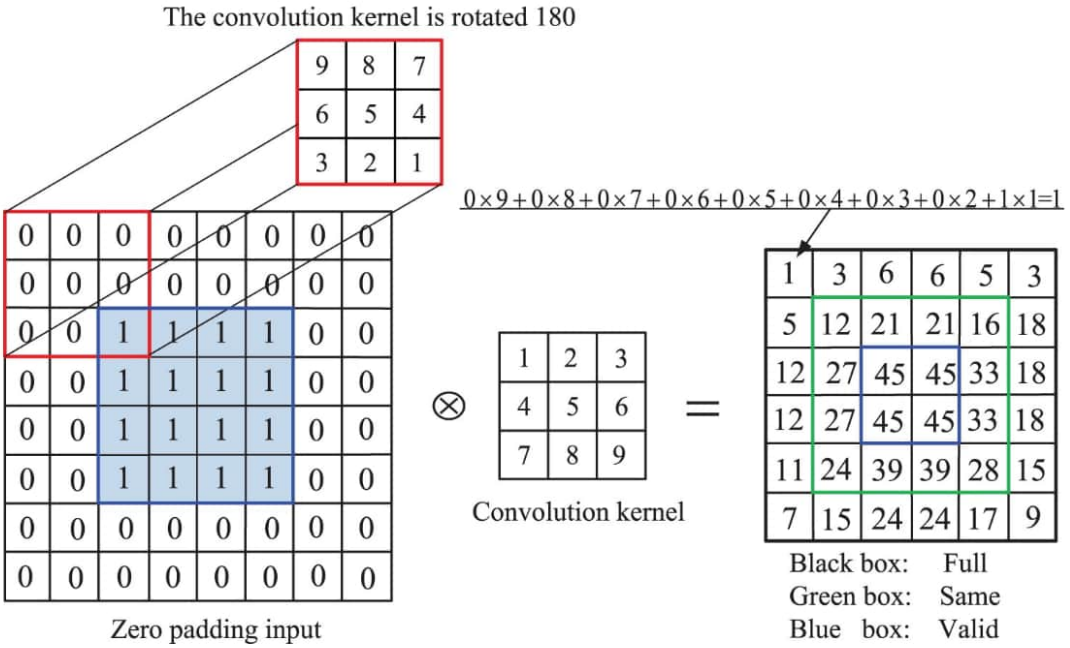

Рисунок 2.4. Операція згортки.

Варто зазначити, що операція згортки може досягти розрідженого множення та спільного використання параметрів, що може стискати розмірність вхідних даних. У порівнянні з DNN, для CNN немає необхідності надавати ваги з'єднань окремо для всіх нейронів вхідних даних. Насправді CNN можна розглядати як звичайний процес вилучення ознак, як і більшість нейронних мереж, які використовуються для вилучення ознак.

1.5. Рецептивне поле

У CNN рецептивне поле — це локальне поле з'єднання нейрона в прихованому шарі [19]. Припустимо, що вхід нейронної мережі – це зображення розміром 100 × 100 пікселів і 100 нейронів у прихованому шарі, між вхідним і прихованими шарами буде 100 × 100 × 100 ваг зв'язку, якщо кожен піксель зображення підключений до всіх нейронів прихованого шару, як показано на малюнку 2.5(a). Немає сумніву, що величезне обчислювальне навантаження знизить ефективність навчання нейронної мережі. Навпаки, якщо кожен прихований нейрон підключено до локального поля вхідного зображення (наприклад, 10 × 10 пікселів), як показано на малюнку 2.5 (b), кількість ваг зв'язку буде зменшено до 10 × 10 × 100, що є 1/100 повного корпусу підключення.

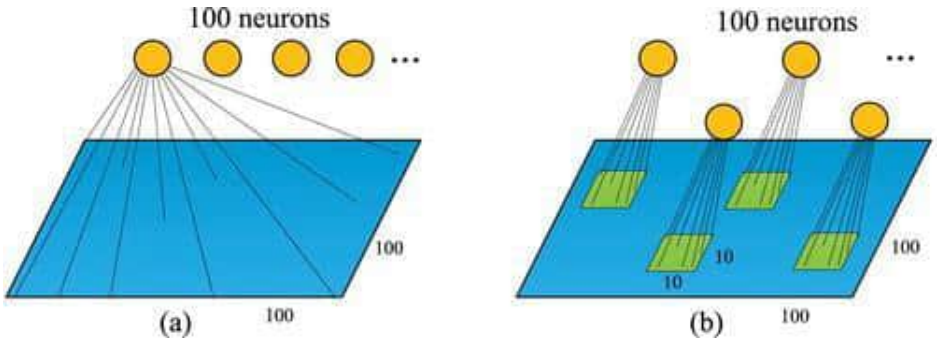

Рисунок 2.5. Рецептивне поле.

На практиці ваги з’єднань можна додатково зменшити за допомогою методу спільної ваги, тобто всі нейрони мають однакові ваги в одному ядрі згортки. Таким чином, кількість ваг з’єднання може бути зменшена з 10×10×100 до 10×10. Відповідно, швидкість навчання нейронної мережі також може бути значно підвищена.

1.6. Об’єднання(pooling)

Шари об’єднання, які зазвичай розташовані за згортковими шарами, в основному використовуються для стиснення вихідних даних об’єктів згорткових шарів. Після шару об’єднання покращені результати виведення можуть зменшити ймовірність надмірного підгонки нейронної мережі [20]. Крім того, характеристика зображення може бути додатково виділена через операцію об’єднання, не впливаючи на отримання інформації про зображення.

Насправді об’єднання — це редукована обробка зображення, яку можна класифікувати як об’єднання середнього значення, об’єднання максимального значення, об’єднання з перекриттям, стохастичне об’єднання та об’єднання глобального середнього значення. Наприклад, об’єднання середнього значення може витягти середнє значення точок ознак та матиме ефект збереження відносного фону; в той час як max-pooling може отримати максимальне значення точок ознак і досягти кращого вилучення текстури. Зокрема, для об’єднання середніх значень, якщо карта функцій розміром 4 × 4 вибирається за допомогою ядра розміром 2 × 2, результатом буде карта функцій розміром 2 × 2, як показано на малюнку 2.6.

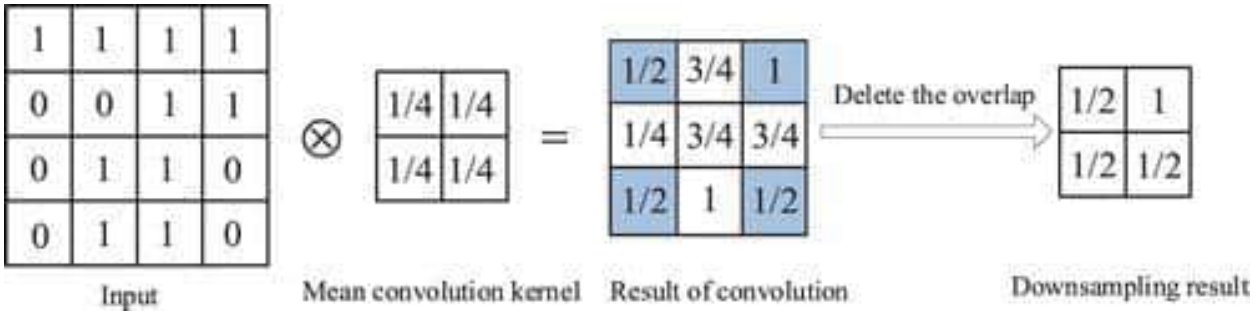

Рисунок 2.6. Операція об'єднання.

Операція об’єднання включає ковзання двовимірного фільтра по кожному каналу карти функцій і узагальнення функцій, що знаходяться в межах області, охопленої фільтром.

Шари об’єднання використовуються для зменшення розмірів карт об’єктів. Таким чином, це зменшує кількість параметрів для вивчення та обсяг обчислень, що виконуються в мережі.

Рівень об’єднання підсумовує об’єкти, наявні в області карти об’єктів, згенерованої шаром згортки. Отже, подальші операції виконуються над узагальненими об’єктами замість точно розташованих об’єктів, згенерованих шаром згортки. Це робить модель більш стійкою до змін у положенні елементів на вхідному зображенні.

Типи шарів об’єднання:

Max Pooling (максимальне об’єднання) — це операція об’єднання, яка вибирає максимальний елемент із області карти об’єктів, охопленої фільтром [23]. Таким чином, результатом використання шару максимального об’єднання буде карта функцій, яка містить найбільшу функції попередньої карти функцій.

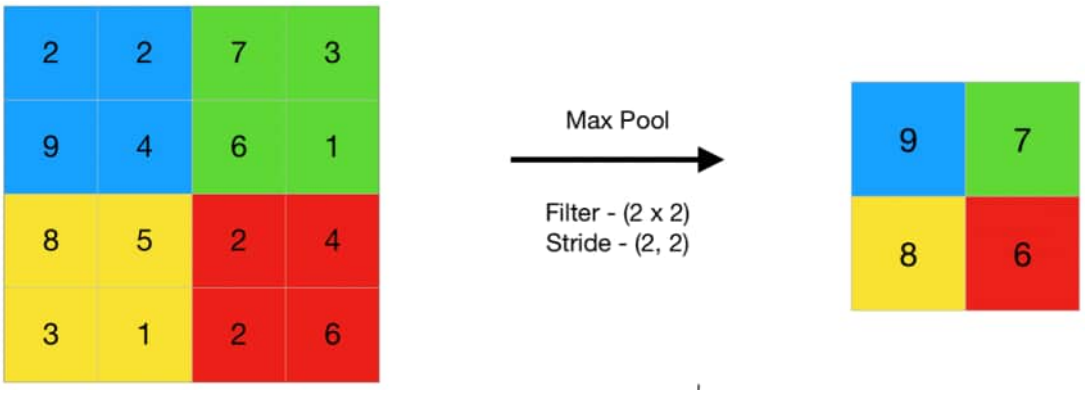

Рисунок 2.7. Приклад роботи максимального об’єднання

Average Pooling (об’єднання середніх значень) — обчислює середнє значення елементів, присутніх в області карти об’єктів, охопленої фільтром. Таким чином, у той час як максимальне об’єднання дає найбільшу функцію в певному патчі карти функцій, середнє об’єднання дає середнє значення функцій, присутніх у пакеті даних.

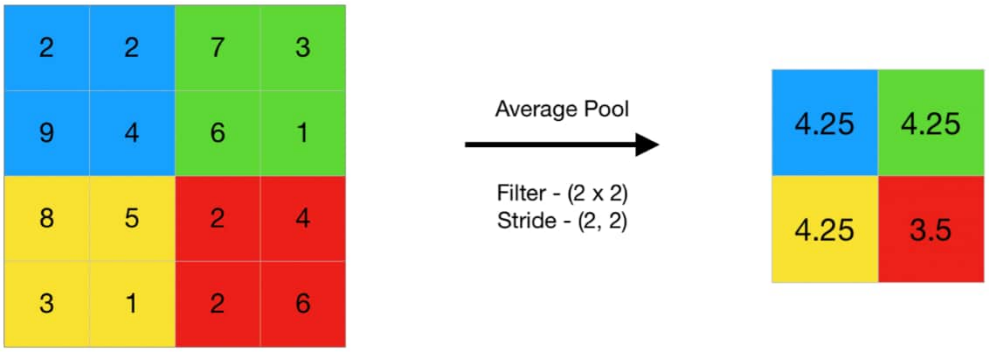

Рисунок 2.8. Приклад роботи об’єднання середніх значень

Global Pooling (глобальне об’єднання) — зменшує кожен канал у карті функцій до одного значення. Таким чином, карта функцій зменшується до карти ознак. Це еквівалентно використанню фільтра розмірів, тобто розмірів карти функцій.

Крім того, це може бути або глобальне максимальне об’єднання, або глобальне об’єднання середніх значень.

1.7. Структуровані результати

Згорткові мережі можна використовувати для виведення багатовимірного структурованого об’єкта, а не просто для прогнозування мітки класу для завдання класифікації або реального значення для завдання регресії. Зазвичай цей об’єкт є просто тензором, випущеним стандартним згортковим шаром. Наприклад, модель може видавати тензор S, де $S_{i,j,k}$ є ймовірністю того, що піксель (j, k) входу в мережу належить до класу «i». Це дозволяє моделі позначати кожен піксель на зображенні та малювати точні маски, які повторюють контури окремих об’єктів [20].

Проблема, яка часто виникає, полягає в тому, що вихідна площина може бути меншою за вхідну. У типах архітектур, які зазвичай використовуються для класифікації одного об’єкта в зображенні, найбільше зменшення просторових

розмірів мережі відбувається завдяки використанню шарів об’єднання з великим кроком. Щоб створити вихідну карту такого ж розміру, як і вхідна, можна взагалі уникнути об’єднання. Інша стратегія полягає в тому, щоб просто випромінювати сітку міток з нижчою роздільною здатністю. Нарешті, в принципі, можна використовувати оператор об’єднання з одиничним кроком.

Однією зі стратегій піксельного маркування зображень є створення початкового припущення міток зображення, а потім уточнення цього початкового припущення за допомогою взаємодії між сусідніми пікселями. Повторення цього кроку уточнення кілька разів відповідає використанню однакових згорток на кожному етапі, розподіляючи ваги між останніми шарами глибокої мережі. Це робить послідовність обчислень, що виконуються послідовними згортковими шарами з вагами, розподіленими між шарами, певним типом рекурентної мережі.

Після того, як зроблено прогноз для кожного пікселя, можна використовувати різні методи для подальшої обробки цих прогнозів, щоб отримати сегментацію зображення на регіони. Загальна ідея полягає в припущенні, що великі групи суміжних пікселів мають тенденцію бути пов’язаними з тією самою міткою. Графічні моделі можуть описувати ймовірнісні зв’язки між сусідніми пікселями. Як альтернатива, згорткову мережу можна навчити, щоб максимізувати наближення до цілі навчання графічної моделі.

2. Метод розпізнавання обличчя

Розпізнавання обличчя відіграє ключову роль у FER. Існують різні стратегії розпізнавання обличчя, включаючи підхід на основі вираження, підхід на основі рамок, підхід на основі ознак і підхід на основі графа сусідства. Трьох етапна схема процесу розпізнавання виразу обличчя показана на рис. 2.9.

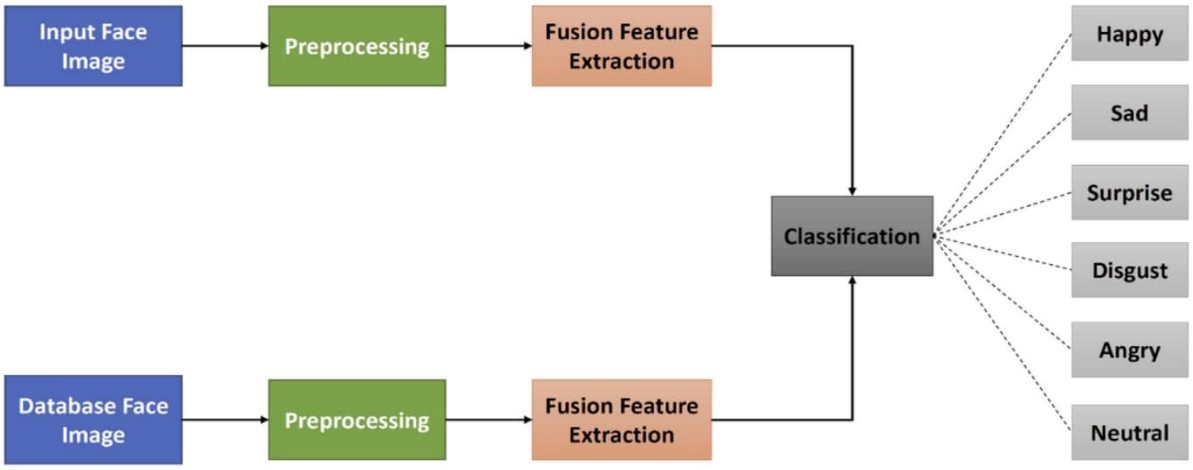

Рисунок 2.9. Зведена блок-схема для трьох етапів методу розпізнавання виразу обличчя (FER).

Спочатку ми бачимо простий шаблон CNN з кількома блоками, які ми можемо легко зрозуміти та співвіднести із запропонованою моделлю CNN. Три типи шарів складають базову CNN, як показано на рис. 2.10, вхідний, прихований і вихідний [24]. Дані надходять у CNN через вхідний рівень, а потім проходять через багато прихованих рівнів, перш ніж досягти вихідного рівня. Прогноз мережі відображається через вихідний рівень. З точки зору втрати або помилки вихід мережі порівнюється з фактичними мітками.

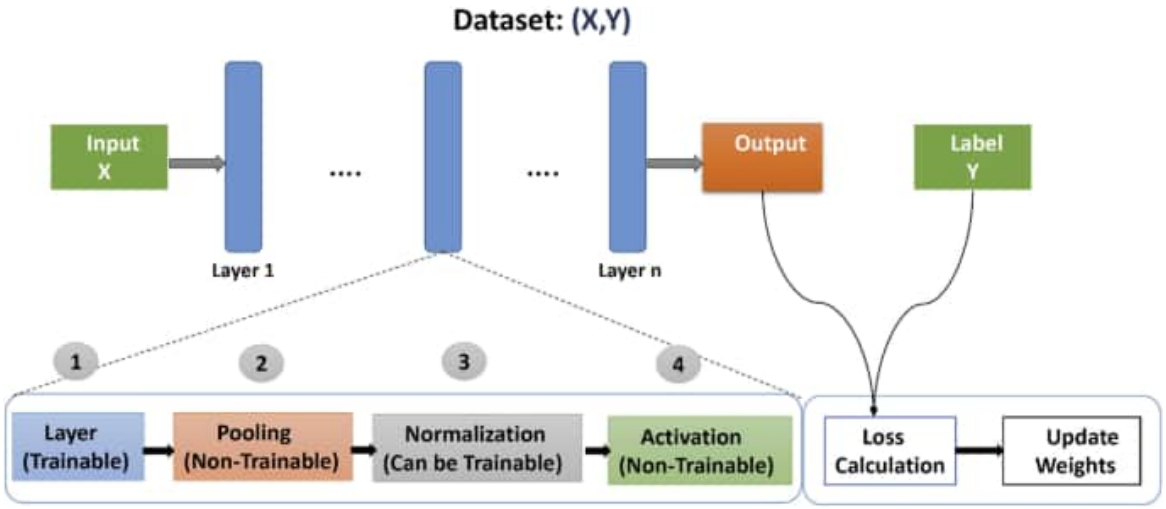

Рисунок 2.10. Шаблон базової CNN (простий шаблон CNN із кількома блоками).

Приховані рівні в мережі діють як базові елементи для перетворення даних. Архітектура згорткової нейронної мережі містить рівні, які складаються із чотирьох підфункцій: функції рівня, об’єднання, нормалізація та активація,

які можуть бути виділені з кожного рівня. У перспективі даної магістерської роботи, я зверну більше уваги параметрам моделі CNN.

У випадку згорткової нейронної мережі головною метою згортки є виділення ознак із вхідного зображення. Вивчаючи характеристики зображення (тобто фільтруючи крихітні зони пікселів вхідних даних), згортка зберігає просторове співвідношення між пікселями. CNN використовує терміни «детектор функцій», «фільтр» та «ядро». «Карта активації», «Згорнута функція» або «Карта функцій» — це матриця, яка створюється шляхом переміщення фільтра по зображенню та обчислення скалярного добутку. У цьому шарі вибирається розмір фільтра та крок (кількість пікселів, при досягненні цієї кількості необхідно змінити фільтр). (W–F + 2P)/S + 1 = O — результат згорткового шару, де F - просторова протяжність, P, S означає крок і відступ відповідно. Якщо було вибрано неправильний крок, тоді вхідні нейрони розташовуються не рівномірно. У такому випадку, використовується нульове заповнення для всього зображення, щоб правильно підігнати нейрони.

Кожне негативне значення із відфільтрованого зображення вилучається та замінюється нулем у цьому шарі. Цей підхід використовується для запобігання нульової суми значень. Функція перетворення Rectified Linear Unit активує вузол, лише якщо вхідні дані перевищують певний поріг; інакше вихід дорівнює нулю. Значення функції активації ReLu полягає в тому, що її градієнт завжди дорівнює 1 (як це показано у формулі 3), це означає, що під час зворотного поширення більша частина помилки передається назад.

Метою шару об’єднання є мінімізація просторової розмірності виправленої карти об’єктів, що призводить до більш компактного вилучення об’єктів. Об’єднана карта представлення є результатом роботи цього шару. Дві найпоширеніші стратегії об’єднання — середнє та максимальне об’єднання. У технології Max Pooling [23] максимальний параметр вилучається на кожному кроці, а решта параметрів зменшуються. Також, це призводить до зменшення мережі. Щоб обчислити рівень об’єднання, використовується формула:

(I + 2P − F)/S + 1 = O, де I - вхідна матриця, F, S, P означає фільтр , крок та відступ відповідно.

Повністю зв'язаний шар перетворює об’єднану карту об’єктів із двовимірної структури на одновимірний вектор або вектор об’єктів. Результатом є об’єднана «зведена» карта функцій. Для класифікації цей вектор ознак діє як стандартний повністю зв’язаний рівень.

Softmax реалізовано через рівень нейронної мережі безпосередньо перед вихідним рівнем. Як вихідний рівень, рівень Softmax повинен мати однакову кількість вузлів.

Пакетний нормалізатор прискорює процес навчання та додає перехід, який підтримує середнє значення активації близьким до 0 і стандартне відхилення активації близько до 1.

3. Метод розпізнавання виразу обличчя

Механізм розпізнавання обличчя має три етапи. По-перше, етап попереднього створення — на цьому етапі відбувається підготування набору даних та їх перетворення до певного вигляду. Після цього, використовується узагальнений алгоритм для перетворених даних, щоб отримати ефективні результати.

По-друге, відбувається ідентифікація обличчя використовуючи зображення, отримані у режимі реального часу, під час етапу виділення ознак. Зрештою, етап класифікації емоцій, він необхідний для того, щоб згрупувати, віднести зображення в один із семи класів, цей етап складається із застосування алгоритму згортки. Крім того, щоб продемонструвати ці етапи для класифікації емоцій можна використати системні блок-схеми. Для прикладу, блок-схема для підходу FER зображена на рис. 2.11.

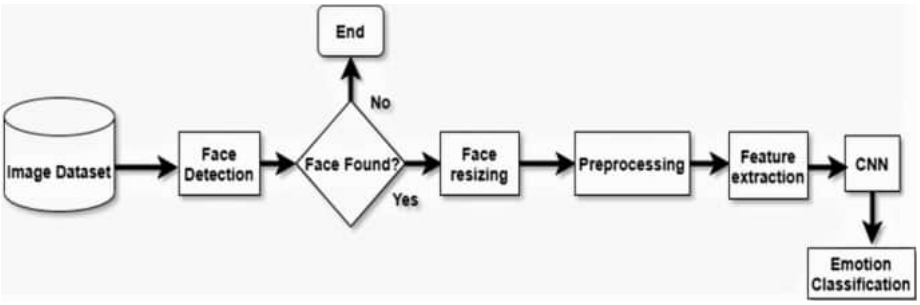

Рисунок 2.11. Системна блок-схема розробленого методу для класифікації емоцій [25].

3.1. Попередня обробка

Для обробки зображень попередня обробка є обов’язковим етапом [24]. Якщо йдеться про зображення, введене в набір даних, воно може містити деякі варіації масштабу, кольору та світла, а також шум. Також, варто зазначити, що усі операції попередньої обробки використовуються для того, щоб забезпечити швидшу та надійнішу роботу за допомогою підходу згорткової нейронної мережі. Для трансформації зображення, використовуються такі основні прийоми попередньої обробки:

- **Нормалізація** — для отримання покращеного зображення та усунення відмінностей використовується нормалізація зображення.
- **Змінення розмірів** — для того, щоб покращити швидкодію та зменшити необхідну пам'ять, потрібно змінити зображення видаливши ту його частину, у якій, немає необхідності.
- **Відтінки сірого** — у цьому методі, значення пікселів зображення залежать від інтенсивності світла на ньому. Варто наголосити, що кольорові зображення є значно складнішими для обробки цим алгоритмом. Оскільки метою є класифікація та розпізнавання виразів обличчя незалежно від багатьох чинників зовнішності, у тому числі кольору шкіри, необхідно, щоб цей процес також був незалежний від кольору вхідного зображення. Для цього потрібно перетворити вхідне зображення на 1-канальне сіре зображення.

3.2. Розпізнавання обличчя

Для того, щоб мати змогу розпізнати вираз обличчя необхідно його ідентифікувати, це є обов'язковим кроком для будь якої системи. Таке рішення є ефективним для виявлення об’єктів та було запропоноване у 2001 році в статті «Швидке виявлення об’єктів за допомогою розширеного каскаду простих

функцій» авторами якої є Пол Віола та Майкл Джонс[34]. У процесі виявлення об’єкта на відео чи зображенні, які містять багато як негативних, так і позитивних зображень використовуються класифікатори, які вивчають та опраьоують каскадну функцію. Також було досліджено, що для досягнення високої точності та ефективного виявлення об’єктів на зображені слід використовувати каскади Хаара. Для прикладу, цей метод використовують для виявлення брів, які представлені темними областями на лобі. Каскади Хаара є ефективними у виявленні на зображенні області обличчя та видалені із зображення зайвих фонових даних[25]. У бібліотеці алгоритмів та функцій комп'ютерного зору - OpenCV, було представлено механізм, який використовує каскадні класифікати Хаара для виявлення обличчя. На рис. 2.12 продемонстровано процес виявлення компонентів обличчя на зображенні, у цьому прикладі використовується підхід прямокутних характеристик.

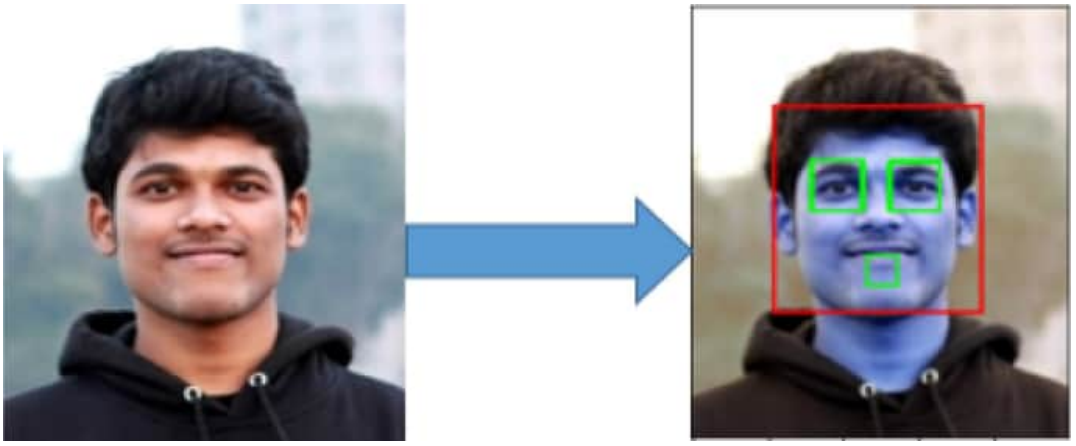

Рисунок 2.12. Вибір різних компонентів обличчя щасливого зображення в процесі виявлення обличчя.

3.3. Вилучення ознак

Через його простоту та довговічність я використовую локальний бінарний шаблон як підхід до виділення ознак, який широко використовується для виділення ознак у різних алгоритмах ідентифікації об’єктів, а також для розпізнавання виразу обличчя з ефективним результатом класифікації. Я помітив, що «уніфіковані» візерунки становлять переважну більшість візерунків для кожного зображення у використаній колекції. Нижче наведено кроки, щоб реалізувати вилучення функцій:

- Розбиття зображення обличчя на маленькі сегменти.
- Для кожного регіону обчислюється гістограма LBP. Для отримання гістограми LBP кожного регіону, виконується її сканування за допомогою оператора LBP.
- Об’єднання гістограми ознак LBP кожного регіону в єдиний вектор ознак.

Гістограма LBP є хорошим індикатором змін зовнішнього вигляду, оскільки вона поєднує інформацію про положення, форму та текстуру. Вектор ознак має бути зменшений на основі підходів лінійного програмування, щоб класифікація була швидшою. На рисунку 2.13 показано характеристики LBP і те, як оператор LBP може перетворити оригінальне зображення на зображення LBP, а також виявляє гістограму характеристик LBP [26].

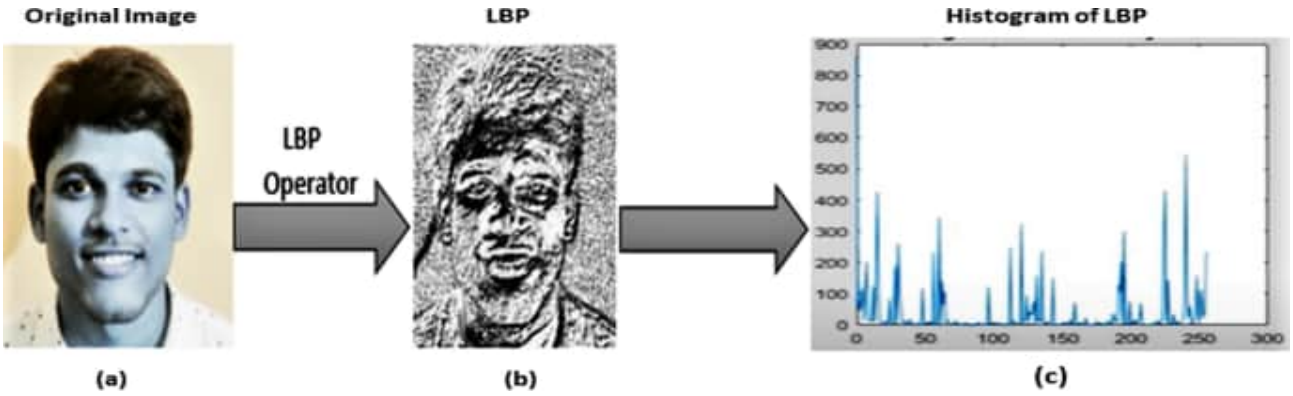

Рисунок 2.13. Демонстрація отримання гістограми LBP, де (a) Виявлене обличчя з вихідного зображення. (b) LBP зображення. (c) Характерна гістограма зображення LBP.

3.4. Класифікація емоцій

На цьому етапі розроблена програма класифікує та розподіляє зображення за одним із семи, зазначених у наборі даних, універсальних виразів обличчя: «Нейтральний», «Щасливий», «Здивований», «Сумний», «Наляканий», «Злий» та «Відчуття огиди». Для навчання використовується згорткова нейронна мережа, яка є представлена сукупністю нейронних мереж. Спочатку, навчаємо набір даних на навчальному полігоні. Для отримання кращої точності для тестувальної вибірки, було проведено експеримент із застосуванням різних архітектур CNN. Процес класифікації емоцій складається із таких етапів:

- Розподіл даних: Перш за все, необхідно розділити початковий набір даних на наступні категорії:
    - Навчання - набір, який використовується для навчання та створення моделі.
    - Тестування - набір даних, необхідний для оцінки ефективності та “навченості” алгоритмів моделі, цей набір також є частиною створення моделі.
    - Приватне тестування - цей набір необхідний для перевірки моделі.

- Навчання та створення моделі: На цьому етапі відбувається навчання нейронної мережі на відповідних наборах даних, використовуючи запропоновану модель. Після процесу навчання, генеруються оновлення, необхідні для покращення моделі, які застосовують до початкової структури.

- Оцінка моделі: Результатом попереднього етапу є оновлення моделі, яке оцінюється, використовуючи набір даних отриманий під час начання моделі.

- Класифікація тестових даних(зображень) використовуючи отриману модель: Завершивши попередні етапи, отримуємо модель, яка складається із відповідних значень і ваг. Цю модель можна застосувати, щоб виявити нові емоції на зображеннях, до того ж, процес розпізнавання буде значно швидшим, адже вона вже містить вагові коефіцієнти.

РОЗДІЛ 3. ВИКОРИСТАНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ. РОЗРОБКА ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ ЗА ДОПОМОГОЮ МОВИ ПРОГРАМУВАННЯ PYTHON

Відповідно до Python Software Foundation, «Python — це інтерпретована, об’єктно-орієнтована мова програмування високого рівня з динамічною семантикою» [28]. При детальному розборі мови Python, можна виділити наступні характеристики:

- Однією із головних особливостей провідних мов програмування, таких як Python, є об'єктно-орієнтованість. Ідея цього підходу полягає в тому, що програма, яка створюється, розбивається на окремі, незалежні об’єкти, між якими реалізовано “спілкування”. Об’єктно-орієнтовані мови спрощують створення, налагодження та підтримку програмного забезпечення, а написаний код є більш структурованим.
- Python — це одна із мов високого рівня, тобто вона містить фрази та слова, які є зрозумілі для людини. Python використовує інтерпретатор для того, щоб перевести вхідний код, написаний на мові високого рівня, у, зрозумілий комп’ютерам, машинний код. Python, як будь-яка інтерпретована мова, є легким у налагодженні, надає розробникам можливість реалізовувати завдання різної складності лише за декілька кроків, ефективно використовуючи пам’ять та швидко редагуючи код.
- Останньою характеристикою є динамічність мови Python. Це дає можливість розробнику не звертати увагу на оголошення типів змінних, адже у цьому немає потреби. Цей процес організований таким чином, що при запуску програми, Python проводить перевірку типів і можливих помилок та завершує її до початку перетворення вхідного коду у машинний, тобто компіляції. Такий підхід дозволяє збільшити швидкість розробки, покращити стійкість та гнучкість і, звісно, зменшити кількість коду.

Python є багатоцільовою мовою, і її можна використовувати для створення практично будь-чого. Компанії в усьому світі використовують Python для

штучного інтелекту та машинного навчання, розробки веб-сайтів, наукових і чисельних обчислень, ігор та багатьох інших цілей.

Що стосується штучного інтелекту, Python стоїть вище інших мов програмування та вважається найкращою мовою програмування для додатків на основі ШІ. Насправді Python вирішує будь-яке завдання штучного інтелекту: машинне навчання, аналіз і візуалізація даних, обробка природної мови та комп'ютерне бачення.

Є десятки компаній, які використовують Python у тій чи іншій формі через його гнучкість, масштабованість, продуктивність і швидкий розвиток. Таку популярність мова високого рівня Python отримала за рахунок низки своїх переваг, які я наводжу нижче.

Потужний і швидкий

Мова Python є неймовірно потужною, адже стандартні бібліотеки пропонують неймовірну кількість функцій за замовчуванням, що робить можливим її використання для вирішення будь-якого завдання програмування. Використовуючи Python та готові рішення, які він надає, розробник значно скоротить зусилля та час, необхідні для реалізації поставленого завдання, незалежно від того, чи це розроблення протоколів та інтерфейсів операційної системи, оброблення і класифікація зображень чи виконання наукових обчислень.

До того ж, вищезгадані переваги мови Python, стають ще важливішими, якщо вам необхідно адаптувати продукт спираючись на відгуки користувачів або отримати швидку окупність інвестицій. Це стає можливим завдяки пришвидшені розробки із використанням готових рішень.

Кросплатформний

Python реалізований та працює на таких платформах: IBM, VMS, а також AIX. Також ця мова є доступною для всіх операційних систем, серед яких найпопулярніші Windows, UNIX, Linux, Android та macOS

Сумісність з іншими мовами

Функціональність Python є достатньою для взаємодії з кодом та фреймворками інших мов програмування. Ви можете поєднає Python із різними мовами, шляхом вбудування проекту, створеного на одній із них у фреймворк Python або навпаки. Таким чином, можна отримати проект, який буде містити переваги декількох мов.

Легкий та доступний у навчанні

Завдяки зрозумілості та простоті синтаксису, Python вважають одним із найефективніших способів кодування.

Для прикладу, одним із аспектів простоти синтаксису є те, що для позначення початку та кінця конструкцій програмування використовують пробіли, а не спеціальні символи такі, як дужки. Завдяки таким спрощенням, якщо певний проект потребує оновлення, значно легше знайти особу, яка зможе розібратися із функціоналом програми і внести необхідні корективи , навіть якщо вона бачить цей код вперше. Саме завдяки таким особливостям, досягається зрозумілість програмного коду та легкість вивчення мови Python.

Відкритий

Python — це мова з відкритим кодом, яка адмініструється та підтримується незалежною некомерційною організацією: Python Software Foundation. Однією з головних переваг програмного забезпечення з відкритим кодом є те, що його можна вільно використовувати, змінювати та поширювати.

Оскільки спільнота Python відкрита для всіх і заохочує різноманітність, мільйони досвідчених розробників у всьому світі роблять свій внесок і доповнюють базу даних Python з відкритим кодом, що зростає.

Є багато конференцій і зустрічей, а також багато спільної роботи над кодом. Це означає, що Python менш схильний до помилок і більш безпечний, ніж деякі інші популярні мови. Крім того, це допомагає мінімізувати витрати на розробку.

Python є багатоцільовою мовою, і її можна використовувати для створення практично будь-чого. Компанії по всьому світу використовують Python для штучного інтелекту та машинного навчання, розробки веб-сайтів, наукових і чисельних обчислень, ігор та багатьох інших цілей.

Що стосується штучного інтелекту, Python стоїть вище інших мов програмування та вважається найкращою мовою програмування для додатків на основі ШІ. Насправді Python справляється з будь-якими завданнями ШІ: машинним навчанням, аналізом і візуалізацією даних, обробкою природної мови та комп'ютерним баченням.

Є десятки компаній, які використовують Python у тій чи іншій формі через його гнучкість, масштабованість, продуктивність і швидкий розвиток. Нижче наведено приклади підприємств, які широко використовують Python:

- Google:

  "Python був важливою частиною Google із самого початку й залишається нею, оскільки система росте й розвивається. Сьогодні десятки інженерів Google використовують Python, і ми шукаємо більше людей, які володіють цією мовою." — Пітер Норвіг, директор з досліджень Google.

Однією з найбільших компаній, що покладаються на Python, є Google. Перша версія пошукової системи Google і весь стек технологій були насправді написані на Python. Сьогодні Python є однією з трьох основних мов програмування, які використовує Google разом із Java та C++.

Google долучився до безлічі сфер: пошуку, електронної пошти, потокового передавання музики та відео, обладнання, безпілотних автомобілів і реклами. Насправді весь стек технологій YouTube написаний на Python. Спочатку YouTube був написаний на PHP. Однак через швидке зростання кількості користувачів і необхідність швидкої розробки та впровадження нових функцій YouTube незабаром перейшов на Python. Куонг До — архітектор програмного забезпечення в YouTube — каже, що «Python достатньо швидкий для нашого сайту та дозволяє нам створювати функції, які можна підтримувати, за рекордно короткий час із мінімумом розробників».

- Dropbox:

Dropbox — це хмарна платформа для зберігання документів, музики, зображень, відео тощо. Програма Dropbox доступна на більшості пристроїв і пропонує користувачам доступ будь-де та будь-коли. Dropbox із самого початку використовує Python для всього стеку технологій. По-перше, це був код Python з використанням PyWin32: однієї з бібліотек Python для спеціальних функцій Windows. Пізніше вони перейшли на Python 2, а потім на Python 3. Зараз додаток Dropbox використовує налаштовану версію Python 3.5. Вони також використовують структуру Django, яка підтримує функції зберігання файлів, синхронізацію облікових записів і обмін файлами.

- Facebook:

Ще одним великим гравцем, який використовує Python, є Facebook. Разом із PHP і C++, Python допомагає Facebook підтримувати, керувати та масштабувати свою інфраструктуру ефективно та надійно.

Facebook також використовує Python для машинного навчання. У Facebook алгоритми машинного навчання використовуються для контролю контенту в стрічці новин і виявлення об’єктів на фотографіях. Користувачів Facebook впізнають на фотографіях, навіть якщо вони не позначені. А для людей з вадами зору Facebook описує фотографії словами.

1. Python як засіб розробки штучного інтелекту

Проекти, які використовують штучний інтелект відрізняються від традиційних програмних проектів. Відмінності полягають у наборі технологій, навичках, необхідних для створення проекту на основі ШІ, і необхідності глибоких досліджень. Щоб реалізувати свої прагнення щодо штучного інтелекту, вам слід використовувати стабільну, гнучку мову програмування та доступні інструменти. Python пропонує все це, тому сьогодні ми бачимо багато проектів Python AI [28].

Від розробки до розгортання та обслуговування, Python допомагає розробникам бути продуктивними та впевненими щодо програмного забезпечення, яке вони створюють. Переваги, завдяки яким Python найкраще

підходить для машинного навчання та проектів на основі ШІ, включають простоту та узгодженість, доступ до незамінних бібліотек і фреймворків для ШІ та машинного навчання (ML), гнучкість, незалежність від платформи та широке співтовариство, про які я розповідав вище.

Python пропонує стислий і читабельний код. Хоча за машинним навчанням і штучним інтелектом стоять складні алгоритми та різноманітні робочі процеси, простота Python дозволяє розробникам створювати надійні системи. Замість того, щоб зосереджуватися на технічних нюансах мови, розробники докладають усіх зусиль для вирішення проблеми ML.

Крім того, Python привабливий для багатьох розробників, оскільки його легко вивчити. Код Python зрозумілий людям, що полегшує створення моделей для машинного навчання.

Багато програмістів кажуть, що Python більш інтуїтивно зрозумілий, ніж інші мови програмування. Інші вказують на безліч фреймворків, бібліотек і розширень, які спрощують реалізацію різних функцій. Загальновизнано, що Python підходить для спільного впровадження, коли задіяно кілька розробників. Оскільки Python є мовою загального призначення, він може виконувати низку складних завдань машинного навчання та дає змогу швидко створювати прототипи, які дозволяють тестувати ваш продукт для цілей машинного навчання.

Впровадження алгоритмів AI і ML може бути складним і вимагає багато часу. Дуже важливо мати добре структуроване та перевірене середовище, щоб дозволити розробникам пропонувати найкращі рішення програмування [28].

Щоб скоротити час розробки, програмісти звертаються до ряду фреймворків і бібліотек Python. Бібліотека програмного забезпечення — це попередньо написаний код, який розробники використовують для вирішення типових завдань програмування. Python зі своїм багатим набором технологій має великий набір бібліотек для штучного інтелекту та машинного навчання. Ось деякі з них:

- Keras, TensorFlow і Scikit-learn для машинного навчання

- SciPy для передових обчислень
- NumPy для високопродуктивних наукових обчислень і аналізу даних
- Seaborn для візуалізації даних
- Pandas для аналізу даних загального призначення
- Scikit-learn містить різноманітні алгоритми класифікації, регресії та кластеризації, включаючи машини опорних векторів, посилення градієнта, k-середні та DBSCAN, ця бібліотека розроблена для роботи з числовими та науковими бібліотеками Python NumPy та SciPy.

За допомогою цих рішень ви можете швидше розробляти свій продукт. Вашій команді розробників не доведеться заново винаходити колесо, і вона може використати наявну бібліотеку для впровадження необхідних функцій.

Нижче наведена таблиця 1 поширених випадків використання ШІ та технологій, які найкраще підходять для них. Рекомендовано використовувати такі:

Аналіз та візуалізація даних	Numpy, scipy, pandas, seaborn
Машинне навчання	Tensorflow, Keras, Scikit-learn
Комп'ютерний зір	Opencv
Обробка природної мови	NLTK, spacy

Таблиця 1. Області застосування різних фреймворків мови Python

Мова Python незалежна від платформи. Незалежність від платформи стосується мови програмування або фреймворку, що дозволяє розробникам впроваджувати речі на одній машині та використовувати їх на іншій без будь-яких (або лише з мінімальними) змінами. Одним із ключових факторів популярності Python є те, що це незалежна від платформи мова. Python підтримується багатьма платформами, включаючи Linux, Windows і macOS. Код Python можна використовувати для створення автономних виконуваних програм для більшості поширених операційних систем, що означає, що програмне забезпечення Python можна легко поширювати та використовувати в цих операційних системах без інтерпретатора Python.

Більше того, розробники зазвичай використовують такі служби, як Google або Amazon для своїх обчислювальних потреб. Однак ви часто можете знайти компанії та дослідників даних, які використовують власні машини з потужними графічними процесорами (GPU) для навчання своїх моделей ML. А той факт, що Python не залежить від платформи, робить це навчання набагато дешевшим і простішим.

В опитуванні розробників 2020 року, проведеному Stack Overflow, Python увійшов до 5-ти найпопулярніших мов програмування, що зрештою означає, що ви можете знайти та найняти компанію-розробника з необхідним набором навичок для створення вашого проекту на основі ШІ.

Згідно із опитуванням розробників Python у 2020, Python зазвичай використовується для веб-розробки. На перший погляд, переважає веб-розробка, на яку припадає понад 26% випадків використання. Однак, якщо поєднати науку про дані та машинне навчання, вони складають приголомшливі 27%.

Онлайн-репозиторії містять понад 140 000 спеціально створених пакетів програмного забезпечення Python. Наукові пакети Python, такі як Numpy, Scipy та Matplotlib, можна інсталювати в програмі, що працює на Python. Ці пакети забезпечують машинне навчання та допомагають розробникам виявляти шаблони у великих наборах даних. Python настільки надійний, що Google використовує його для сканування веб-сторінок, Pixar використовує його для створення фільмів, а Spotify використовує його для рекомендацій пісень.

Добре відомий факт, що спільнота Python AI розширилася по всьому світу. Є форуми Python і активний обмін досвідом, пов’язаним з рішеннями машинного навчання. Для будь-якого завдання, яке ви можете мати, досить висока ймовірність того, що хтось інший мав справу з такою ж проблемою. Ви можете отримати поради та вказівки від розробників. Ви не будете самотні й обов’язково знайдете найкраще рішення для своїх конкретних потреб, звернувшись до спільноти Python.

2. Інші мови програмування штучного інтелекту

Штучний інтелект все ще розвивається та росте, і є кілька мов, які домінують у розробці. Тому я наведу список мов програмування, які надають розробникам екосистеми для створення проектів за допомогою ШІ та машинного навчання. Давайте розглянем ключові із них:

- R мова програмування:

R зазвичай застосовується, коли вам потрібно аналізувати та маніпулювати даними для статистичних цілей. У R є такі пакети, як Gmodels, Class, Tm і RODBC, які зазвичай використовуються для створення проектів машинного навчання. Ці пакети дозволяють розробникам впроваджувати алгоритми машинного навчання без зайвих проблем і дозволяють їм швидко впроваджувати бізнес-логіку.

R був створений статистиками для задоволення своїх потреб. Ця мова може дати вам поглиблений статистичний аналіз незалежно від того, чи працюєте ви з даними із пристрою чи аналізуєте фінансові моделі.

Більше того, якщо ваше завдання потребує високоякісних графіків і діаграм, ви можете використовувати R. Завдяки ggplot2, ggvis, googleVis, Shiny, rCharts та іншим пакетам можливості R значно розширюються, допомагаючи вам перетворювати візуальні елементи на інтерактивні веб-програми.

Порівняно з Python, R має репутацію повільного та відстаючого, коли йдеться про великомасштабні дані. Для фактичної розробки продукту краще використовувати Python або Java з їх гнучкістю.

- Scala:

Scala є безцінною, коли йдеться про великі дані. Він пропонує дослідникам обробки даних набір інструментів, таких як Saddle, Scalalab і Breeze. Scala має чудову підтримку паралелізму, що допомагає обробляти великі обсяги даних. Оскільки Scala працює на JVM, вона виходить за межі всіх можливостей разом із Hadoop, структурою розподіленої обробки з відкритим кодом, яка керує обробкою та зберіганням даних для програм великих даних, що працюють у кластерних системах. Незважаючи на меншу кількість

інструментів машинного навчання порівняно з Python і R, Scala легко підтримувати.

- Julia:

Якщо вам потрібно створити рішення для високопродуктивних обчислень і аналізу, ви можете розглянути Джулію. Julia має синтаксис, подібний до Python, і розроблений для обробки чисельних обчислювальних завдань. Джулія забезпечує підтримку глибокого навчання через оболонку TensorFlow.jl і фреймворк Mocha.

Однак ця мова не підтримується багатьма бібліотеками та ще не має сильної спільноти, як Python, оскільки вона відносно нова.

- Java:

Ще одна мова, яку варто згадати, це Java. Java є об’єктно-орієнтованою, портативною, зручною для обслуговування та прозорою. Вона підтримується численними бібліотеками, такими як WEKA і Rapidminer.

Java широко поширена, коли йдеться про обробку природної мови, алгоритми пошуку та нейронні мережі. Вона дозволяє швидко створювати великомасштабні системи з відмінною продуктивністю.

Але якщо ви хочете виконувати статистичне моделювання та візуалізацію, тоді Java є останньою мовою, яку ви хочете використовувати. Незважаючи на те, що існують пакети Java, які підтримують статистичне моделювання та візуалізацію, їх недостатньо. Python, з іншого боку, має передові інструменти, які добре підтримуються спільнотою.

Враховуючи описані дослідження, я вважаю, що екосистема Python добре підходить для проектів на основі ШІ. Python з його простотою, великою спільнотою та інструментами дозволяє розробникам створювати архітектури, близькі до досконалості, зосереджуючись при цьому на поставлених завданнях.

При усіх перевагах інших мов, я зупинився на мові високого рівня з відкритим кодом Python за рахунок її переваг та низки фреймворків і бібліотек, які значно полегшують і пришвидшують розробку системи штучного інтелекту. Усі використані бібліотеки я наводжу нижче у розділі 3.

3. Використані фреймворки та бібліотеки

3.1. Пакет Python pandas

Пакет pandas є найважливішим інструментом у розпорядженні спеціалістів із обробки даних та аналітиків, які сьогодні працюють на Python. Потужне машинне навчання та найрізноманітніші інструменти візуалізації можуть привернути всю увагу, але pandas є основою більшості проектів обробки даних.

Якщо ви думаєте про роботу з науковими даними, то в першу чергу важливо вивчити pandas. У цій частині ми розглянемо основну інформацію про pandas, включаючи те, як його встановити, використовувати та як він працює з іншими поширеними пакетами аналізу даних Python, такими як matplotlib і scikit-learn.

Pandas має так безліч застосувань, тому доцільно було б перерахувати речі, які він не може робити, а не те, що є можливим. Цей інструмент, є вмістилищем для ваших даних. За допомогою нього ви переглядаєте свої дані, перетворюючи, аналізуючи та очищаючи їх.

Наприклад, скажімо, ви хочете вивчити набір даних, що зберігається у файлі CSV на вашому комп’ютері. Pandas витягне дані з цього CSV у DataFrame — по суті, у таблицю — а потім дозволить вам робити такі речі, як:

- Обчислення статистичних даних та відповість на запитання щодо даних:
    - Яке середнє, максимальне або мінімальне значення кожного стовпця?
    - Чи співвідноситься стовпець A зі стовпцем B?
    - Як виглядає розподіл даних у стовпці C?
- Очищення даних, наприклад видаливши відсутні значення та відфільтрувавши рядки чи стовпці за певними критеріями;
- Візуалізація даних за допомогою Matplotlib. Відрізки, лінії, графіки, гістограми тощо;
- Зберігання очищених, трансформованих даних у CSV, інший файл або базу даних.

Перш ніж приступити до моделювання або складної візуалізації, вам потрібно добре зрозуміти природу вашого набору даних, а pandas — найкращий шлях для цього.

Бібліотека pandas є не лише центральним компонентом набору інструментів для обробки даних, але й використовується в поєднанні з іншими бібліотеками в цій колекції.

Pandas побудовано на основі пакета NumPy, тобто велика частина структури NumPy використовується або копіюється в Pandas. Дані в pandas часто використовуються для статистичного аналізу в SciPy, побудови графіків функцій з Matplotlib і алгоритмів машинного навчання в Scikit-learn.

Jupyter Notebook пропонує збалансоване середовище, для легкого використання pandas для дослідження та моделювання даних, але pandas можна також легко використовувати в текстових редакторах.

Jupyter Notebook дає нам можливість виконувати код у конкретній комірці, а не запускати весь файл. Це економить багато часу при роботі з великими наборами даних і складними перетвореннями. Notebook також забезпечує простий спосіб візуалізації DataFrames і графіків pandas.

Pandas — це пакет, який легко встановити. Відкрийте термінальну програму (для користувачів Mac) або командний рядок (для користувачів Windows, Linux та Ubuntu) і встановіть пакет за допомогою команди «pip install pandas» або у Jupyter Notebook «!pip install pandas» [32].

Двома основними компонентами pandas є Series та DataFrame (рис 3.1).

Series — це, по суті, стовпець, а DataFrame — це багатовимірна таблиця, що складається з колекції Series.

Рис 3.1. Приклад компонент Series та DataFrame.

DataFrames і Series досить схожі в тому, що багато операцій, які ви можете виконувати з одним, ви можете виконувати з іншим, наприклад, заповнення нульових значень і обчислення середнього.

3.2. Модуль Python NumPy

NumPy це модуль із відкритим кодом для python, який надає загальні числові та математичні операції, вони подаються у вигляді швидких, наперед скомпільованих функцій. Вони поєднуються у високо-рівневі пакети даних. Ці операції забезпечують функціонал, схожий до того, який використовується у MatLab. NumPy (Numeric Python) надає базові методи для маніпуляції з великими масивами та матрицями [31]. Використавши SciPy (Scientific Python), ви можете значно розширити функціонал numpy великою колекцією незамінних алгоритмів, таких як перетворення Фур'є, мінімізація, регресія та інших алгоритмів прикладної математики.

Головною особливістю numpy є об'єкт array(масив). Цей об’єкт є схожий зі списками в python, за винятком того, що всі його елементи повинні бути однакового типу даних, наприклад integer або float. Завдяки масивам ви можете проводити, різного роду, числові операції із значно більшим обсягом інформації у рази ефективніше та швидше, ніж зі списками.

NumPy — дуже популярна бібліотека Python для обробки великих багатовимірних масивів і матриць за допомогою великої колекції математичних функцій високого рівня. Це дуже корисно для фундаментальних наукових обчислень у машинному навчанні. Це особливо важливо для використання лінійної алгебри, перетворень Фур’є та роботи із випадковими числами. Бібліотеки високого класу, такі як TensorFlow, приховано використовують NumPy для маніпулювання тензорами.

3.3. Бібліотека Python matplotlib

Бібліотека matplotlib - це бібліотека двовимірної графіки для мови програмування python, за допомогою якої можна створювати високоякісні

графіки різних форматів. Matplotlib представляє собою модульний пакет для python.

Matplotlib складається з безлічі, ієрархічно пов'язаних між собою, модулів, які наповнені різними функціями та класами.

Створення графіка в matplotlib схоже з малюнком в реальному житті. Для прикладу, художнику потрібно взяти інструменти, полотно, мати уявлення про майбутній малюнок (що саме він буде малювати) і, нарешті, поєднати це все і створити рисунок крок за кроком.

Структура matplotlib також містить ці всі етапи, в якості виконавця виступає сама бібліотека. Користувач, у свою чергу, має керувати діями matplotlib, вказуючи, що саме він повинен зобразити та якими інструментами. Щодо процесу непрогнозованого відображення та створення основи малюнка, цей етап повністю контролює matplotlib. Отже, будучи користувачем, вам необхідно лише здійснювати керування бібліотекою matplotlib. І чим краще ви управляєте кінцевим результатом роботи matplotlib, тим краще.

Так як matplotlib організований ієрархічно, а найбільш простими для розуміння людини є високорівневі функції, то знайомство з matplotlib починається з самого високорівневого інтерфейсу matplotlib.pyplot. Так, щоб створити гістограму за допомогою цього модуля, потрібно викликати всього одну команду: matplotlib.pyplot.hist(arr).

Користувачу не потрібно думати, як саме бібліотека буде малювати цю діаграму. Якщо б ми зображали гістограму самостійно, то помітили б, що вона складається з повторюваних у формі фігур - прямокутників. А щоб нарисувати прямокутник, потрібно знати хоча б координату одного кута, а також ширину та довжину.

Цей приклад відображає ієрархічність малюнків, коли підсумкова діаграма (високий рівень) складається з простих геометричних фігур (більш низький, середній рівень), створених декількома універсальними методами

малювання (низький рівень). Якщо б кожен рисунок потрібно було створити ось так, з нуля, це було б дуже довго і неефективно.

Інтерфейс є набором функцій і команд, які виконують синтаксис графічної команди matplotlib.pyplot, схожої на команди, які використовуються в середовищі MATLAB. Початково, matplotlib планувався як безкоштовна альтернатива MATLAB, де в одному середовищі були засоби як для малювання, так і для чисельного аналізу. Саме так в Matplotlib з'явився pylab, який об'єднує модулі pyplot і numpy в одному просторі імен.

Водночас для більш серйозних завдань (внесення matplotlib в графічний інтерфейс користувача) потрібно більше контролю над процесом і більше гнучкості, ніж можуть надати ці два модуля. Необхідний доступ до можливостей нижчого рівня бібліотеки, яка реалізована в об'єктно-орієнтованому стилі. Такий стиль помітно складніший і вимагає знань про роботу конкретних класів та їх методів, але надає найбільші можливості для взаємодії з бібліотекою matplotlib.

Головною одиницею (об'єктом найвищого рівня) при роботі з matplotlib є рисунок. Будь-який рисунок в matplotlib має вкладену структуру. Користувацька робота передбачає операції з різними рівнями цієї структури:

- Малюнок (Figure)

Малюнок є об'єктом найвищого рівня, на якому розташовуються одна або кілька областей малювання (Axes), елементи малюнка Artisits (заголовки, легенда тощо) та основа-полотно (Canvas). На малюнку може бути кілька областей малювання Axes, але усі вони можуть належати лише одному малюнку Figure.

- Область малювання (Axes)

Область малювання є об'єктом середнього рівня, який, мабуть, є головним об'єктом роботи з графікою matplotlib в об'єктно-орієнтованому стилі. Це те, що асоціюється зі словом "plot", це частина зображення з простором даних. Кожна область малювання Axes містить дві (або три у разі тривимірних даних) координатних осі (Axis об'єктів), які впорядковують відображення даних.

- Координатна вісь (Axis)

Координатна вісь є об'єктом середнього рівня, який визначає область зміни даних, на них наносяться поділи ticks та підписи до поділів ticklabels. Розташування поділів визначається об'єктом Locator, а підписи поділів обробляє об'єкт Formatter. Конфігурація координатних осей полягає у комбінуванні різних властивостей об'єктів Locator та Formatter.

- Елементи малюнка (Artists)

Практично все, що відображається на малюнку, є елементом малюнка (Artist), навіть об'єкти Figure, Axes і Axis. Елементи малюнка Artists включають такі прості об'єкти як текст (Text), плоска лінія (Line2D), фігура (Patch) та інші.

Коли відображається малюнок (figure rendering), всі елементи малюнка Artists наносяться на основу-полотно (Canvas). Більшість їх пов'язується з областю малювання Axes. Також елемент малюнка не може спільно використовуватися декількома областями Axes або переміщуватися з однієї на іншу.

Pyplot

Інтерфейс pyplot дозволяє користувачеві зосередитись на виборі готових рішень та налаштуванні базових параметрів малюнка. Це його головна перевага, тому вивчення matplotlib краще починати саме з інтерфейсу pyplot.

Малюнки matplotlib створюються шляхом послідовного виклику команд: або в інтерактивному режимі (в консолі), або в скрипті (текстовий файл з python-кодом). Графічні елементи (точки, лінії, фігури тощо) нашаровуються один на одний послідовно, тобто наступний елемент перекриває попередній, якщо вони займають спільну ділянки малюнку (регулюється параметром **zorder**).

У matplotlib працює правило *"поточної області" ("current axes")*, яке означає, що всі графічні елементи наносяться на поточну область малювання. Незважаючи на те, що областей малювання може бути декілька, одна з них завжди є поточною.

Як сказано вище найголовнішим об'єктом в matplotlib є малюнок Figure. Тому створення наукової графіки слід розпочинати саме зі створення малюнка. Створити малюнок у matplotlib означає задати форму, розміри та властивості основи-полотна (canvas), на якому буде створюватися майбутній графік.

Створити рисунок figure дозволяє метод plt.figure(). Після виклику будь-якої графічної команди, тобто функції, яка створює будь-який графічний об'єкт, наприклад plt.scatter() або plt.plot(), завжди існує хоча б одна область для малювання (за замовчуванням прямокутної форми).

Щоб результат малювання, тобто поточний стан малюнка, відобразився на екрані, можна скористатися командою plt.show(). Буде показано всі малюнки (figures), які були створені.

Зазвичай малюнок в matplotlib являє собою прямокутну область, задану у відносних координатах: від 0 до 1 включно по обох осях. Другий поширений варіант типу малюнка – кругла область (polar plot).

Щоб зберегти малюнок, що вийшов, потрібно скористатися методом plt.savefig(). Цей метод зберігає поточну конфігурацію малюнка у графічний файл із обраним розширенням (png, jpeg, pdf та інші.), який можна задати через параметр fmt. Тому, цю функцію потрібно викликати в кінці вихідного коду після виклику всіх інших команд. Якщо в скрипті python створити кілька малюнків figure і спробувати зберегти їх однією командою plt.savefig(), то буде збережено останній малюнок figure.

Розмаїття і зручність створення графіки в matplotlib забезпечується за рахунок створених графічних команд та багатого арсеналу щодо конфігурації типових форм. Ці налаштування включають роботу з кольором, формою, типом лінії або маркера, товщиною ліній, ступенем прозорості елементів, розміром і типом шрифту та іншими властивостями.

Параметри, які визначають ці властивості у різних графічних командах, зазвичай мають однаковий синтаксис, тобто називаються однаково. Стандартним способом завдання властивостей будь-якого

створюваного об'єкта (або методу) є передача по ключу: ключ = значення. Назви параметрів зміни властивостей графічних об'єктів, що найчастіше зустрічаються, перераховані нижче:

- color/colors/c - колір;
- linestyle - тип лінії;
- linewidth/linewidths - товщина лінії;
- marker – тип маркера;
- alpha – ступінь прозорості (від повністю прозорого 0 до непрозорого 1);
- fontsize - розмір шрифту;
- rotation – поворот рядка на X градусів.
- s - розмір маркера у методі plt.scatter (тільки цифри);

При створенні функцій чи методів класів, особливо у разі успадкування, параметри часто передають як об'єднань послідовностей: кортежу чи словника. Для цього є спеціальні символи-приставки: "*" або "**" відповідно. Це особливо корисно у випадках, коли функція або метод може приймати змінну кількість параметрів.

Прийнято, що з передачі кортежу використовується змінна args, тоді як у словнику - kwargs. Якщо перед змінною args вказано символ "*", всі додаткові аргументи, передані функції або методу, збережуться в args як кортеж. Якщо перед args буде вказано символ "**", всі додаткові параметри будуть розглядатися як пари "ключ - значення" у словнику.

У функціях та методах, які описують властивості таких графічних об'єктів як лінія, текст, прямокутник, параметри часто об'єднують як послідовність *args, чи словників **kwargs. Так зручніше під час створення класів та його методів.

Якщо в описі, графічний метод вказано приблизно так, як, plt.plot(*args, **kwargs), то це означає, що як вхідні дані потрібно спочатку ввести список або кортеж параметрів (найчастіше потрібна хоча б одна послідовність типу значень

функції), а після цього можна передавати значення параметрів за ключовими іменами цих параметрів.

3.4. Фреймворк Python TensorFlow

TensorFlow, створена командою Google Brain і вперше опублікована в 2015 році, є бібліотекою з відкритим кодом для чисельних обчислень і великомасштабного машинного навчання. TensorFlow об’єднує низку моделей і алгоритмів машинного та глибокого навчання (так званих нейронних мереж) і робить їх корисними за допомогою загальних програмних метафор. Він використовує Python або JavaScript, щоб забезпечити зручний інтерфейсний API для створення додатків, одночасно виконуючи ці додатки на високопродуктивній C++ [30].

TensorFlow, який конкурує з такими фреймворками, як PyTorch і Apache MXNet, може навчати та запускати глибокі нейронні мережі для класифікації рукописних цифр, розпізнавання зображень, вбудовування слів, рекурентних нейронних мереж, моделей послідовності для машинного перекладу, обробки природної мови та Моделювання на основі PDE (рівняння з частковими похідних). Найкраще те, що TensorFlow підтримує масштабне прогнозування виробництва з тими самими моделями, що використовуються для навчання.

TensorFlow також має широку бібліотеку попередньо навчених моделей, які можна використовувати у власних проектах. Ви також можете використовувати код із створених моделей TensorFlow як приклади найкращих практик для навчання власних моделей.

TensorFlow дозволяє розробникам створювати графи потоків даних — структури, які описують, як дані переміщуються через граф або ряд вузлів обробки. Кожен вузол на графі представляє математичну операцію, а кожне з’єднання або ребро між вузлами є багатовимірним масивом даних або тензором.

Програми TensorFlow можна запускати майже на будь-якій зручній платформі: локальній машині, кластері в хмарі, пристроях iOS і Android,

процесорах або графічних процесорах. Якщо ви використовуєте власну хмару Google, ви можете запустити TensorFlow на спеціальному модулі обробки TensorFlow (TPU) Google для подальшого прискорення. Однак отримані моделі, створені TensorFlow, можна розгорнути на будь-якому пристрої, де вони використовуватимуться для надання прогнозів.

TensorFlow 2.0, випущений у жовтні 2019 року, багато в чому оновив фреймворк на основі відгуків користувачів, щоб зробити його легшим для роботи (як приклад, за допомогою відносно простого Keras API для навчання моделей) і більш продуктивним. Розподілене навчання легше проводити завдяки новому API, а підтримка TensorFlow Lite дає змогу розгортати моделі на більшій різноманітності платформ. Однак код, написаний для попередніх версій TensorFlow, потрібно переписати, щоб максимально використати переваги нових функцій TensorFlow 2.0.

Використання TensorFlow з Python

TensorFlow надає весь свій функціонал для програміста за допомогою мови Python. Python простий у вивченні та з ним легко працювати, і він надає зручні способи вираження того, як абстракції високого рівня можна об’єднати. TensorFlow підтримується на версіях Python від 3.7 до 3.10, і хоча він може працювати на попередніх версіях Python, частина функціоналу може бути недоступною [30].

Вузли та тензори в TensorFlow є об’єктами Python, а програми TensorFlow самі є програмами Python. Фактичні математичні операції, однак, не виконуються в Python. Бібліотеки перетворень, доступні через TensorFlow, написані як високопродуктивні двійкові файли C++. Python просто спрямовує трафік між частинами та забезпечує абстракції програмування високого рівня, щоб з’єднати їх разом.

Робота високого рівня в TensorFlow — створення вузлів і шарів і їх зв’язування — використовує бібліотеку Keras. Keras API зовні простий; базову модель із трьома шарами можна визначити менш ніж за 10 рядків коду, а навчальний код для цього займає ще кілька рядків. Якщо ви хочете детальніше

розібратися і виконати більш тонку роботу, наприклад, написати власну тренувальну петлю, ви зможете це зробити, приділивши трохи більше часу цій бібліотеці.

Перевага застосування TensorFlow

Найбільша перевага TensorFlow для розробки машинного навчання — це абстракція. Замість того, щоб мати справу з дрібними деталями реалізації алгоритмів або з'ясуванням належних способів зв'язати вихідні дані однієї функції з вхідними даними іншої, розробник може зосередитися на загальній логіці програми. TensorFlow подбає про деталі за лаштунками.

TensorFlow пропонує додаткові зручності для розробників, яким потрібно налагодити додатки TensorFlow і отримати самоаналіз. Кожну операцію з графіком можна оцінювати та змінювати окремо та прозоро, замість того, щоб будувати весь графік як один непрозорий об'єкт і оцінювати його весь одразу. Цей так званий «режим активного виконання», наданий як опція в старіших версіях TensorFlow, тепер є стандартним.

Набір візуалізацій TensorBoard дозволяє перевіряти та профілювати те, як працюють графіки, за допомогою інтерактивної веб-панелі. Служба Tensorboard.dev (розміщена Google) дозволяє розміщувати та ділитися експериментами машинного навчання, написаними на TensorFlow. Його можна використовувати безкоштовно зі сховищем до 100 млн скалярів, 1 ГБ тензорних даних і 1 ГБ даних двійкових об'єктів. (Зверніть увагу, що будь-які дані, розміщені в Tensorboard.dev, є загальнодоступними, тому не використовуйте їх для конфіденційних проектів.)

TensorFlow також отримує багато переваг завдяки підтримці комерційної організації в списку Google. Google сприяв швидкому розвитку проекту та створив багато важливих пропозицій, які полегшують розгортання та використання TensorFlow. Вищезгаданий підхід для прискореної роботи в хмарі Google є лише одним із прикладів.

Навчання детермінованої моделі за допомогою TensorFlow

Кілька деталей реалізації TensorFlow ускладнюють отримання повністю детермінованих результатів навчання моделі для деяких навчальних завдань. Іноді модель, навчена на одній системі, дещо відрізнятиметься від моделі, навченої на іншій, навіть якщо вони отримують однакові дані. Причини такої різниці є неочивидними — одна з причин полягає в тому, як і де засипаються випадкові числа; інша пов'язана з певною недетермінованою поведінкою під час використання GPU. Гілка TensorFlow 2.0 має можливість увімкнути детермінізм у всьому робочому процесі за допомогою кількох рядків коду. Однак ця функція має низьку продуктивність, і її слід використовувати лише під час налагодження робочого процесу.

Порівняння TensorFlow з PyTorch, CNTK і MXNet

TensorFlow конкурує з безліччю інших фреймворків машинного навчання. PyTorch, CNTK і MXNet є трьома основними фреймворками, які задовольняють багато однакових потреб. Ось основні переваги та недоліки цих фреймворків у порівнянні із TensorFlow:

- **PyTorch** створено на Python і має багато інших подібностей до TensorFlow: компоненти з апаратним прискоренням, високо-інтерактивну модель розробки та багато корисних компонентів, які вже включені. PyTorch, як правило, є кращим вибором для швидкої розробки проектів, які потрібно запустити за короткий час, але TensorFlow виграє для великих проектів і складніших робочих процесів.
- **CNTK**, Microsoft Cognitive Toolkit, схожий на TensorFlow у використанні графової структури для опису потоку даних, але він зосереджений переважно на створенні нейронних мереж глибокого навчання. CNTK швидше виконує багато завдань нейронної мережі та має ширший набір API (Python, C++, C#, Java). Але наразі його не так просто освоїти чи розгорнути, як TensorFlow. Він також доступний лише за ліцензією GNU GPL 3.0, тоді як TensorFlow доступний за більш ліберальною ліцензією Apache. Також,

підтримка CNTK є на нижчому рівні; останній великий апдейт був у 2019 році.

- **Apache MXNet**, використовується Amazon як провідний фреймворк глибокого навчання на AWS, може масштабуватися майже лінійно на кількох графічних процесорах і на кількох машинах. MXNet також підтримує широкий спектр мовних API — Python, C++, Scala, R, JavaScript, Julia, Perl, Go — хоча з його рідними API не так приємно працювати, як із TensorFlow. Він також має набагато меншу спільноту користувачів і розробників.

3.5. Python sklearn

Sklearn або scikit-learn у Python є, безумовно, однією з найкорисніших доступних бібліотек з відкритим кодом, які можна використовувати для машинного навчання в Python. Бібліотека scikit-learn — це вичерпна колекція найефективніших інструментів для статистичного моделювання та машинного навчання. Деякі з цих інструментів включають регресію, класифікацію, зменшення розмірності та кластеризацію.

Бібліотека scikit-learn в основному написана на Python і побудована на SciPy, NumPy і Matplotlib. Бібліотека використовує уніфікований і узгоджений інтерфейс Python для реалізації різних алгоритмів попередньої обробки, машинного навчання, візуалізації та перехресної перевірки.

Спочатку відомий як scikit-learn, sklearn у Python був розроблений Девідом Курнапо в 2007 році в рамках проекту Google. Згодом Гаель Варокво, Фабіан Педрегоза, Александр Грамфор і Вінсент Мішель з Французького інституту досліджень комп'ютерних наук і автоматизації публічно випустили бета-версію v0.1 у 2010 році.

Відтоді було випущено нові версії scikit-learn, остання версія 0.23.1 була випущена в травні 2020 року. Scikit-learn — це проект, керований спільнотою, і будь-хто може зробити свій внесок у його розвиток. Microsoft, Intel і NVIDIA є одними з головних спонсорів проекту.

Бібліотека машинного навчання scikit-learn у Python містить багато функцій для спрощення машинного навчання. Тут я згадю деякі з них:

- **Алгоритми керованого навчання:** будь-який алгоритм машинного навчання, про який ви, можливо, чули, має дуже високу ймовірність належати до бібліотеки scikit-learn. Набір інструментів scikit-learn має репертуар таких алгоритмів навчання під наглядом, який включає – узагальнені лінійні моделі, такі як лінійна регресія, дерева рішень, опорні векторні машини та байєсівські методи.
- **Алгоритми неконтрольованого навчання:** ця колекція алгоритмів включає факторизацію, кластерний аналіз, аналіз головних компонентів і неконтрольовані нейронні мережі.
- **Вилучення функцій:** за допомогою scikit-learn ви можете витягувати функції з тексту та зображень.
- **Перехресна перевірка:** точність і валідність контрольованих моделей на невидимих даних можна перевірити за допомогою scikit-learn.
- **Зменшення розмірності**: за допомогою цієї функції можна зменшити кількість атрибутів у даних для подальшої візуалізації, узагальнення та вибору ознак.
- **Кластеризація:** ця функція дозволяє групувати дані без міток.
- **Методи ансамблю:** за допомогою цієї функції можна об’єднати прогнози кількох контрольованих моделей.

Перш ніж почати використовувати останню версію scikit-learn, необхідно переконатися, що встановлені наступні бібліотеки:

- Python (>=3,5)
- NumPy (>= 1.11.0)
- SciPy (>= 0,17,0)li
- Joblib (>= 0,11)
- Matplotlib (>= 1.5.1): ця бібліотека потрібна для можливостей побудови графіків scikit-learn.
- Pandas (>= 0.18.0): це потрібно для структури та аналізу даних.

Плюси scikit-learn:

- Бібліотека розповсюджується за ліцензією BSD, що робить її безкоштовною з мінімальними юридичними та ліцензійними обмеженнями.
- Він простий у використанні.
- Бібліотека scikit-learn є дуже універсальною та зручною та служить для цілей реального світу, як-от прогнозування поведінки споживачів, створення нейро-зображень тощо.
- Scikit-learn підтримується та оновлюється численними авторами, співавторами та великою міжнародною онлайн-спільнотою.
- Веб-сайт scikit-learn надає детальну документацію API для користувачів, які хочуть інтегрувати алгоритми зі своїми платформами.

Серед мінусів зазвичай виділяють те, що scikit-learn не найкращий вибір для поглибленого навчання.

Також потрібно врахувати, що Scikit-learn оптимізований для навчання алгоритмів, таких як опорні векторні машини (SVM), логістична регресія та лінійний дискримінантний аналіз (LDA). Він не оптимізований для графових алгоритмів і не дуже добре обробляє рядки. Наприклад, scikit-learn не надає вбудованого способу створення простої хмари слів. Scikit-learn не має потужної бібліотеки лінійної алгебри, тому використовуються sciру та numpy. Він не містить бібліотеки малювання, але дозволяє використовувати різні бібліотеки малювання.

Зростання та популярність мови машинного навчання потребує ефективних інструментів, а sklearn у Python задовольняє потреби початківців, а також тих, хто вирішує проблеми навчання під наглядом. Ефективність і універсальність використання роблять scikit-learn одним із найкращих виборів академічних і промислових організацій для виконання різноманітних операцій.

РОЗДІЛ 4. ДЕМОНСТРАЦІЯ РОЗРОБЛЕНОГО ДЕТЕКТОРА ЕМОЦІЙ

1. Навчання згорткової нейронної мережі

У якості навчальних даних була використана відкрита база даних набору облич fer2013.

Дані складаються із зображень облич у градаціях сірого розміром 48x48 пікселів. Обличчя було автоматично зареєстровано таким чином, що обличчя розташоване більш-менш по центру та займає приблизно однакову кількість місця на кожному зображенні.

Завдання полягає в тому, щоб розділити кожне обличчя на одну з семи категорій на основі емоцій, відображених у виразі обличчя (0=Злість, 1=Огида, 2=Страх, 3=Щастя, 4=Сум, 5=Подив, 6=Нейтральний). Навчальний набір складається з 28 709 прикладів, а відкритий тестовий набір складається з 3 589 прикладів.

Дані зберігаються у форматі csv та зчитуються за допомогою пакетного модулю pandas (рис 4.1)

```
In [3]: dataset = pd.read_csv('fer2013.csv')
        print("Shape of dataset:",dataset.shape)
        dataset.head()

        Shape of dataset: (35887, 3)
```

Out[3]:

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

Рис 4.1. Зчитування даних навчання

Навчальні дані несуть у собі інформацію про кожен піксель зображення розміру 48x48 пікселів з ознакою (рис 4.3), яка емоція зображена на даному екземплярі(рис. 4.2).

Рис 4.2. Навчальні дані у вигляді зображень

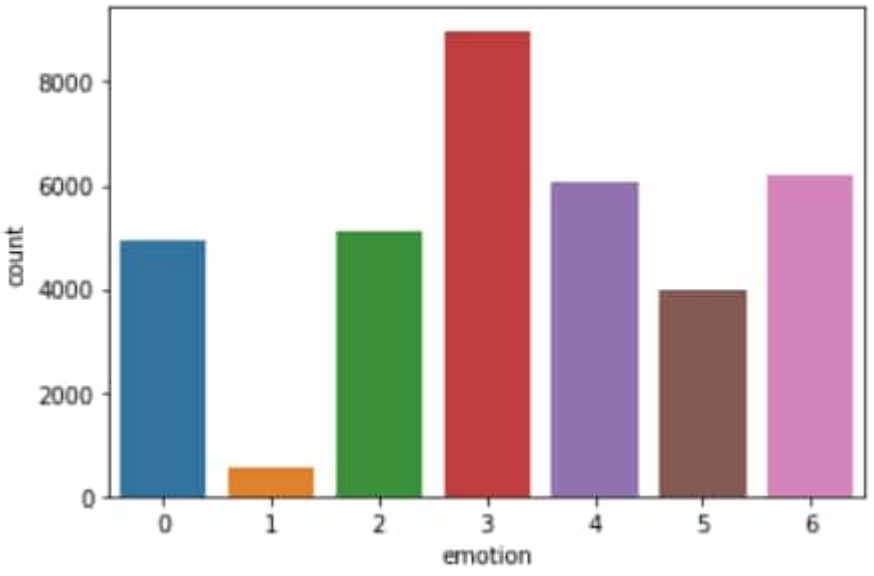

Рис 4.3. Кількість навчальних пар для кожної емоції, де 0: "гнів", 1: "огида", 2: "страх", 3: "щастя", 4: "сум", 5: "здивування", 6: "нейтральний".

Тож більшість класів належить до 3: Веселий, 4: Сумний і 6: Нейтральний, і нас також цікавлять лише ці три класи:

```
chosen_label = [3, 4, 6]
```

Обов’язковими попередніми процесами повинні бути розділення даних на набір для навчання та перевірки та нормалізація даних, оскільки нейронні мережі дуже чутливі до ненормалізованих даних:

```
train_x, test_x, train_y, test_y = train_test_split(elem_arr, elem_label, stratify=elem_label,
                                                    shuffle=True, random_state=42,test_size=0.1)
print("train x shape:",train_x.shape)
print("test x shape:",test_x.shape)
print("train y shape:",train_y.shape)
print("test y shape:",test_y.shape)
train_x = train_x / 255.
test_x = test_x / 255.
```

Реалізована нейронна мережа — це глибока згорткова нейронна мережа (DCNN). Я використав відсівання через рівні проміжки часу, з метою узагальнення .

Для уникнення проблем із відсіюванням relu, я використав `ELU` як функцію активації (рис 4.4). Також, у порівнянні із LeakyRelu, ELU працює значно краще для даної задачі.

Як ініціалізатор ядра, я обрав `he_normal`, оскільки він сумісний із обраною функцією активації. Для покращення результату, використовую нормалізацію BatchNormalization.

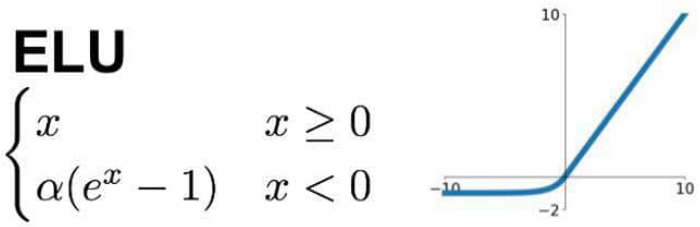

Рис 4.3. Функція активації ELU

Нейронна мережа реалізована з використанням фреймворку TensorFlow із застосуванням Keras API. Keras — це API, призначений для людей, а не для машин. Keras використовує практику для зменшення когнітивного навантаження: він пропонує зручні та прості API, мінімізує кількість дій користувачів і надає чіткі та дієві повідомлення про помилки. Він також містить розширену документацію та посібники для розробників.

Створений на основі TensorFlow 2, Keras є потужною галузевою структурою, яка може масштабуватися до великих кластерів графічних процесорів або цілого модуля TPU.

Keras є центральною частиною тісно пов’язаної екосистеми TensorFlow 2, яка охоплює кожен етап робочого процесу машинного навчання, керуючи даними до навчання гіпер-параметрів і рішень для розгортання.

Функціональний API Keras — це спосіб створення моделей, які є більш гнучкими, ніж tf.keras.SequentialAPI. Функціональний API може обробляти моделі з нелінійною топологією, спільними шарами та навіть кількома входами чи виходами.

Основна ідея полягає в тому, що модель глибокого навчання зазвичай є орієнтованим ациклічним графом (DAG) шарів. Отже, функціональний API є способом побудови графіків шарів.

У даній глибокій згортковій нейронній мережі я використовував два зворотних виклики, один — це `рання зупинка`, щоб уникнути переобладнання навчальних даних, а інший `ReduceLROnPlateau` для швидкості навчання.

```
call_EarlyStop = EarlyStopping(
    monitor='val_accuracy', patience=11, min_delta=0.00005,
```

```
    verbose=1,restore_best_weights=True, )

call_ReduceLR = ReduceLROnPlateau(
    monitor='val_accuracy', patience=7, factor=0.5,
    min_lr=1e-7, verbose=1, )

callbacks = [  call_EarlyStop, call_ReduceLR,  ]
```

У ролі оптимізатора, я спробував два алгоритми:«Адам» і «Надам». Щодо результатів, відчутньої різниці не було, тому обрав Nadam, як покращену версію Adam.

Історія епох (рис. 4.5) показує, що точність поступово зростає та досягає 83% точності як на тренувальному, так і на тестувальному наборі, але в кінці модель починає перенавчати тренувальні дані.

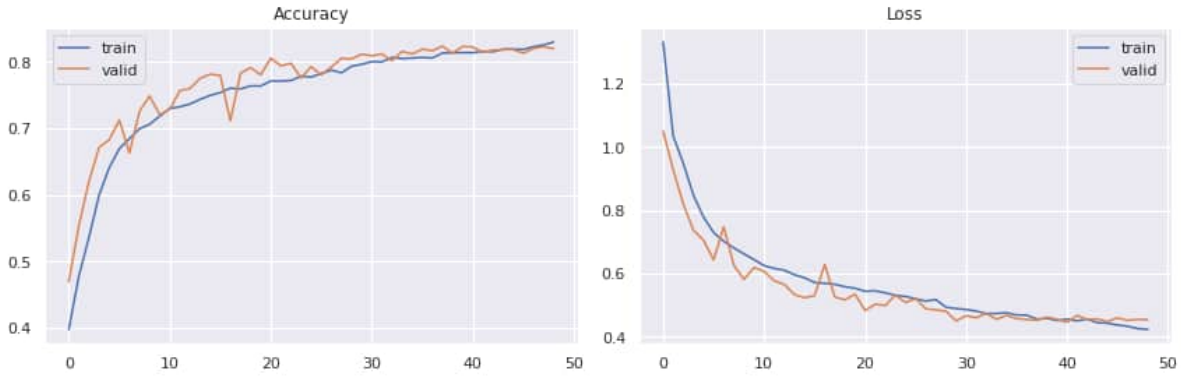

Рис 4.5. Графік історії навчання епох.

Загальна кількість неправильних прогнозів перевірки становить 374, а матриця плутанини (рис 4.6) у свою чергу чітко показує, що моя модель добре справляється з класом "щасливий", але її продуктивність низька для інших двох класів. Однією з причин цього може бути те, що ці два класи мають менше даних. Але коли я переглянув зображення, я виявив, що на деяких зображеннях із цих двох класів людині навіть важко визначити, сумна ця людина чи нейтральна. Вираз обличчя також залежить від людини. Чиєсь нейтральне обличчя виглядає сумним.

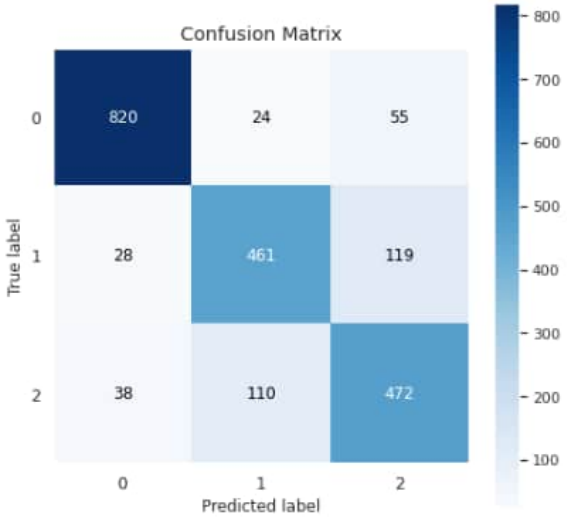

Рис 4.6. Матриця плутанини.

2. Тестування згорткової нейронної мережі

Тестування глибокої згорткової нейронної мережі відбувається за допомогою тих ж самих даних, за допомогою яких навчалась моя мережа. Вибірка генерується випадковим чином для отримання кращих результатів, також мною було прийнято рішення відкинути із тестування вибірку, яка включає в себе дані з класом "щасливий", оскільки матриця плутанини чітко показує, що розроблена мережа добре справляється з цією категорією.

```
np.random.seed(2)
sad_random = np.random.choice(np.where(test_y[:, 1]==1)[0], size=9)
neutral_random = np.random.choice(np.where(test_y[:, 2]==1)[0], size=9)

figures  = pyplot.figure(1, (18, 4))

for j, (sad_id, neut_id) in enumerate(zip(sad_random, neutral_random)):
        ax = pyplot.subplot(2, 9, j+1)
        test_elem = test_x[sad_id,:,:,0]
        ax.imshow(test_elem, cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(f"true:sad, pred:{emotion_types[model.predict_classes(test_elem.reshape(1,48,48,1))[0]]}")

        ax = pyplot.subplot(2, 9, j+10)
        test_elem = test_x[neut_id,:,:,0]
        ax.imshow(test_elem, cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(f"t:neutral, p:{emotion_types[model.predict_classes(test_elem.reshape(1,48,48,1))[0]]}")

        pyplot.tight_layout()
```

Рис 4.7 Результат тестування глибокої згорткової нейронної мережі

, де t (true) — істине значення з вибірки даних навчання, а p (pred) — результат оцінки нейронною мережею.

Серед значної кількості тестувань, на рисунку 4.7 я відобразив найгірший з отриманих результатів, оскільки тільки такі дані дають зрозуміти слабкі місця даної нейронної мережі. З рисунку 4.7 ми можемо помітити, що незадовільний результат отримано на зображеннях, де особисто людині важко розпізнати, яка саме емоція зображена на шаблоні.

ВИСНОВКИ

У магістерській роботі реалізована глибока згорткова нейронна мережа (DCNN) за допомогою фреймворку мови високого рівня Python TensorFlow на ядрі Keras API. У якості середовища розробки було використано Jupyter Notebook, оскільки він дає можливість візуалізувати проміжні результати виконання програми без додаткових затримок та накладань.

У роботі розглянуто різні види нейронних мереж та області їх застосування. Для розпізнання обличчя та емоцій, краще всього підходить згорткова нейронна мережа, оскільки обличчя ідентифікується із зображень, зібраних у режимі реального часу на етапі виділення ознак. Щоб згрупувати зображення в один із семи класів, етап класифікації емоцій складається із застосування алгоритму CNN. У випадку згорткової нейронної мережі головною метою згортки є виділення ознак із вхідного зображення.

Також, було розглянуто процес реалізації згорткової нейронної мережі та мови програмування, за допомогою яких можна розробити подібні проекти. При усіх перевагах інших мов, я зупинився на мові високого рівня з відкритим кодом Python за рахунок її переваг та низки фреймворків і бібліотек, які значно полегшують і пришвидшують розробку системи штучного інтелекту. Усі використані бібліотеки я наводжу у розділі 3.

У якості функції активації згорткової нейронної мережі виступає функція активації ELU. З метою узагальнення я використав відсіювання через рівні проміжки часу та функцію оптимізації Надам. Усі використані фреймворки, бібліотеки, пакети та матеріали, результати виконання та оцінка ефективності глибокої згорткової нейронної мережі наведені у роботі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Xie Y. Deep Learning for Natural Language Processing. In Handbook of Statistics / Y. Xie, L. Le, Y. Zhou, V. Raghavan // Elsevier: Amsterdam, The Netherlands, 2018.
2. Kumar R. Natural language processing. Machine Learning and Cognition in Enterprises / R. Kumar // Springer: Berlin, Germany, 2017; pp. 65–73.
3. Rojas R. Neural Networks: A Systematic Introduction / R. Rojas // Springer: Berlin, Germany, 2013.
4. Karpathy A. CS231n Convolutional Neural Networks for Visual Recognition / A. Karpathy // 2018.
5. Boureau Y.-L. A theoretical analysis of feature pooling in visual recognition / Y.-L. Boureau, J. Ponce, Y. LeCun //
6. Fürnkranz J. In Proceedings of the 27th International Conference on Machine Learning / J. Fürnkranz, J. Thorsten // Haifa, Israel, 21–24 June 2010; pp. 111–118.
7. Scherer D. Evaluation of pooling operations in convolutional architectures for object recognition / D. Scherer, A. Müller, S. Behnke // In Proceedings of the International Conference on Artificial Neural Networks, Thessaloniki, Greece, 15–18 September 2010; pp. 92–101.
8. Wu H. Max-pooling dropout for regularization of convolutional neural networks / H. Wu, X. Gu // In Proceedings of the International Conference on Neural Information Processing, Istanbul, Turkey, 9–12 November 2015;
9. Nwosu L. Deep convolutional neural network for facial expression recognition using facial parts / L. Nwosu, H. Wang, J. Lu, I. Unwala, X. Yang, T. Zhang // In 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/Cyber SciTech) 1318–1321 (IEEE, 2017).

10. Yang B. 3D palm print recognition using shape index representation and fragile bits / B. Yang, X. Xiang, D. Xu, X. Wang, X. Yang // Multimed. Tools Appl. 76(14), 15357–15375 (2017).
11. Kumar N. A scheme of features fusion for facial expression analysis: A facial action recognition / N. Kumar, D. Bhargava // J. Stat. Manag. Syst. 20(4), 693–701 (2017).
12. Zhang H. A face emotion recognition method using convolutional neural network and image edge computing / H. Zhang, A. Jolfaei, M. Alazab // IEEE Access 7, 159081–159089 (2019).
13. Mayer C. Cross-database evaluation for facial expression recognition. Pattern Recognit / C. Mayer, M. Eggers, B. Radig // Image Anal. 24(1), 124–132 (2014).
14. Gan Y. Facial expression recognition using convolutional neural network /Y. Gan // In Proceedings of the 2nd International Conference on Vision, Image and Signal Processing 1–5 (2018).
15. Chollet F. Deep learning with depth wise separable convolutions / F. Chollet // arXiv preprint arXiv:1610.02357, 2016.
16. Ciresan D. Multi-column deep neural networks for image classification / D. Ciresan, U. Meier, J. Schmidhuber // CoRR, abs/1202.2745, 2012.
17. Canziani A. An analysis of deep neural network models for practical applications / A. Canziani, A. Paszke, E. Culurciello.. 05 2016.
18. Chung J. Simplifying deep neural networks for neuromorphic architectures / J. Chung, T. Shin // In Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE, pages 1–6. IEEE, 2016.
19. Chen W. Compressing convolutional neural networks / W. Chen, J. T. Wilson, S. Tyree, K. Q Weinberger, Y. Chen // arXiv preprint arXiv:1506.04449, 2015.
20. Bengio Y. Deep Learning / Y. Bengio, A. Courville, I. Goodfellow // MIT Press, 2016. 800 c.

21. Denton E. Exploiting linear structure within convolutional networks for efficient evaluation / E. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus // In Advances in Neural Information Processing Systems, pages 1269–1277, 2014.
22. Szegedy C. Deep neural networks for object detection / C. Szegedy, A. Toshev, D. Erhan. // NIPS. – 2013.
23. Hoai M. Regularized max pooling for image categorization / Hoai. // BMVC. – 2014.
24. Girshick R. Fast R-CNN / Girshick. // ICCV. – 2015.
25. Krizhevsky A. Imagenet classification with deep convolutional neural networks / A. Krizhevsky, I. Sutskever, G. E. Hinton. // Advances in neural information processing systems. – 2012.
26. Vedaldi A. Convolutional neural networks for matlab / A. Vedaldi, K. Lenc. // Proceedings of the 23rd Annual ACM Conference on Multimedia Conference. – 2015.
27. Dai J. Convolutional feature masking for joint object and stuff segmentation / J. Dai, K. He, J. Sun. // CVPR. – 2015. Szegedy C. Going deeper with convolutions / C. Szegedy, W. Liu, Y. Jia. // CVPR. – 2015.
28. Long J. Fully convolutional networks for semantic segmentation / J. Long, E. Shelhamer, T. Darrell. // CoRR. – 201
29. Deep J. Programming : 4 Books in 1: Python Programming and Crash Course, Machine Learning for Beginners Python Machine Learning / J. Deep // Independently Published, 2020. 502 p.
30. Ketkar N. Introduction to Tensorflow. Deep Learning with Python / N. Ketkar // Berkeley, CA, 2017. P. 159–194. URL: https://doi.org/10.1007/978-1-4842-2766-4_11 (date of access: 28.10.2022).
31. Manaswi N. K. Basics of TensorFlow. Deep Learning with Applications Using Python / N. K. Manaswi // Berkeley, CA, 2018. P. 1–30. URL: https://doi.org/10.1007/978-1-4842-3516-4_1 (date of access: 28.10.2022).
32. NumPy documentation. URL: https://numpy.org/.
33. Pandas documentation. URL: https://pandas.pydata.org/docs/.

34. Goodfellow I. Deep learning / I. Goodfellow, Y. Bengio, A. Courville // The MIT Press, 2016, 800 pp, ISBN: 0262035618
35. Viola P. Rapid object detection using a boosted cascade of simple features / P. Viola, M. Jones //in Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001, vol. 1. IEEE, 2001, pp. I–I.

ДОДАТОК

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


get_ipython().system('pip install pandas')
get_ipython().system('pip install numpy')


# In[2]:


import math
import pandas as pd
import numpy as np
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

import scikitplot
from matplotlib import pyplot
from keras.utils import np_utils

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D
```

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dropout, BatchNormalization, LeakyReLU, Activation
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import Callback, EarlyStopping, ReduceLROnPlateau



# In[3]:



dataset = pd.read_csv('fer2013.csv')
print("Shape of dataset:",dataset.shape)
dataset.head()



# In[4]:



print("Unique emotions:")
dataset.emotion.unique()



# In[5]:



set_labels = {0:'anger', 2:'fear', 4: 'sadness', 6: 'neutral', 3:'happiness', 5: 'surprise', 1:'disgust'}
```

```
# In[6]:

print("Number of elements for each emotion:")
dataset.emotion.value_counts()


# In[7]:


sns.countplot(dataset.emotion)
print("Histogram of elements count:")
pyplot.show()


# In[8]:


math.sqrt(len(dataset.pixels[0].split(' ')))


# In[9]:


figures  = pyplot.figure(1, (13, 13))
i = 0
for unique_labels in sorted(dataset.emotion.unique()):
    for l in range(7):
        pix = dataset[dataset.emotion==unique_labels].pixels.iloc[i]
        pix = np.array(pix.split(' ')).reshape(48, 48).astype('float32')
        i += 1
```

```
        elem = pyplot.subplot(7, 7, i)
        elem.imshow(pix, cmap='gray')
        elem.set_xticks([])
        elem.set_yticks([])
        elem.set_title(set_labels[unique_labels])
        pyplot.tight_layout()



# In[10]:


chosen_label = [3, 4, 6]


# In[11]:


dataset = dataset[dataset.emotion.isin(chosen_label)]
print("Shape of chosen labels dataset:",dataset.shape)



# In[12]:


elem_arr = dataset.pixels.apply(lambda i: np.array(i.split(' ')).reshape(48, 48,
1).astype('float32'))
elem_arr = np.stack(elem_arr, axis=0)
```

```
# In[13]:


print("Shape of image array :",elem_arr.shape)


# In[14]:


lab_enc = LabelEncoder()
elem_label = lab_enc.fit_transform(dataset.emotion)
elem_label = np_utils.to_categorical(elem_label)
print("Shape of image labels:",elem_label.shape)



# In[15]:


lab_enc_mapping = dict(zip(lab_enc.classes_, lab_enc.transform(lab_enc.classes_)))
print("Label encoder name mapping:",lab_enc_mapping)



# In[16]:


train_x, test_x, train_y, test_y = train_test_split(elem_arr, elem_label,
stratify=elem_label,
                                                    shuffle=True, random_state=42,test_size=0.1)

print("train x shape:",train_x.shape)
print("test x shape:",test_x.shape)
```

```
print("train y shape:",train_y.shape)
print("test y shape:",test_y.shape)



# In[17]:


del dataset
del elem_arr
del elem_label



# In[18]:


elem_width = train_x.shape[1]
elem_height = train_x.shape[2]
elem_depth = train_x.shape[3]
num_classes = train_y.shape[1]



# In[19]:


train_x = train_x / 255.
test_x = test_x / 255.
```

```
# In[20]:


def create_dcnn(dcnnopt):

    dcnn1 = Sequential(name='DCNN')

    dcnn1.add(
        Conv2D(
            kernel_size=(5,5),
            filters=64,
            input_shape=(elem_width, elem_height, elem_depth),
            activation='elu',
            kernel_initializer='he_normal',
            padding='same',
            name='2dconv_1'
        )
    )
    dcnn1.add(BatchNormalization(name='batch_norm_1'))
    dcnn1.add(
        Conv2D(
            kernel_size=(5,5),
            filters=64,
            activation='elu',
            kernel_initializer='he_normal',
            padding='same',
            name='2dconv_2'
        )
    )
    dcnn1.add(BatchNormalization(name='batch_norm_2'))
```

```
dcnn1.add(MaxPooling2D(pool_size=(2,2), name='maxpool_1'))
dcnn1.add(Dropout(0.4, name='dropout_1'))

dcnn1.add(
    Conv2D(
        kernel_size=(3,3),
        filters=128,
        activation='elu',
        kernel_initializer='he_normal',
        padding='same',
        name='2dconv_3'
    )
)
dcnn1.add(BatchNormalization(name='batch_norm_3'))
dcnn1.add(
    Conv2D(
        kernel_size=(3,3),
        filters=128,
        activation='elu',
        kernel_initializer='he_normal',
        padding='same',
        name='2dconv_4'
    )
)
dcnn1.add(BatchNormalization(name='batch_norm_4'))

dcnn1.add(MaxPooling2D(pool_size=(2,2), name='maxpool_2'))
dcnn1.add(Dropout(0.4, name='dropout_2'))

dcnn1.add(
```

```
    Conv2D(
        kernel_size=(3,3),
        filters=256,
        activation='elu',
        kernel_initializer='he_normal',
        padding='same',
        name='2dconv_5'
    )
)
dcnn1.add(BatchNormalization(name='batch_norm_5'))
dcnn1.add(
    Conv2D(
        kernel_size=(3,3),
        filters=256,
        activation='elu',
        kernel_initializer='he_normal',
        padding='same',
        name='2dconv_6'
    )
)
dcnn1.add(BatchNormalization(name='batch_norm_6'))

dcnn1.add(MaxPooling2D(pool_size=(2,2), name='maxpool_3'))
dcnn1.add(Dropout(0.5, name='dropout_3'))

dcnn1.add(Flatten(name='flatten'))

dcnn1.add(
    Dense(
        128,
        activation='elu',
```

```
            kernel_initializer='he_normal',
            name='dense_1'
        )
    )
    dcnn1.add(BatchNormalization(name='batch_norm_7'))

    dcnn1.add(Dropout(0.6, name='dropout_4'))

    dcnn1.add(
        Dense(
            num_classes,
            activation='softmax',
            name='out_layer'
        )
    )

    dcnn1.compile(
        optimizer=dcnnopt,
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    dcnn1.summary()

    return dcnn1



# In[21]:
```

```
call_EarlyStop = EarlyStopping(
    monitor='val_accuracy',
    patience=11,
    min_delta=0.00005,
    verbose=1,
    restore_best_weights=True,
)

call_ReduceLR = ReduceLROnPlateau(
    monitor='val_accuracy',
    patience=7,
    factor=0.5,
    min_lr=1e-7,
    verbose=1,
)

callbacks = [
    call_EarlyStop,
    call_ReduceLR,
]



# In[22]:

datagen_train = ImageDataGenerator(
    height_shift_range=0.15,
    width_shift_range=0.15,
    rotation_range=15,
    zoom_range=0.15,
    shear_range=0.15,
```

```
    horizontal_flip=True,
)
datagen_train.fit(train_x)



# In[31]:



batch_size = 32 #batch розмір 32 працює найкраще.
epochs = 100
chosen_opt = [
    optimizers.Adam(0.001),
    optimizers.Nadam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07,
name='Nadam'),
]

model = create_dcnn(chosen_opt[1])
history = model.fit_generator(
    datagen_train.flow(train_x, train_y, batch_size=batch_size),
    validation_data=(test_x, test_y),
    steps_per_epoch=len(train_x) / batch_size,
    epochs=epochs,
    callbacks=callbacks,
    use_multiprocessing=True
)



# In[32]:
```

```
model_yaml = model.to_yaml()
with open("model.yaml", "w") as yaml_file:
    yaml_file.write(model_yaml)

model.save("model.h5")



# In[33]:

sns.set()
figures  = pyplot.figure(0, (14, 5))

ax = pyplot.subplot(1, 2, 1)
sns.lineplot(history.epoch, history.history['accuracy'], label='train')
sns.lineplot(history.epoch, history.history['val_accuracy'], label='test')
pyplot.title('Точність')
pyplot.tight_layout()

ax = pyplot.subplot(1, 2, 2)
sns.lineplot(history.epoch, history.history['loss'], label='train')
sns.lineplot(history.epoch, history.history['val_loss'], label='test')
pyplot.title('Втрати')
pyplot.tight_layout()

pyplot.savefig('network_epoch_history.png')
pyplot.show()
```

```
# In[34]:


dataset_accu = pd.DataFrame({'train': history.history['accuracy'], 'test':
history.history['val_accuracy']})
dataset_loss = pd.DataFrame({'train': history.history['loss'], 'test':
history.history['val_loss']})


figures  = pyplot.figure(0, (14, 5))
ax = pyplot.subplot(1, 2, 1)
sns.violinplot(x="variable", y="value", data=pd.melt(dataset_accu),
showfliers=False)
pyplot.title('Точність')
pyplot.tight_layout()

ax = pyplot.subplot(1, 2, 2)
sns.violinplot(x="variable", y="value", data=pd.melt(dataset_loss), showfliers=False)
pyplot.title('Втрати')
pyplot.tight_layout()

pyplot.savefig('network_performance.png')
pyplot.show()



# In[35]:



mod_pred = model.predict_classes(test_x)
scikitplot.metrics.plot_confusion_matrix(np.argmax(test_y, axis=1), mod_pred,
figsize=(7,7))
pyplot.savefig("dcnn_confusion_mat.png")
```

```
print(f'Кількість помилкових передбачень: {np.sum(np.argmax(test_y, axis=1) != mod_pred)}\n\n')
print(classification_report(np.argmax(test_y, axis=1), mod_pred))



# In[36]:



emotion_types = {
   0: "happy", 1: "sad", 2: "neutral",
}



# In[37]:



np.random.seed(2)
sad_random = np.random.choice(np.where(test_y[:, 1]==1)[0], size=9)
neutral_random = np.random.choice(np.where(test_y[:, 2]==1)[0], size=9)

figures  = pyplot.figure(1, (18, 4))

for j, (sad_id, neut_id) in enumerate(zip(sad_random, neutral_random)):
      ax = pyplot.subplot(2, 9, j+1)
      test_elem = test_x[sad_id,:,:,0]
      ax.imshow(test_elem, cmap='gray')
      ax.set_xticks([])
      ax.set_yticks([])
      ax.set_title(f"true:sad,
pred:{emotion_types[model.predict_classes(test_elem.reshape(1,48,48,1))[0]]}")
```

```
        ax = pyplot.subplot(2, 9, j+10)
        test_elem = test_x[neut_id,:,:,0]
        ax.imshow(test_elem, cmap='gray')
        ax.set_xticks([])
        ax.set_yticks([])
        ax.set_title(f"t:neutral,
p:{emotion_types[model.predict_classes(test_elem.reshape(1,48,48,1))[0]]}")

        pyplot.tight_layout()
```